Étude des Techniques de Deep Learning en Reconnaissance Automatique du Locuteur

Soufiane Hourri

جامعة سيدي محمد بن عبد الله بفاس
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ ⴱ ⴼⴰⵙ
UNIVERSITÉ SIDI MOHAMED BEN ABDELLAH DE FES

كلية العلوم والتقنيات فاس
ⵜⴰⵖⵡⵍⵓⵜ ⵏ ⵜⵣⵣⵉⵡⵉⵏ ⴷ ⵜⵙⵙⵏⵜⵉⵏ - ⴼⴰⵙ
FACULTÉ DES SCIENCES ET TECHNIQUES DE FÈS

Centre d'Etudes Doctorales : Sciences et Techniques de l'Ingénieur

N° d'ordre 20/2021

THESE DE DOCTORAT

Présentée par

Mr : Soufiane Hourri

Spécialité : Informatique

Sujet de la thèse : Étude des Techniques de Deep Learning en Reconnaissance Automatique du Locuteur

Thèse présentée et soutenue le Lundi 22 Février 2021 devant le jury composé de :

Prénom Nom	Titre	Etablissement	
Aicha MAJDA	PES	Faculté des Sciences et Techniques de Fès	Présidente
Ahmed BEN HAMIDA	PES	Ecole Nationale d'Ingénieurs de Sfax	Rapporteur
Mohamed BAHAJ	PES	Faculté des Sciences et Techniques de Settat	Rapporteur
Abderrahim BEN ABBOU	PES	Faculté des Sciences et Techniques de Fès	Rapporteur
Khalid ABBAD	PH	Faculté des Sciences et Techniques de Fès	Examinateur
Jamal KHARROUBI	PES	Faculté des Sciences et Techniques de Fès	Directeur de thèse

Laboratoire d'accueil : Systèmes Intelligents et Applications (LSIA)

Etablissement : Faculté des Sciences et Techniques de Fès

Cette thèse est dédiée à mes grands-parents

Remerciements

Aujourd'hui, j'ai remis mon rapport de thèse de doctorat. Ce fut un voyage long et parfois décourageant au cours des quatre dernières années et deux mois. Cela n'aurait pas été possible sans le soutien de nombreuses personnalités.

Cette thèse n'aurait pas été possible sans l'inspiration et le soutien d'un certain nombre de personnes formidables — je les remercie et les apprécie tous pour avoir participé à ce voyage et rendu cette thèse possible. Je suis très reconnaissant envers mon directeur de thèse, le professeur Jamal Kharroubi. Sans son enthousiasme, ses encouragements, son soutien et son optimisme permanent, cette thèse n'aurait guère pu être achevée.

Je remercie le Professeur Nikola S. Nikolov, qui a été mon encadrant pendant mon séjour en Irlande en tant que doctorant invité à l'université de Limerick, pour sa contribution.

Je suis à jamais reconnaissant à mes collègues du Laboratoire des Systèmes Intelligents et Applications pour leur amitié et leur soutien, et pour avoir créé un environnement de travail cordial.

Je tiens également à remercier le reste du comité de thèse : Professeur Aicha Majda, Professeur Ahmed Ben Hamida, Professeur Mohamed Bahaj, Professeur Khalid Abbad, et Professeur Abderahim Ben Abbou pour leurs commentaires perspicaces et leurs encouragements, et aussi pour toutes les questions qui m'ont incité à élargir encore plus mes recherches à l'avenir.

Enfin, ma profonde et sincère gratitude à ma famille pour leur amour, leur aide et leur soutien continus et sans égal. Je suis à jamais redevable à tous les membres de ma grande famille de m'avoir donné les opportunités et les expériences qui ont fait de moi ce que je suis. Ils m'ont encouragée de manière désintéressée à explorer de nouvelles directions dans la vie et à chercher mon propre destin. Ce voyage n'aurait pas été possible sans eux, et je leur dédie cette étape importante.

Abstract

Speaker Recognition (SRE), sometimes referred to as speaker biometrics, involves the identification, verification (authentication), classification and, by extension, segmentation, tracking and detection of speakers. It is a generic term used for any procedure involving the recognition of a person's identity based on their voice. In this context, deep learning has attracted much more interest from speech processing researchers, and it has recently been introduced in SRE. In most cases, deep learning models are adapted from automatic speech recognition (ASR) applications and applied to SRE, and have shown their ability to compete with state-of-the-art approaches. Nevertheless, the use of deep learning in SRE is still linked to ASR. On the other hand, deep learning models are now considered state-of-the-art in many areas of pattern recognition. In SRE, several architectures have been studied, such as deep neural networks (DNNs), deep belief networks (DBNs), restricted Boltzmann machines (RBMs), etc., while convolutional neural networks (CNNs) are the most widely used models in image processing.

The objective of this thesis is to study deep learning models for the SRE domain. For this reason, we proposed a new way to use DBNs and DNNs in SRE, with the aim to extract the deep speaker features (DeepSF). Subsequently, we proposed a new usage of CNNs for the SRE problem. Although they are particularly designed for image processing problems, CNNs have recently been applied to SRE using spectrograms as input images. We believe that this approach is not optimal because it can lead to two cumulative errors in solving image processing and SRE problems. This is why we have developed a new method that allows the use of CNNs without using images.

The results of the thesis represent an important finding to understand how deep learning models can be adapted to the problem of speaker recognition.

Résumé

La reconnaissance automatique du locuteur (RAL), parfois appelée biométrie du locuteur, implique l'identification, la vérification (authentification), la classification et, par extension, la segmentation, le suivi et la détection des locuteurs. Il s'agit d'un terme générique utilisé pour toute procédure impliquant la reconnaissance de l'identité d'une personne sur la base de sa voix. Dans ce contexte, le deep learning a suscité beaucoup plus d'intérêt de la part des chercheurs en traitement de la parole, et il a été introduit récemment dans la RAL. Dans la plupart des cas, les modèles de deep learning sont adaptés des applications de reconnaissance automatique de la parole (RAP) et appliqués à la RAL, et ils ont montré leur capacité à concurrencer les approches de l'état de l'art. Néanmoins, l'utilisation de deep learning dans la RAL est toujours liée à la RAP. D'autre part, les modèles de deep learning sont maintenant considérés comme état de l'art dans nombreux domaines de la reconnaissance des formes. En RAL, plusieurs architectures ont été étudiées, telles que les réseaux de neurones profonds (DNN), les réseaux de croyances profonds (DBN), les machines de Boltzmann restreintes (RBM), etc. tandis que les réseaux de neurones convolutifs (CNN) sont les modèles les plus utilisés en traitement d'images.

L'objectif de cette thèse est d'étudier les modèles de deep learning pour le domaine de la RAL. Pour cette raison, nous avons proposé une nouvelle façon d'utiliser les DBN et les DNN dans la RAL, dans le but d'extraire les caractéristiques profondes du locuteur (DeepSF). Par la suite, nous avons proposé une nouvelle utilisation des CNN pour le problème de la RAL. Bien qu'ils soient particulièrement conçus pour les problèmes de traitement d'images, les CNN ont récemment été appliqués à la RAL en utilisant des spectrogrammes comme images d'entrée. Nous pensons que cette approche n'est pas optimale car elle peut entraîner deux erreurs cumulatives dans la résolution d'un problème de traitement d'images et de RAL. C'est pourquoi nous avons développé une nouvelle méthode qui permet d'utiliser les CNN sans utiliser d'images.

Les résultats de la thèse représentent une découverte importante pour comprendre comment les modèles de deep learning peuvent être adaptés au problème de la RAL.

Sommaire

Liste des Figures

Liste des Tableaux

Liste des Abréviations

DNN . . . . . Deep Neural Network

CNN . . . . . Convolutional Neural Network

BM . . . . . Boltzmann Machine

RBM . . . . . Restricted Boltzmann Machine

GB-RBM . . . Gaussian-Bernoulli Restricted Boltzmann Machine

BB-RBM . . . Bernoulli-Bernoulli Restricted Boltzmann Machine

DBN . . . . . Deep Belief Network

MLP . . . . . Multi-Layer Perceptron

ANN . . . . . Artificial Neural Network

ReLU . . . . . Rectified Linear Unit

SGD . . . . . Stochastic Gradient Descent

IA . . . . . Intelligence Artificielle

CD . . . . . Contrastive Divergence

TDNN . . . . Time Delay Neural Network

BN . . . . . Bottleneck

DB . . . . . Decibel

MFCC . . . . . Mel Frequency Cepstral Coefficients

CMVN . . . . Cepstral Mean and Variance Normalization

RAL . . . . . Reconnaissance Automatique du Locuteur

VAL . . . . . Verification Automatique du Locuteur

IAL . . . . . Identification Automatique du Locuteur

GMM . . . . . Gaussian Mixture Model

UBM . . . . . Universal Background Model

SVM . . . . . Support Vector Machine

DTW . . . . . Dynamic Time Warping

FFT . . . . . Fast Fourier Transform

VQ . . . . . Vector Quantization

FA . . . . . Factor Analysis

JFA . . . . . Joint Factor Analysis

***i*-vector** . . . . Identification Vector

PLDA . . . . . Probabilistic Linear Discriminant Analysis

LDA . . . . . Linear Discriminant Analysis

DET . . . . . Detection Error Tradeoff

EER . . . . . Equal Error Rate

ROC . . . . . Receiver Operating Characteristic

FAR . . . . . False Acceptance Rate

FRR . . . . . False Rejection Rate

minDCF . . . Minimum Cost Function

DCF . . . . . Decision Cost Function

MAP . . . . . Maximum a Posteriori Probability

EM . . . . . Expectation Maximization

NearC . . . . . The Nearest Cluster

DeepSF . . . . Deep Speaker Features

ConvVectors . Convolutional Vectors

IDL . . . . . Institute of Deep Learning

NIST . . . . . National Institute of Standards and Technology

SNR . . . . . Signal to Noise Ratio

IRM . . . . . Imagerie par Résonance Magnétique

GPU . . . . . Graphics Processing Unit

CPU . . . . . Central Processing Unit

GPL . . . . . General Public License

PC . . . . . Personal Computer

OS . . . . . Operating System

RAM . . . . . Random Access Memory

TM . . . . . Trade Mark

Partie I

Introduction

1 Introduction

Sommaire

1.1 Contexte

Depuis longtemps, les humains rêvent de créer des machines intelligentes. L'intelligence humaine ne se manifeste qu'à travers le pouvoir de communication et d'échange de l'information entre les humains. Cette communication n'est possible qu'à travers la parole. Dans les années soixante-dix, la reconnaissance automatique vocale était considérée comme de la science-fiction, mais l'origine de ce rêve a commencé beaucoup plus tôt dans les mythes de la Grèce antique qui parlait des objets intelligents comme des statues d'êtres humains animées et les tables qui exécutent des ordres et qui arrivent pleines de nourriture et de boissons.

Lorsque les ordinateurs programmables ont été conçus, les chercheurs se sont demandé s'ils pouvaient devenir intelligents. Il faudrait plusieurs années avant de pouvoir en construire un, dont on pourrait dire qu'il est intelligent. Aujourd'hui,

l'intelligence artificielle (IA) est un domaine florissant, avec de nombreuses applications pratiques et des sujets de recherche actuels passionnants. Nous nous tournons vers des solutions intelligentes pour automatiser le traitement de routine, c'est-à-dire pour traiter des informations provenant de différentes sources, principalement la parole et les images.

Aux débuts, l'intelligence artificielle a rapidement abordé et résolu des problèmes qui sont intellectuellement difficiles pour les êtres humains mais relativement simples pour les ordinateurs, principalement les problèmes qui peuvent être décrits par une liste de règles formelles et mathématiques. Par conséquence, le véritable défi pour l'intelligence artificielle s'est avéré être la résolution des problèmes difficiles à décrire formellement, comme la reconnaissance des mots parlés, de visages en images, ou de personnes à partir de leurs voix. Ce type de problèmes que l'être humain peut résoudre intuitivement.

Il existe une solution à ce type de problèmes qui permet aux ordinateurs d'apprendre par expérience et de comprendre le monde selon une hiérarchie de concepts où chaque concept est lié à des sous-concepts plus simples. Le rassemblement de ces connaissances sous forme de graphe hiérarchique permet de mettre en évidence le lien indirect entre celles-ci. Si nous dessinons un graphique montrant comment ces connaissances sont construites, on constate que ce graphique est profond avec de nombreuses couches. C'est pour cette raison qu'on appel ce concept de l'intelligence artificielle : *deep learning* (apprentissage profond).

Le deep learning découle d'une approche appelée "connexionnisme", une approche qui vise à rendre compte de la cognition humaine par le biais de réseaux de neurones artificiels. D'autre part, la reconnaissance automatique du locuteur est considérée comme un problème de reconnaissance de formes qui pourrait bénéficier des techniques de deep learning. De nos jours, elle a suscité un grand intérêt en raison de la forte demande d'applications d'accès vocal et de sécurité. Prenons l'exemple de certaines banques qui utilisent cette technique comme principal moyen d'authentification des clients de leurs centres d'appels. En effet, le système de reconnaissance automatique du locuteur essaye de vérifier l'identité des clients par téléphone en quelques secondes à travers une conversation normale. Pour ce faire, le système de reconnaissance doit utiliser des techniques d'apprentissage automatique robustes et rapides.

Dans cette thèse, nous étudions différentes techniques de deep learning dans le domaine de la reconnaissance automatique du locuteur.

1.2 Questions de recherche

Dans cette section, nous relevons les questions liées à cette étude dont les réponses constituent l'essentiel du travail que nous présentons dans ce document:

1. L'extraction de caractéristiques est la première phase d'un système de reconnaissance automatique du locuteur; cette phase consiste à représenter une petite durée de signal vocal par un vecteur appelé vecteur caractéristique. Le problème qui apparaît est lorsque plusieurs locuteurs ont des vecteurs caractéristiques similaires, par exemple le vecteur caractéristique qui représente le silence. Les questions que l'on peut se poser sont les suivantes : Pourrait-on extraire les caractéristiques spécifiques du locuteur ? Pourrait-on utiliser la distribution et la répartition des vecteurs caractéristiques communs dans les vecteurs caractéristiques du locuteur comme une caractéristique ?

2. La vérification automatique du locuteur est une branche de la reconnaissance automatique du locuteur qui consiste à comparer un segment de parole à un modèle de locuteur. Les approches traditionnelles utilisent le modèle dit *modèle générique des locuteurs* (*UBM: Universal Backgound Model*) afin de modéliser une large population de locuteurs, puis d'en déduire le modèle de locuteur cible. Ensuite, le segment de test est comparé au modèle UBM et au modèle du locuteur, et voir si le segment et proche du modèle de locuteur ou bien du modèle UBM. Le problème ici est que nous avons besoin d'une énorme quantité de données pour modéliser les locuteurs. Les questions qui se posent sont les suivantes : serait-il possible de modéliser les locuteurs sans utiliser une énorme quantité de données ? Pourrait-on modéliser les locuteurs sans s'appuyer sur un modèle de générique des locuteurs ?

3. L'objectif de la thèse est d'intégrer des modèles de deep learning dans la reconnaissance automatique du locuteur. L'un des modèles les plus populaires est celui des réseaux de neurones convolutifs (CNN). Ils ont été utilisés dans la reconnaissance d'images et sont considérés comme la méthode de pointe. Cependant, la nature des réseaux CNN, qui ne sont conçus que pour des données bidimensionnelles, rend difficile leur intégration dans plusieurs domaines de la reconnaissance des formes. Les réseaux CNN ont été intégrés dans la reconnaissance automatique de la parole, mais en se reposant sur des spectrogrammes et des formes d'onde, qui sont une représentation de la parole sous forme d'image. Les questions que nous pouvons poser sont les suivantes : pourrions-nous intégrer les réseaux CNN dans la reconnaissance automatique

du locuteur sans utiliser d'images ? Si oui, comment construire les données bidimensionnelles ? Pourrions-nous représenter les caractéristiques spécifiques du locuteur en utilisant des réseaux CNN ?

1.3 Contributions

Dans cette thèse, nous présentons la liste des contributions par chapitre/article :

➥ Les contributions du **chapitre 4** (Bouziane, A., Kadi, H., Hourri, S., & Kharroubi, J, 2016) [1] :

- La proposition d'un nouveau corpus multilingue pour la reconnaissance automatique du locuteur.

➥ Les contributions du **chapitre 5** (Hourri, S. & Kharroubi, J, 2019) [2] :

- La proposition d'une nouvelle méthode de mesure de similarité pour la vérification automatique du locuteur ;
- La proposition d'une nouvelle méthode pour extraire les caractéristiques spécifiques du locuteur.

➥ Les contributions du **chapitre 6** (Hourri, S. & Kharroubi, J, 2019) [3] :

- La proposition d'un nouvel usage des réseaux de neurones profonds dans la reconnaissance automatique du locuteur ;
- La propositions de nouveaux vecteurs caractéristiques (DeepSF).

➥ Les contributions du **chapitre 7** (Hourri, S., Nikolov, N.S. & Kharroubi, J., 2020) [4] :

- La présentation des premiers travaux de reconnaissance automatique des locuteurs à utiliser CNN sans spectrogramme ni forme d'onde ;
- La proposition d'une nouvelle méthode pour créer des locuteurs cibles et non-cibles ;
- La proposition d'une nouvelle façon d'utiliser l'UBM pour générer des données du locuteur.

➥ Les contributions du **chapitre 8** (Hourri, S., Nikolov, N.S. & Kharroubi, J., 2021) [5] :

- La proposition d'une approche originale pour extraire les caractéristiques du locuteur en formant des filtres pour le réseau CNN correspondant à ce locuteur ;

- La proposition de nouveaux vecteurs pour identifier les locuteurs : ConvVectors.

1.4 Développement informatique des méthodes proposées

Durant ce travail de thèse, nous avons développé toutes les méthodes et algorithmes présentés dans ce rapport au sein de notre laboratoire des systèmes intelligents et applications (LSIA), en utilisant le langage de programmation Python, version 3.6. En effet, nous avons développé des extracteurs de caractéristiques pour transformer le signal vocal "wav" en trames (vecteurs caractéristiques), les méthodes de l'état de l'art auxquelles nous avons comparé les méthodes que nous avons proposé et tous les modèles de deep learning en utilisant deux Frameworks : TensorFlow et Keras.

1.4.1 TensorFlow

TensorFlow[1] est un Framework de deep learning de bout en bout, open-source, développé par Google et publié en 2015. Il est connu pour sa documentation et son support de formation, ses options de production et de déploiement évolutives, ses multiples niveaux d'abstraction et sa prise en charge de différentes plateformes, telles qu'Android.

TensorFlow est une bibliothèque mathématique symbolique utilisée pour les réseaux de neurones et est la mieux adaptée à la programmation de flux de données pour toute une série de tâches. Elle offre de multiples niveaux d'abstraction pour la construction et l'entraînement de modèles. TensorFlow est une entrée prometteuse et en pleine croissance dans le monde de deep learning. Elle offre un écosystème flexible et complet de ressources communautaires, de bibliothèques et d'outils qui facilitent la création et le déploiement d'applications de deep learning.

[1] https://www.tensorflow.org/

1.4.2 Keras

Keras[2] est une interface de programmation d'application (API) de réseau de neurones de haut niveau efficace, écrite en Python. Cette bibliothèque de réseau de neurones à code source ouvert est conçue pour permettre une expérimentation rapide des réseaux de neurones profonds et peut fonctionner en utilisant TensorFlow comme système de gestion.

Keras est une API modulaire, convivial et extensible. Il ne traite pas les calculs de bas niveau, mais les transmet à une autre bibliothèque appelée Backend. Keras a été adopté et intégré dans TensorFlow à la mi-2017. Les utilisateurs peuvent y accéder via le module *tf.keras.* Toutefois, la bibliothèque Keras peut toujours fonctionner séparément et indépendamment.

1.5 Déroulement et structure de la thèse

Dans cette thèse, nous avons essayé de couvrir autant de détails que possible et de garder la plupart des informations nécessaires complètes et rigoureuses. La plupart des détails de haut niveau sont énoncés dans les sept chapitres qui composent cette thèse. La structure globale de ce manuscrit de thèse se présente sous la forme de quatre parties avec sept chapitres organisés comme suit :

Partie I : Introduction (partie actuelle)

Partie II : État de l'art

Dans le **chapitre 2**, nous présentons un aperçu détaillé de la reconnaissance automatique du locuteur, de ses branches, de ses méthodes de pointe. Nous présentons également le processus de reconnaissance automatique du locuteur, de l'extraction des caractéristiques à l'évaluation des locuteurs.

Dans le **chapitre 3**, nous présentons les architectures de deep learning utilisées dans cette thèse, en partant du perceptron jusqu'aux architectures les plus profondes.

Partie III : Contributions

Dans le **chapitre 4**, nous présentons deux corpus de parole pour la reconnaissance automatique du locuteur. Le premier corpus, appelé FSCSR, c'est un corpus de parole gratuit et open source que nous avons rassemblé manuellement. Le second corpus, appelé THUYG, est le corpus que nous avons utilisé dans tous nos

[2]https://keras.io/

travaux. Dans ce chapitre, nous avons également présenté le protocole expérimental suivi dans tous les travaux de la thèse.

Dans le **chapitre 5**, nous proposons une méthode d'évaluation pour la vérification automatique du locuteur, appelée méthode du plus proche cluster (NearC). Cette méthode est basée sur la mesure de la similarité entre les locuteurs en visant les caractéristiques spécifiques des locuteurs.

Dans le **chapitre 6**, nous proposons une approche de deep learning pour la vérification automatique du locuteur. En effet, cette approche est considérée comme une amélioration de la méthode proposée dans le chapitre 5. Dans cette approche, l'objectif est de transformer les vecteurs caractéristiques du locuteur en vecteurs caractéristiques profonds (DeepSF) afin de mieux traiter les données de parole proche des conditions réelles. Dans ce travail, nous avons utilisé deux architectures de deep learning: les réseaux de neurones profonds et les réseaux de croyances profonds.

Dans le **chapitre 7**, nous proposons une nouvelle approche pour intégrer les réseaux de neurones convolutifs dans la reconnaissance automatique du locuteur. Dans ce travail, nous avons utilisé des machines Boltzmann restreintes pour transformer les vecteurs caractéristiques du locuteur en données bidimensionnelles, et le réseau de neurones convolutif comme classifieur pour la vérification automatique du locuteur.

Dans le **chapitre 8**, nous proposons une nouvelle façon de développer un réseau de neurones convolutif pour la reconnaissance automatique du locuteur. Ce travail est une amélioration du travail présenté dans le chapitre 7.

Partie IV : Conclusions et Perspectives

Dans cette partie nous présentons une conclusion générale de la thèse ainsi que les perspectives et les travaux futurs à envisager.

Partie II

État de l'art

2

Reconnaissance Automatique du Locuteur

Sommaire

2.1 Introduction

Dans ce chapitre, nous présentons un aperçu détaillé de la reconnaissance automatique du locuteur, de ses branches et de ses méthodes avancées. Nous présentons également le processus de reconnaissance automatique du locuteur, de l'extraction des caractéristiques à l'évaluation des locuteurs. Le chapitre est organisé comme suit : dans la section 2.2 nous présentons les principes de base de la reconnaissance automatique du locuteur, y compris ses branches et ses méthodes. Dans la section 2.3, nous présentons la méthode de pointe d'extraction de caractéristiques. Dans la section 2.4, nous présentons les techniques de modélisation du locuteur. Dans la section 2.5, nous présentons les approches les plus récentes de la modélisation du locuteur. Dans la section 2.7, nous présentons les méthodes populaires utilisées pour la mesure des performances des systèmes de reconnaissance automatique du locuteur, et dans la dernière section 2.8, nous clôturons le chapitre.

2.2 Principes fondamentaux de la reconnaissance automatique du locuteur

2.2.1 Définition

La *reconnaissance automatique du locuteur* (RAL), parfois appelée *biométrie du locuteur*, implique l'*identification*, la *vérification* (authentification), la *classification* et, par extension, la *segmentation*, le *suivi* et la *détection* des locuteurs. Il s'agit d'un terme générique utilisé pour toute procédure qui implique la reconnaissance de l'identité d'une personne sur la base de sa voix.

En abordant la RAL, de nombreux termes différents ont été inventés, dont certains ont causé beaucoup de confusion. Les recherches se poursuivent depuis longtemps et, naturellement, il existe une certaine confusion entre la reconnaissance automatique de la parole (RAP) et la RAL. Un terme qui a majoré cette confusion est celui de *reconnaissance vocale*.

Le terme *reconnaissance vocale* a été utilisé dans certains milieux pour désigner la RAL. Bien qu'il s'agisse d'un nom conceptuellement correct pour le sujet, il est recommandé de ne pas l'utiliser. Dans le passé, le terme *reconnaissance vocale* [6–8] a été mal assigné à la RAP, et ces termes sont devenus synonymes depuis longtemps. Dans une application de RAP, ce n'est pas la voix de l'individu qui est reconnue, mais le contenu de son discours. Malheureusement, le terme existe et a été mal associé pendant trop longtemps.

Outre ce qui précède, une myriade de terminologies différentes en anglais ont été utilisées pour désigner ce sujet. Il s'agit notamment de, *voice biometrics* [9], *speech biometrics* [10, 11], *biometric speaker identification* [12, 13], *talker identification* [14, 15], *talker clustering* [13], *voice identification* [16], et *voiceprint identification* [17]. À l'exception du terme *speech biometrics* qui introduit également l'ajout d'une base de connaissances de la parole à la tâche de la RAL.

Pour éviter toute confusion supplémentaire, nous proposons une utilisation standard des termes les plus populaires, et les plus concis pour le sujet dans cette discipline. Ces termes sont la reconnaissance automatique du locuteur (RAL) pour l'ensemble de la classe de problèmes, l'*identification automatique du locuteur*, la *vérification automatique du locuteur*, la *classification automatique du locuteur*, la *segmentation automatique du locuteur*, le *suivi automatique du locuteur* et la *détection automatique du locuteur* pour les branches spécifiques de la discipline. Bien sûr, il existe d'autres combinaisons de la discipline de la RAL avec d'autres sources de connaissances telles que la *diarisation du locuteur* et la *biométrie de la parole*.

Un système de RAL tente, d'abord, de modéliser les caractéristiques du tractus vocal d'une personne. Il peut s'agir d'un modèle mathématique du système physiologique qui produit la parole humaine [18, 19], ou simplement d'un modèle statistique dont les caractéristiques de sortie sont similaires à celles du tractus vocal humain [10]. Une fois qu'un modèle est établi et a été associé à un individu, de nouveaux cas de parole peuvent être évalués pour déterminer la probabilité qu'ils aient été générés par le modèle en question, par opposition à d'autres modèles observés. C'est la méthodologie qui sous-tend toutes les applications de la RAL.

2.2.2 Branches de la RAL

La discipline de la RAL comporte de nombreuses branches qui sont directement ou indirectement liées. En général, elle se manifeste de six manières différentes. Nous classifions ces branches en deux groupes différents, *simple* et *composé*. Les branches simples sont celles qui sont autonomes. D'autre part, les branches composées sont celles qui utilisent une ou plusieurs des branches simples, éventuellement avec des techniques supplémentaires. Les branches simples de la RAL sont la vérification, l'identification, et la classification automatique du locuteur. Selon la définition ci-dessus, les branches composées de la RAL sont la segmentation, la détection, et le suivi automatique du locuteur. Actuellement, la vérification automatique du locuteur est la branche la plus populaire en raison de la forte demande d'applications de sécurité et d'accès vocal.

2.2.2.1 Vérification automatique du locuteur

Dans l'application générique de vérification automatique du locuteur (VAL), la personne faisant l'objet de la vérification, appelée *locuteur de test*, proclame une identité, généralement en utilisant des méthodes non-vocales (par exemple, un nom d'utilisateur, un numéro d'identification, etc.). Dans le cas des méthodes non-vocales, il s'agit de méthodes strictement basées sur le contenu ; cette information peut toujours être délivrée par le support vocal, mais le signal du support vocal n'est pas directement utilisé pour l'identification dans les méthodes dites non-vocales. L'identité proclamée est utilisée pour extraire le modèle de cette personne d'une base de données. Ce modèle est appelé le modèle du *locuteur cible*. Ensuite, le signal vocal du locuteur de test (segment de test) est comparé au modèle du locuteur cible. En général, une comparaison avec le modèle du locuteur cible n'est pas suffisante.

Le processus de vérification nécessite deux éléments opposés. En moralité, lorsqu'on veut évaluer le comportement humain, il serait difficile de l'évaluer sans avoir deux sens opposés, qui seraient la bonté et la méchanceté. Nous pouvons obtenir un modèle de bonté que nous pouvons comparer à celui-ci, mais nous ne pourrons pas construire une comparaison significative sans avoir le modèle opposé. Le même principe est vrai pour la VAL, pour faire une comparaison significative d'un locuteur, nous devons calculer la similarité entre le locuteur de test et le modèle du locuteur cible et la similarité entre le locuteur de test et d'autres locuteurs qui représentent "le reste du monde".

Pour mesurer correctement la proximité d'un locuteur de test par rapport à un modèle de locuteur cible, la littérature propose plusieurs approches, parmi lesquelles le *modèle générique des locuteurs (UBM: Universal Background Model)* [6] qui est le plus largement utilisé dans le domaine de la VAL. Ce modèle est généralement basé sur des données collectées auprès d'une large population de locuteurs. Le concept de ce modèle fait référence à la construction de l'opposé du modèle du locuteur cible. Ainsi, nous pouvons comparer le locuteur de test aux deux modèles: le modèle du locuteur cible et le modèle UBM.

2.2.2.2 Identification automatique du locuteur

Dans l'identification automatique du locuteur (IAL) le locuteur de test est comparé à un ensemble de modèles de locuteurs, et le(s) candidat(s) ayant la probabilité de similarité la plus élevée est celui(ceux) qui est(sont) retourné(s).

L'IAL comporte deux types d'identification : l'identification dans un ensemble fermé et l'identification dans un ensemble ouvert. Pour l'identification dans un

ensemble fermé, nous mesurons la similarité entre le segment de test et les modèles de locuteurs connus, où le nombre de locuteurs est limité. Par conséquent, le(s) locuteur(s) le(s) plus proche(s) est(sont) choisi(s). D'autre part, l'identification dans un ensemble ouvert peut être considérée comme une combinaison de la VAL et de l'IAL dans un ensemble fermé. On peut expérimenter une identification dans un ensemble fermé suivie d'une vérification pour rassurer les résultats.

2.2.2.3 Classification automatique du locuteur

En général, la classification est le processus qui consiste à étiqueter des formes en fonction de caractéristiques communes. Il en va de même pour la classification des locuteurs, l'objectif étant de placer des signaux audio similaires dans des groupes spécifiques.

Les sous-branches les plus couramment utilisées dans la classification des locuteurs sont la classification par genre et la classification par âge. La tâche de classification par genre, comme son nom l'indique, tente de regrouper les locuteurs masculins dans une case et les locuteurs féminins dans l'autre. La tâche de classification par âge tente également de séparer les locuteurs dans deux ou plusieurs bacs étiquetés, par exemple, en classant les locuteurs en trois classes : enfants, jeunes et personnes âgées.

La classification des locuteurs peut exploiter différents ensembles de caractéristiques des locuteurs utilisés dans la vérification ou l'identification des locuteurs. Pour la classification par genre, nous nous intéressons davantage aux voyelles et aux fricatives car elles se caractérisent par la fréquence fondamentale de l'appareil vocal et ses harmoniques supérieures, ce qui nous permet de distinguer les hommes, les femmes et les enfants (les fréquences fondamentales du conduit vocal sont de 130 Hz pour les hommes, 220 Hz pour les femmes et 265 Hz pour les enfants) [20]. Alors que dans la classification par âge, nous pouvons utiliser les caractéristiques de Jitter et Shimmer [21].

2.2.2.4 Segmentation automatique du locuteur

La segmentation automatique du locuteur consiste à diviser un flux audio en segments, contenant des locuteurs distincts, en reconnaissant leurs caractéristiques vocales spécifiques. Le concept de segmentation consiste à trouver des moments où nous détectons des changements acoustiques dans le flux audio.

La segmentation du locuteur est considérée comme un type de RAL, étant donné la grande similitude entre le traitement de la segmentation et les techniques de reconnaissance du locuteur. En pratique, une marque de segmentation est faite au

point de transition, entre deux segments dissemblables, dans le flux audio, et il est alors préférable d'étiqueter ces segments avec les locuteurs qui leur sont associés.

Les applications de la segmentation des locuteurs sont différentes, et la plus connue est la segmentation des locuteurs pour les applications de téléconférence. Les participants à une téléconférence se trouvent dans des lieux différents, associés à des appareils de télécommunication. Chaque participant peut se joindre à la conversation en utilisant un code d'accès spécifique, et à la fin de la téléconférence, il est utile d'établir le discours complet de chaque participant. Si tous les locuteurs sont enregistrés, de sorte que la tâche d'identification est appliquée, alors les segments qui contiennent la voix de ce locuteur doivent être segmentés et retirés des locuteurs adjacents.

Parfois, le terme "segmentation du locuteur" est utilisé comme nom alternatif pour la diarisation du locuteur, qui est une combinaison de la segmentation du locuteur et de la classification du locuteur.

2.2.2.5 Détection et le suivi automatique du locuteur

La détection automatique du locuteur peut être considérée comme un mélange de segmentation du locuteur et d'identification et/ou de vérification du locuteur. Il s'agit de l'action de détecter un ou plusieurs locuteurs, dans un flux audio.

Un exemple de la combinaison de la segmentation du locuteur, et d'autres techniques de RAL est celui d'une liste connue de locuteurs, où nous détecterions un locuteur dans le flux audio, puis une IAL dans un ensemble fermé est appliquée aux segments. Un exemple plus complexe est celui où le flux audio contient un locuteur en dehors de la liste, de la musique ou un autre type d'audio, et si le nombre de locuteurs n'est pas grand, alors une tâche de VAL peut être utilisée pour chaque membre de la liste. En outre, si le nombre de locuteurs est grand, une IAL dans un ensemble ouvert peut être effectuée.

D'autre part, le suivi automatique du locuteur est le processus qui consiste à suivre un ou plusieurs locuteurs à travers un flux audio. Il est assez similaire à la détection automatique du locuteur, mais dans ce cas, un ou plusieurs locuteurs sont suivis à travers un flux audio. Chaque segment est étiqueté et attribué au locuteur cible. Dans cette tâche, nous disposons d'une liste de locuteurs cibles connus. Le suivi des locuteurs est couramment utilisé dans les conversations pour marquer chaque locuteur.

2.2.3 Modalités de la RAL

Théoriquement, la RAL peut être mise en œuvre selon différentes modalités. En pratique, ces modalités ne sont pertinentes que pour la VAL. Dans ce contexte, en fonction des exigences de l'application, différentes sources d'information peuvent être mélangées aux informations acoustiques présentes dans le conduit vocal.

2.2.3.1 RAL dépendante du texte

La RAL en mode *dépendant du texte*, telle qu'elle peut apparaître à partir de son nom, incite le locuteur à dire une phrase spécifique au moment du test. Elle a été développée principalement pour lutter contre l'usurpation d'identité par des imposteurs. Si le locuteur n'anticipe pas le texte qui lui sera demandé à prononcer, il ne pourra pas se préparer à tromper le système.

L'approche principale pour concevoir un système RAL dépendant du texte, consiste à générer des phrases aléatoires pour ses clients. La réponse doit donc correspondre aux caractéristiques vocales du locuteur cible ainsi qu'au contexte de la phrase à citer. Notons que la RAL en mode dépendant du texte n'a de sens que pour la VAL.

2.2.3.2 RAL indépendante du texte

La RAL en mode *indépendant du texte* est la modalité viable, qui peut être utilisée dans toutes les branches de la RAL. Un système de RAL purement indépendant du texte et de la langue, ne repose que sur les caractéristiques du tractus vocal du locuteur, et ne fait aucune supposition sur le contexte du discours. L'un des problèmes les plus importants des systèmes de RAL indépendants du texte est la possibilité d'une mauvaise couverture de la partie du discours. Prenons l'exemple d'un segment d'entraînement. Dans un processus indépendant du texte, il n'y a généralement pas de contrainte sur le texte d'entraînement. De plus, un objectif commun à tous les systèmes de reconnaissance est d'essayer de minimiser la longueur des segments d'entraînement et de test.

La réduction de la durée des données d'entraînement et de test réduit la possibilité d'une couverture commune de l'espace phonétique. Par conséquent, certaines parties du segment de test peuvent n'avoir jamais été vues au moment de l'entraînement. Il est donc plausible que l'énoncé d'entraînement d'un imposteur, et l'énoncé de test du locuteur cible, aient plus en commun sur le plan acoustique, que l'énoncé d'entraînement du locuteur cible et le segment de test pour ce locuteur. Ce point commun peut contribuer à une mauvaise reconnaissance. Pour résoudre ce problème, la plupart des moteurs de RAL indépendants du texte nécessitent un temps d'entraînement et de test plus long pour atteindre une précision acceptable.

2.3 Extraction des caractéristiques

Un signal est une mesure observée d'un phénomène physique. Il décrit généralement une observation d'un phénomène physique de niveau supérieur, en corrélation avec des concepts de mesure de niveau inférieur tels que le temps ou l'espace. Dans ce contexte, le signal vocal est une mesure observée, effectuée par rapport au temps qui passe.

Pour simplifier le traitement des signaux continus, l'ensemble infini de valeurs possibles peut être réduit à un ensemble fini grâce à l'*échantillonnage*. Cette action est appelée *discrétisation*, et le signal redéfini est appelé *signal discret*. Ce dernier est capable de transformer l'ensemble fini de points vers une mesure de niveau supérieur.

La parole est un signal variable dans le temps qui transmet plusieurs couches d'informations, notamment des mots, l'identité des locuteurs, les caractéristiques acoustiques, le langage et les émotions. Les informations contenues dans la parole peuvent être observées dans les domaines du temps et des fréquences. L'outil de visualisation le plus largement utilisé est le *spectrogramme*, dans lequel les spectres de fréquence de segments consécutifs de parole à court terme sont affichés sous forme d'image. Dans l'image, les dimensions horizontale et verticale représentent respectivement le temps et la fréquence, et l'intensité de chaque point de l'image indique l'amplitude d'une fréquence particulière à un moment donné, voir la figure 2.1.

Les spectrogrammes sont très utiles pour la visualisation du signal de la parole, mais ils ne sont pas conçus pour la reconnaissance de la parole et du locuteur. Il y a deux raisons à cela. Premièrement, la dimension fréquentielle est encore trop élevée. Pour une transformée de Fourier rapide, *Fast Fourier Transform* (FFT) de 1024 points, la dimension fréquentielle est de 512, ce qui est beaucoup trop important pour la modélisation statistique. Deuxièmement, les composantes de fréquence après la FFT sont fortement corrélées entre elles, ce qui ne facilite pas l'utilisation de matrices de covariance diagonale pour modéliser la variabilité des vecteurs caractéristiques. Pour obtenir une représentation plus compacte des signaux de parole et pour décorréler les composantes caractéristiques, on utilise souvent une représentation *Cepstrale* de la parole.

En général, un signal de la parole varie continuellement pendant toute sa durée en raison des mouvements articulatoires. Cependant, la division en courtes trames qui se chevauchent peut donner lieu à des segments de parole où le signal est considéré comme invariable. À cet égard, les coefficients cepstraux de Mel-Fréquence, *Mel-Frequency Cepstral Coefficients* (MFCC) [22] sont couramment utilisés pour les applications de reconnaissance de la parole et du locuteur.

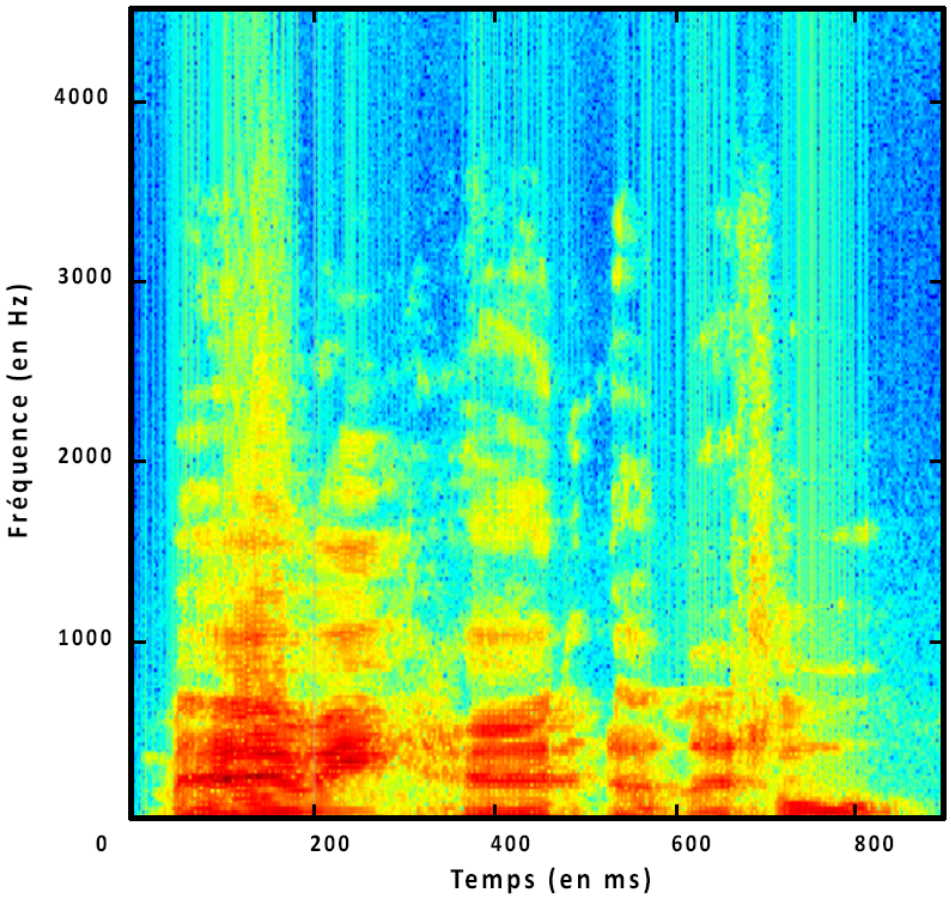


Figure 2.1: Illustration d'un spectrogramme.

La force des MFCC réside dans leur capacité de modéliser la forme de l'appareil vocal dans un spectre de puissance à court terme. La figure 2.2 montre le processus d'extraction des MFCC à partir d'un segment de parole. Dans la figure 2.2, $s(n)$ représente une trame de parole, $X(m)$ est le logarithme du spectre aux fréquences définies par le $m^{ième}$ filtre dans la banque de filtres, et o_i sont ses MFCC comme défini par Eq. 2.1.

$$o_i = \sum_{m=1}^{M} cos\left[i\left(m - \frac{1}{2}\right)\frac{\pi}{2}\right] X(m), i = 1, \ldots, P \tag{2.1}$$

Dans la Figure 2.2, les symboles Δ et $\Delta\Delta$ (Δ^2) représentent respectivement la vitesse et l'accélération des MFCC. En désignant e comme l'énergie logarithmique, et le vecteur acoustique correspondant à $s(n)$ est donné par:

$$o = [e, o_1, \ldots, o_p, \Delta e, \Delta o_1, \ldots, \Delta o_p, \Delta\Delta e, \Delta\Delta o_1, \ldots, \Delta\Delta o_p] \tag{2.2}$$

Dans la plupart des systèmes de RAL, $P = 12$, ce qui donne $o \in \mathbb{R}^{39}$.

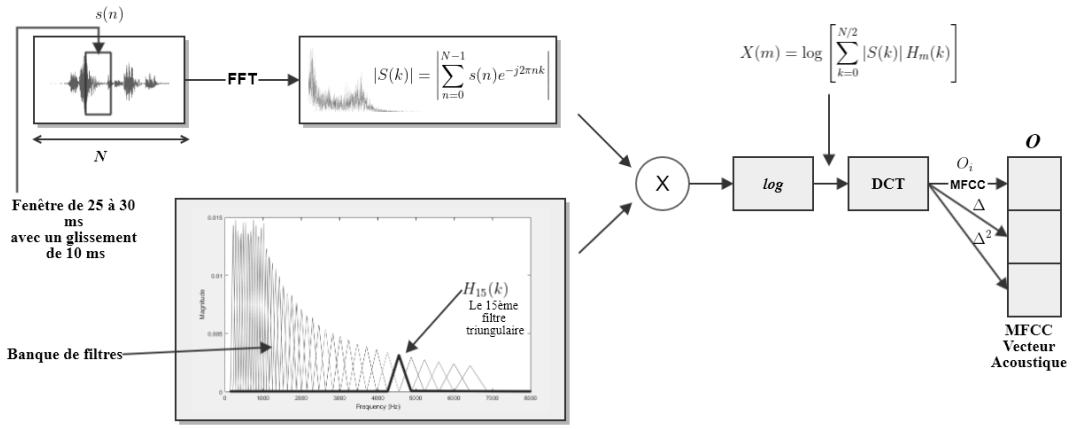


Figure 2.2: Le processus d'extraction d'un vecteur MFCC à partir d'une trame de parole. Voir Eq. 2.1 et Eq. 2.2 pour o_i et o, respectivement.

2.4 Modélisation des locuteurs

En utilisant des vecteurs caractéristiques extraits des paroles d'entraînement d'un locuteur donné, un modèle de locuteur est formé et stocké dans la base de données du système. En RAL dépendant du texte, le modèle est spécifique à l'énoncé, et il inclut les dépendances temporelles entre les vecteurs caractéristiques. La vérification automatique du locuteur en mode dépendant du texte et la reconnaissance de la parole partagent des similitudes dans leurs processus de comparaison des modèles.

En RAL indépendante du texte, nous modélisons souvent la distribution des caractéristiques, c'est-à-dire la forme du *nuage de caractéristiques* plutôt que les dépendances temporelles. Il est important de noter qu'en reconnaissance en mode dépendant du texte, nous pouvons aligner temporellement les énoncés de test et d'entraînement car ils contiennent les mêmes séquences de phonèmes. Cependant, en reconnaissance indépendante du texte, étant donné qu'il y a peu ou pas de correspondance temporelle entre les trames du test et les énoncés de référence, l'alignement au niveau des trames n'est pas possible. Par conséquent, la segmentation du signal en classes phonétiques peut être utilisée comme étape de pré-traitement, ou les modèles de locuteurs peuvent être structurés phonétiquement. De telles approches ont été proposées dans [23–25]. Il est également possible d'utiliser des unités pilotées par les données au lieu des phonèmes strictement linguistiques comme unités de segmentation [26].

Les modèles classiques de locuteurs peuvent être divisés en modèles *Templates* et en modèles *stochastiques* [27]. Dans les modèles Templates, les vecteurs caractéristiques d'entraînement et de test sont directement comparés les uns aux autres en partant du principe que l'un est une réplique imparfaite de l'autre. Le

degré de distorsion entre eux représente leur degré de similarité. La *quantification vectorielle (VQ: Vector Quantization)* [28] et la distorsion temporelle dynamique *distorsion temporelle dynamique (DTW: Dynamic Time Warping)* [29] sont des exemples représentatifs des modèles Templates pour la reconnaissance indépendante et dépendante du texte, respectivement.

Dans les modèles *stochastiques*, chaque locuteur est modélisé comme une source probabiliste avec une fonction de densité de probabilité inconnue mais fixe. La phase d'entraînement consiste à estimer les paramètres de la fonction de densité de probabilité à partir d'un échantillon d'entraînement. La comparaison entre les modèles se fait généralement en évaluant la probabilité de l'énoncé du test concernant le modèle. Le *modèle de mélange gaussien (GMM: Gaussian Mixture Model)* [30, 31] est le modèle le plus populaires pour la reconnaissance indépendante et du texte.

En 2000 [31], Reynolds a proposé d'utiliser la parole d'un grand nombre de locuteurs pour entraîner un modèle GMM, afin de modéliser les caractéristiques acoustiques d'une population générale. Le modèle qui en résulte est le modèle UBM. La densité des vecteurs acoustiques est donnée par :

$$p(\mathbf{o}|\text{UBM}) = p(\mathbf{o}|\Lambda^{ubm}) = \sum_{c=1}^{C} \pi_c^{ubm} \mathcal{N}(\mathbf{o}|\mu_c^{ubm}, \sum\nolimits_c^{ubm}), \tag{2.3}$$

les paramètres de l'UBM, $\Lambda^{ubm} = \{\pi_c^{ubm}, \mu_c^{ubm}, \sum_c^{ubm}\}_{c=1}^{C}$, sont estimés par l'algorithme *expectation-maximization* (EM) [32] utilisant la parole de nombreux locuteurs.

Tandis que l'UBM représente la population générale, les locuteurs individuels sont modélisés par des modèles GMM dépendant du locuteur. Plus précisément, pour un locuteur cible, son modèle GMM est donné par :

$$p(\mathbf{o}|Locuteur^{(cible)}) = p(\mathbf{o}|\Lambda^{(cible)}) = \sum_{c=1}^{C} \pi_c^{(cible)} \mathcal{N}(\mathbf{o}|\mu_c^{(cible)}, \sum\nolimits_c^{(cible)}), \tag{2.4}$$

où $\Lambda^{(cible)} = \{\pi_c^{(cible)}, \mu_c^{(cible)}, \sum_c^{(cible)}\}_{c=1}^{C}$ sont appris en utilisant une adaptation maximale a posteriori, *maximum a posteriori adaptation* (MAP) [33].

Étant donné les vecteurs acoustiques $O^{(test)}$ d'un segment de test et d'un locuteur cible, la vérification du locuteur revient à calculer le rapport logarithmique de vraisemblance :

$$S_{\text{GMM/UBM}}(O^{(test)}|\Lambda^{ubm}) = \log(\frac{p(O^{(test)}|\Lambda^{(cible)})}{p(O^{(test)}|\Lambda^{(ubm)})}), \tag{2.5}$$

où $\log p(O^{(test)}|\Lambda^{(cible)})$ est le logarithme de vraisemblance de $O^{(test)}$ étant donné le modèle du locuteur cible $\Lambda^{(cible)}$. La figure 2.3 représente le processus d'évaluation de tâche de vérification automatique du locuteur en utilisant le modèle GMM/UBM.

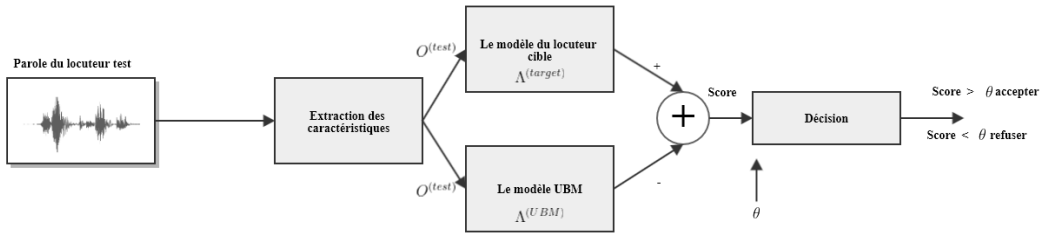


Figure 2.3: La procédure d'évaluation du modèle GMM/UBM pour la tâche de vérification automatique du locuteur.

Selon le paradigme de l'entraînement, les modèles peuvent également être classés en modèles génératifs et discriminatifs. Les modèles génératifs tels que GMM et VQ estiment la distribution des caractéristiques au sein de chaque locuteur. En revanche les modèles discriminatifs, tels que les *réseaux de neurones artificiels (ANN: Artificial Neural Networks)* [34, 35] et les *machines à vecteurs de support (SVM: Support Vector Machines)* [36] modélisent la frontière entre locuteurs.

En résumé, un locuteur est caractérisé par un modèle de locuteur tel que VQ, GMM ou SVM. Au moment de l'exécution, un segment de test est d'abord représentée par une collection de vecteurs caractéristiques ou un supervecteur (une concaténation de plusieurs vecteurs), puis évaluée par rapport aux modèles de locuteurs cibles.

2.5 Approches modernes de la RAL

Le GMM/UBM est une approche basée sur les trames dans la mesure où les modèles de locuteurs (GMM) décrivent la distribution des trames acoustiques. Depuis son introduction en fin des années 90, elle a été la méthode de pointe pour la vérification automatique du locuteur pendant plusieurs d'années. Cependant, elle a ses propres limites. Un inconvénient majeur est que l'entraînement des GMM et des UBM est disjointe, ce qui signifie que les informations contrastées entre les locuteurs cibles et les imposteurs ne peuvent pas être explicitement intégrées dans le processus d'entraînement. Un autre inconvénient est qu'il est difficile de supprimer les informations non-locuteurs (par exemple, le canal et le bruit). Bien que des tentatives aient été faites pour supprimer les variations de canaux et de bruit dans les domaines des caractéristiques, elles ne sont pas aussi efficaces que les approches modernes.

En 2006, les chercheurs ont commencé à examiner le problème de la vérification automatique du locuteur d'une autre manière. Au lieu d'accumuler les scores du logarithme de vraisemblance d'un énoncé basés sur une trame, les chercheurs ont

élaboré des méthodes pour relier les caractéristiques acoustiques de l'ensemble de l'énoncé à un vecteur à haute dimension. Ces vecteurs basés sur l'énoncé vivent dans un espace à haute dimension paramétré par les GMM. Comme la dimension est la même quelle que soit la durée de l'énoncé, les méthodes *Machine Learning* standard telles que les SVM et l'analyse factorielle, *Factor Analysis* (FA) peuvent être appliquées sur cet espace. Les trois méthodes influentes, basées sur cette idée, sont les GMM/SVM, l'analyse factorielle conjointe, *Joint Factor Analysis* (JFA) et les *i-vectors*.

Dans le modèle GMM/SVM [37], les *supervecteurs* sont construits à partir de GMM des locuteurs cibles construits par adaptation du modèle UBM. Pour chaque locuteur cible, un SVM, dépendant du locuteur, est ensuite entraîné à distinguer ses supervecteurs de ceux des imposteurs. Afin de réduire les décalages entre les canaux, les directions correspondant à la variabilité des locuteurs sont projetées vers l'extérieur. L'évaluation revient à calculer les scores SVM des énoncés du test, et les décisions sont prises en calculant les scores avec un seuil de décision.

Dans le modèle JFA [38], les variabilités des locuteurs et des sessions sont représentées par des variables latentes, (*facteurs de locuteurs* et *facteurs de canaux*), dans un modèle d'analyse factorielle. Lors de l'évaluation, les variabilités des sessions sont prises en compte par l'intégration des variables latentes, par exemple les facteurs de canaux.

2.6 L'approche *i*-vector/PLDA

Dans un système *i*-vector [39], les énoncés sont représentés par la moyenne postérieure des facteurs latents, appelés *i*-vectors. Les *i*-vectors capturent à la fois les informations du locuteur et du canal. Lors de l'évaluation, la variabilité indésirable du canal est éliminée par une analyse discriminante linéaire, *Linear Discriminant Analysis* (LDA), ou par l'intégration des facteurs latents dans un modèle LDA probabiliste [40].

2.6.1 *i*-vector

Étant donné un segment de parole, le modèle *i*-vector suppose que le *supervecteur* M dépendant du locuteur est généré par :

$$M = m + Tw, \tag{2.6}$$

où m est un supervecteur indépendant du locuteur et du canal, T est une matrice de rang inférieur, et w est un vecteur de faible dimension qui représente le segment

de parole. En supposant que w suit une distribution normale standard $N(0, I)$, Eq. 2.6 est la représentation d'un modèle linéaire gaussien et M suit une distribution gaussienne $N(m, TT^T)$. L'estimation des paramètres et l'inférence des variables avec ce modèle peuvent être facilement réalisées. Plus précisément, étant donné un ensemble de signaux de parole d'entraînement $\{X_i\}$, la matrice T est estimée en optimisant la fonction de vraisemblance suivante :

$$\mathcal{L}(T) = \sum_i ln\{P(X_i; T)\} = \sum_i ln\{\sum_M P(X_i; M)P(M; T)\}, \tag{2.7}$$

où la probabilité conditionnelle $P(X_i; M)$ est modélisée par une GMM, et la probabilité antérieure $P(M; T)$ est une gaussienne. Une fois que T est estimé, déduire la probabilité postérieure de w pour un segment de parole X est simple puisque $P(w|X)$ est également une gaussienne. Dans la plupart des cas, seul le vecteur moyen (appelé "i-vector") de la partie postérieure est concerné et il peut être obtenu par l'algorithme MAP. Pour plus de détails sur le calcul, voir [41].

Avec des segments de parole représentés par des i-vectors, le score d'un segment de test sur un locuteur déclaré (locuteur cible) peut être calculé comme la distance en cosinus entre les i-vectors du locuteur de test et la parole d'entraînement du locuteur déclaré.

2.6.2 Analyse discriminante linéaire probabiliste (PLDA)

Le modèle i-vector est un modèle à variabilité totale, ce qui signifie qu'un i-vector représente à la fois les caractéristiques du locuteur et d'autres facteurs non liés au locuteur, en particulier les canaux. C'est certainement peu idéal pour discriminer les locuteurs. L'analyse discriminante linéaire probabiliste, *probabilistic linear discriminant analysis* (PLDA) sépare l'espace de variabilité totale en un sous-espace de locuteurs et un sous-espace de canaux, afin que les locuteurs puissent être représentés avec plus de précision. Ce modèle peut être formulé par :

$$w_r = m + Ux_r + Vy + \epsilon_r, \tag{2.8}$$

où w_r est l'i-vector du $r^{ème}$ segment de parole, et m est la moyenne de la population. U représente le sous-espace du canal et x_r est un vecteur du canal; V représente le sous-espace du locuteur et y est un vecteur du locuteur, et ϵ_r représente le résidu. Notons que x_r et y suivent la distribution gaussienne standard, et que ϵ_r suit une distribution gaussienne $N(0, \sum)$. Les paramètres $\{m, U, V, \sum\}$ peuvent être estimés à l'aide de l'algorithme EM. L'évaluation d'un segment de test peut être effectuée avec les vecteurs du locuteur en utilisant la distance cosinus.

2.7 Mesures de performance

La vérification automatique du locuteur comporte un ensemble riche de mesures de performance. Bien que les différents ensembles de données aient des mesures légèrement différentes, leurs principes restent les mêmes. Les mesures communes comprennent le taux de faux rejets, *False Rejection Rate* (FRR), le taux de fausses acceptations, *False Acceptance Rate* (FAR), le taux d'erreur égal, *Equal Error Rate* (EER), la fonction de coût minimum, *Minimum Cost Function* (minDCF) et la fonction de coût réel de la décision, *actual Decision Cost Function* (DCF).

La plupart des travaux de vérification automatique du locuteur utilisent l'EER pour l'évaluation des performances.

2.7.1 Taux d'erreur égal (EER)

Avant de définir le taux d'erreur égal, nous devrions d'abord définir le taux de faux rejets et le taux de fausses acceptations. En effet, le taux de fausses acceptations est le pourcentage de cas d'identification dans lesquels des personnes non autorisées sont acceptées à tort. Alors que taux de faux rejets est le pourcentage de cas d'identification dans lesquels des personnes autorisées sont rejetées à tort.

Dans les applications biométriques, il y a un compromis entre le FFR et le FAR tolérés du système. Dans la plupart des applications, il existe un seuil qui peut être choisi pour modifier ce compromis dans un sens ou dans l'autre. La plupart des fournisseurs de moteurs biométriques essaient de résoudre le problème biométrique général et ne sont pas nécessairement conçus pour une application spécifique avec des exigences prédéfinies. Pour cette raison, ils devront rendre compte de leurs résultats de performance dans certaines conditions de fonctionnement. Afin de pouvoir citer des pourcentages pour le taux d'erreur, un point de fonctionnement populaire est le EER, c'est-à-dire le point où il y a une chance égale de faux rejets et de fausses acceptations. En d'autres termes, plus le nombre de fausses acceptations diminue, plus le nombre de faux rejets augmente, et inversement. Le point d'intersection des lignes est le point où l'EER est enregistré. La figure 2.4 représente le point d'intersection (EER) des courbes FFR et FAR. Notons que dans le reste de chapitre, le *taux d'erreur* désigne le taux d'erreur égal.

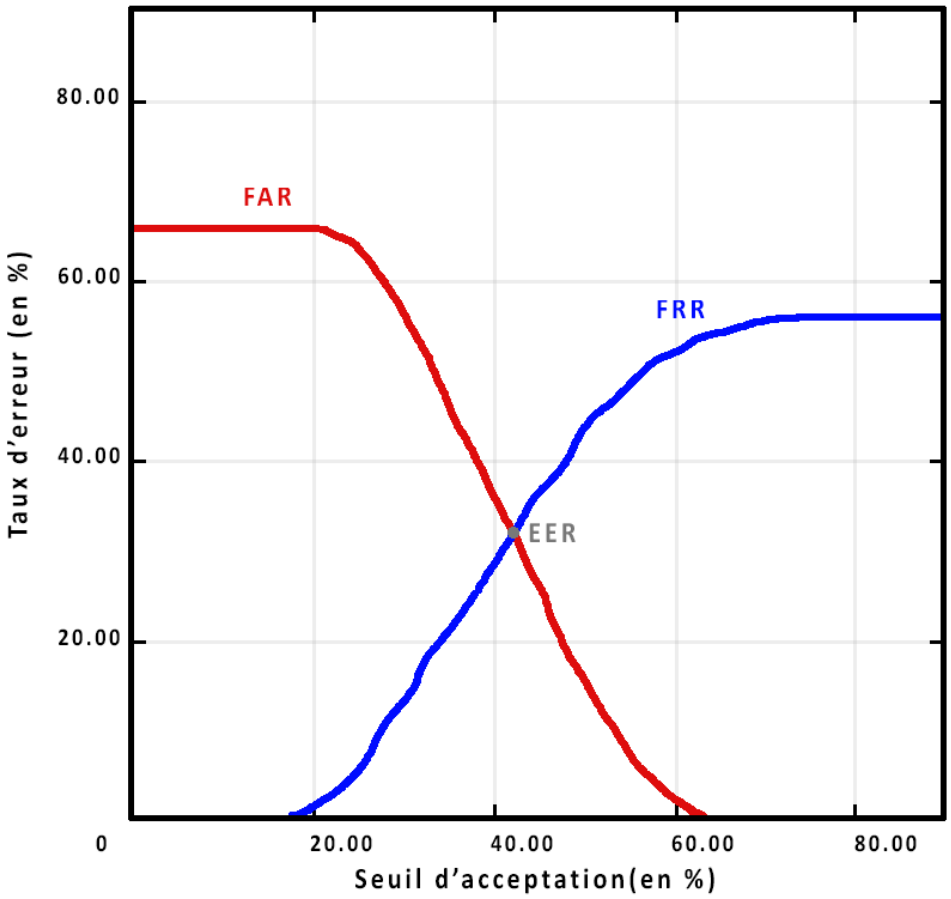


Figure 2.4: Illustration du point d'intersection des courbes FRR et FAR.

2.7.2 Courbe du compromis de l'erreur de détection (DET)

La courbe du compromis de l'erreur de détection, *Detection Error Trade-off* (DET) a été préférée à d'autres courbes, telles que les courbes traditionnelles de caractéristiques de fonctionnement du récepteur, *Receiver Operating Characteristic* (ROC) depuis son introduction par l'*institut national des normes et de la technologie* (NIST) en 1997. La courbe DET trace la *Probabilité manquante* sous forme de pourcentage, en fonction de la *Probabilité de fausse alarme*, également en pourcentage. La figure 2.5 montre un exemple de courbe DET comparant quatre méthodes de vérification différentes. La courbe DET utilise toujours une échelle logarithmique et trace deux entités différentes l'une par rapport à l'autre. Dans ce cas, l'abscisse de la probabilité de fausse alarme qui se trouve être le même nombre que le faux rejet. L'ordonnée est la probabilité d'erreur qui est la même que celle de la fausse acceptation. L'échelle logarithmique rend généralement cette

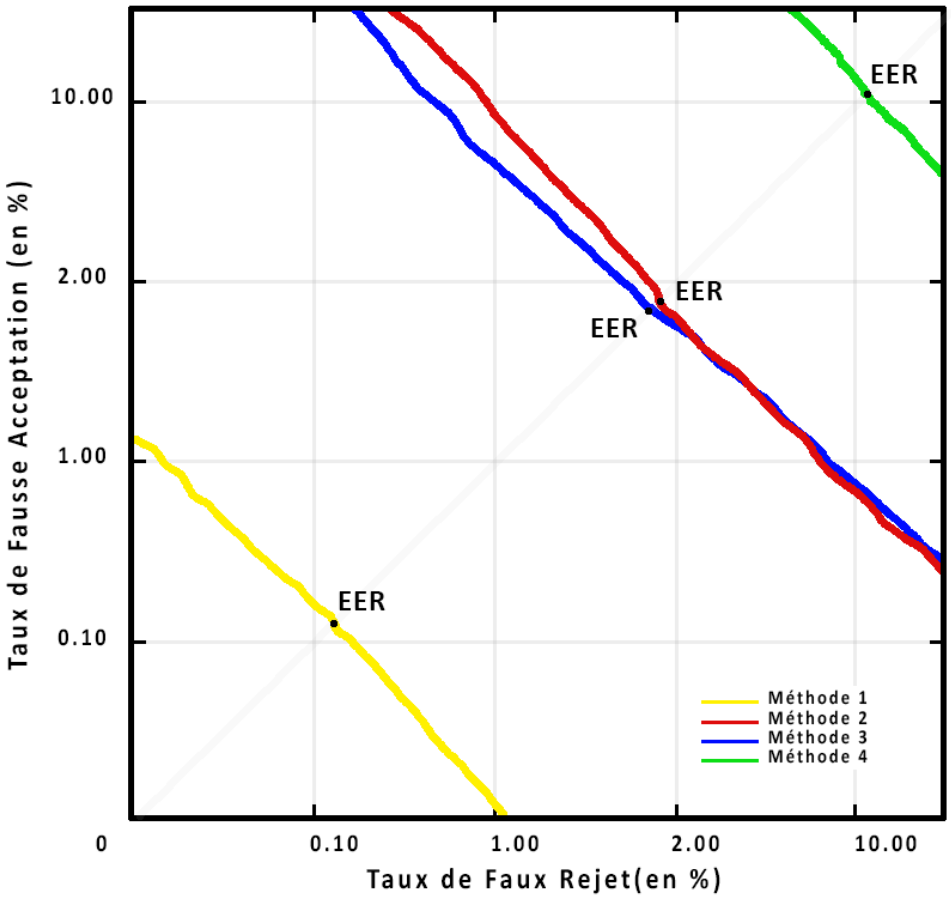


Figure 2.5: Le compromis d'erreur de détection d'échantillon (DET) représentant la performance de quatre techniques de vérification automatique du locuteur.

courbe plus linéaire et le ROC plus concave. Le point de fonctionnement pour l'EER est marqué dans les figures 2.4 et 2.5.

Il convient de noter que le simple fait de connaître l'EER ne permet pas vraiment de se faire une idée des autres points de fonctionnement possibles. Il n'est presque jamais souhaitable d'opérer au point EER. En fonction de l'importance que nous accordons à la sécurité du système protégé par le moteur de vérification par rapport à la facilité d'utilisation, nous pourrions utiliser différents points de fonctionnement. Si nous ne voulons absolument pas que quelqu'un entre dans le système sans autorisation appropriée, alors nous fonctionnerons à un point où nous devrons peut-être déranger les clients en rejetant à tort certains utilisateurs valides afin de maintenir un niveau de sécurité élevé. Ce problème est généralement atténué en demandant aux utilisateurs de fournir davantage de données audio ou en utilisant des informations supplémentaires pour prendre notre décision.

2.8 Conclusion

Dans ce chapitre, nous avons introduit la reconnaissance automatique du locuteur et de ses branches. Nous avons également présenté les approches avancées de la reconnaissance automatiques du locuteur. Dans la suite de cette thèse, nous nous concentrons sur la vérification automatique du locuteur. De plus, nous comparons les méthodes que nous proposons à deux méthodes de pointe : GMM/UBM et *i*-vector/PLDA. Dans le chapitre suivant, nous proposons un aperçu détaillé du deep learning.

3

Apprentissage Profond — Deep Learning

Sommaire

3.1 Introduction

Dans ce chapitre, nous présentons un aperçu détaillé des architectures de deep learning. Dans la section 3.2, nous proposons un aperçu de l'intelligence artificielle, du machine learning et du deep learning, en soulignant la différence entre eux. Dans la section 3.3, nous présentons une introduction aux modèles de deep learning. Dans la section 3.4, nous présentons le perceptron, son utilisation et ses limites. Dans la section 3.5, nous présentons le perceptron multicouche. Dans les sections 3.6, 3.7

et 3.8, nous présentons respectivement les architectures de deep learning suivantes : les machines Boltzmann restreintes, les réseaux de croyances profonds et les réseaux de neurones convolutifs. Dans la section 3.9, nous concluons le chapitre.

3.2 Intelligence artificielle, Machine learning, et Deep learning!

En 2020, les gens bénéficient chaque jour de l'intelligence artificielle : les systèmes de recommandation musicale, Google maps, Uber, et bien d'autres applications sont propulsés par l'intelligence artificielle. Cependant, il y a une confusion entre les termes "intelligence artificielle", "machine learning" et "deep learning". Voici l'une des requêtes les plus populaires de Google "L'intelligence artificielle et le machine learning sont-ils la même chose ?".

Mettons les choses au clair : l'intelligence artificielle, le machine learning et le deep learning sont trois choses différentes:

- L'intelligence artificielle est une science comme les mathématiques ou la biologie. Elle étudie les moyens de construire des programmes et des machines intelligents qui peuvent résoudre des problèmes de manière créative, ce qui a toujours été considéré comme une prérogative humaine ;
- Le machine learning est un domaine de l'intelligence artificielle qui fournit aux systèmes la capacité d'apprendre et de s'améliorer automatiquement à partir de l'expérience sans être explicitement programmé ;
- Le deep learning est un sous-domaine de machine learning, qui utilise les réseaux de neurones pour analyser différents facteurs avec une structure similaire au système neural humain.

De nombreux succès initiaux de l'intelligence artificielle ont eu lieu dans des environnements relativement stériles et formels, et n'ont pas nécessité que les ordinateurs aient beaucoup de connaissances sur le monde. Par exemple, le système de jeu d'échecs *Deep Blue* d'IBM a battu le champion du monde *Garry Kasparov* en 1997 [42]. Les échecs sont bien sûr un monde très simple, ne contenant que soixante-quatre emplacements et trente-deux pièces qui ne peuvent se déplacer que de manière strictement limitée. Concevoir une stratégie d'échecs réussie est une réalisation formidable, mais le défi n'est pas dû à la difficulté de décrire les concepts pertinents à l'ordinateur. Les échecs peuvent être entièrement décrits par une très brève liste de règles complètement formelles, facilement fournies à l'avance par le programmeur.

Ironiquement, les tâches abstraites et formelles qui font partie des tâches mentales les plus difficiles pour un être humain sont parmi les plus faciles pour un ordinateur. Les ordinateurs sont depuis longtemps capables de battre même le meilleur joueur d'échecs humain, mais ce n'est que récemment qu'ils ont atteint un niveau de reconnaissance des objets ou de la parole comparable à celui de l'être humain. La vie quotidienne d'une personne exige une immense connaissance du monde. Une grande partie de ces connaissances sont subjectives et intuitives et donc difficiles à exprimer formellement. Les ordinateurs doivent capturer ces mêmes connaissances pour se comporter de manière intelligente. L'un des principaux défis de l'intelligence artificielle est de savoir comment faire entrer ces connaissances informelles dans un ordinateur.

Plusieurs projets d'intelligence artificielle ont cherché à coder en dur les connaissances sur le monde en langages formels. Un ordinateur peut raisonner automatiquement sur des énoncés dans ces langages en utilisant des règles d'inférence logique. C'est ce qu'on appelle l'approche de l'intelligence artificielle basée sur la connaissance. Aucun de ces projets n'a connu de succès majeur. L'un des projets les plus célèbres est *Cyc*[43]. Cyc est un moteur d'inférence et une base de données d'énoncés dans un langage appelé *CycL*. Ces déclarations sont saisies par une équipe de superviseurs humains. C'est un processus lourd. Les gens s'efforcent de concevoir des règles formelles suffisamment complexes pour décrire le monde avec précision.

Les difficultés rencontrées par les systèmes reposant sur des connaissances codées en dur suggèrent que les systèmes d'intelligence artificielle doivent être capables d'acquérir leurs connaissances, en extrayant des modèles à partir de données brutes. Cette capacité est connue sous le nom de *Machine learning* (apprentissage machine). L'introduction du machine learning a permis aux ordinateurs de s'attaquer à des problèmes impliquant la connaissance du monde réel et de prendre des décisions qui semblent subjectives. Un simple algorithme de machine learning, appelé *logistic regression* (régression logistique), peut déterminer s'il faut recommander une césarienne [44]. Un autre simple algorithme de machine learning appelé *naive Bayes* (Bayes naïf) peut séparer le courrier électronique légitime du courrier électronique non sollicité.

La performance de ces algorithmes simples du machine learning dépend fortement de la représentation des données qui leur sont données. Par exemple, lorsque la régression logistique est utilisée pour recommander une césarienne, le système d'intelligence artificielle n'examine pas directement la patiente. Au lieu de cela, le médecin communique au système plusieurs informations pertinentes, telles que la présence ou l'absence d'une cicatrice utérine. Chaque élément d'information

inclus dans la représentation de la patiente est connu comme une caractéristique. La régression logistique apprend comment chacune de ces caractéristiques de la patiente est en corrélation avec divers résultats. Cependant, elle ne peut en aucun cas influencer la façon dont les caractéristiques sont définies. Si la régression logistique recevait une image IRM[1] en 3D du patient, plutôt que le rapport formalisé du médecin, elle ne pourrait pas faire de prédictions utiles. Les voxels[2] individuels d'une IRM ont une corrélation négligeable avec les complications qui pourraient survenir pendant l'accouchement.

De nombreuses tâches d'intelligence artificielle peuvent être résolues en concevant le bon ensemble de caractéristiques à extraire pour cette tâche, puis en fournissant ces caractéristiques à un simple algorithme de machine learning. Par exemple, une caractéristique utile pour la classification automatique du locuteur à partir du son est la hauteur du son. La hauteur du son peut être formellement spécifiée : il s'agit de la crête majeure de la plus basse fréquence du spectrogramme. Elle est utile pour la classification automatique du locuteur car elle est déterminée par la taille de l'appareil vocal, et donne donc un bon indice pour savoir si le locuteur est un homme, une femme ou un enfant.

Cependant, pour de nombreuses tâches, il est difficile de savoir quelles caractéristiques doivent être extraites. Par exemple, supposons que nous souhaitions écrire un programme pour détecter des voitures sur des photographies. Nous savons que les voitures ont des roues, nous pourrions donc utiliser la présence d'une roue comme caractéristique. Malheureusement, il est difficile de décrire exactement à quoi ressemble une roue en termes de valeurs de pixels. Une roue a une forme géométrique simple mais son image peut être compliquée par les ombres qui tombent sur la roue, le soleil qui éblouit les parties métalliques de la roue, l'aile de la voiture ou un objet au premier plan qui obscurcit une partie de la roue, etc. Le problème est d'autant plus complexe en RAL, du fait que nous ne connaissons même pas les caractéristiques qui distinguent les locuteurs les uns des autres.

Une solution à ce problème est d'utiliser le machine learning pour découvrir non seulement la correspondance entre la représentation et la sortie, mais aussi la représentation en elle-même. Cette approche est connue sous le nom *Representation learning* (apprentissage de la représentation). Les représentations apprises donnent souvent de bien meilleures performances que celles obtenues avec des représentations conçues à la main. Elles permettent également aux systèmes intelligents de s'adapter

[1]IRM: Imagerie par Résonance Magnétique est une technique d'imagerie médicale permettant d'obtenir des vues en deux ou en trois dimensions de l'intérieur du corps.

[2]Une voxel est la valeur en un seul point d'un balayage 3D.

rapidement à de nouvelles tâches, avec une intervention humaine minimale. Un algorithme d'apprentissage des représentations peut découvrir un bon ensemble de caractéristiques pour une tâche simple en quelques minutes, tandis que pour une tâche complexe ça peut prendre quelques heures voir quelques mois en fonction de la complexité de la tâche. La conception manuelle de caractéristiques pour une tâche complexe exige beaucoup de temps et d'efforts humains; elle peut prendre des décennies pour une communauté entière de chercheurs.

L'exemple par excellence d'un algorithme d'apprentissage de la représentation est l'*Auto-Encoder*. Un auto-codeur est la combinaison d'une fonction de codage qui convertit les données d'entrée en une représentation différente, et d'une fonction de décodage qui reconvertit la nouvelle représentation dans le format d'origine. Les auto-codeurs sont formés pour préserver autant d'informations que possible lorsque l'entrée est passée par le codeur puis le décodeur, mais ils sont également formés pour que la nouvelle représentation ait diverses propriétés intéressantes. Les différents types d'auto-codeurs visent à obtenir différents types de propriétés.

Lorsque nous concevons des caractéristiques ou des algorithmes d'apprentissage de caractéristiques, notre objectif est généralement de séparer les facteurs de variation qui expliquent les données observées. Dans ce contexte, nous utilisons le mot "facteurs" simplement pour faire référence à des sources d'influence distinctes; les facteurs ne sont généralement pas combinés par multiplication. Souvent, ces facteurs ne sont pas des quantités directement observées, mais ils peuvent exister sous forme d'objets non observés ou de forces dans le monde physique qui affectent les quantités observables, ou encore ils sont des constructions dans l'esprit humain qui fournissent des explications simplificatrices utiles, ou des causes inférées des données observées. Ils peuvent être considérés comme des concepts ou des abstractions qui nous aident à comprendre la grande variabilité des données. Lors de l'analyse d'un enregistrement de parole, les facteurs de variation comprennent l'âge du locuteur, son genre, son accent et les mots qu'il prononce. Lors de l'analyse d'une image d'une voiture, les facteurs de variation comprennent la position de la voiture, sa couleur, ainsi que l'angle et la luminosité du soleil.

Une source majeure de difficulté dans de nombreuses applications d'intelligence artificielle du monde réel est que de nombreux facteurs de variation influencent chaque élément de données que nous pouvons observer. Les pixels individuels d'une image d'une voiture rouge peuvent être très proches du noir la nuit. La forme de la silhouette de la voiture dépend de l'angle de vision. La plupart des applications exigent que nous démêlions les facteurs de variation et que nous éliminions ceux qui ne nous intéressent pas.

Bien sûr, il peut être très difficile d'extraire des données brutes des caractéristiques abstraites de si haut niveau. Nombre de ces facteurs de variation, tels que l'accent du locuteur, ne peuvent être identifiés qu'en utilisant une compréhension sophistiquée des données proche du niveau humain. Lorsqu'il est presque aussi difficile d'obtenir une représentation que de résoudre le problème original, l'apprentissage de la représentation ne semble pas, à première vue, nous aider.

Le deep learning résout ce problème central de l'apprentissage de la représentation en introduisant des représentations qui sont exprimées en termes d'autres représentations plus simples. Le deep learning permet à l'ordinateur de construire des concepts complexes à partir de concepts plus simples.

L'exemple par excellence d'un modèle de deep learning est le *réseau de neurones profond (DNN: Deep Neural Network)* ou le *perceptron multicouche (MLP: Multi-Layer Perceptron)*. Un MLP n'est qu'une fonction mathématique mettant en correspondance un ensemble de valeurs d'entrée avec des valeurs de sortie. La fonction est formée par la composition de nombreuses fonctions plus simples. On peut considérer que chaque application d'une fonction mathématique différente fournit une nouvelle représentation de l'entrée.

Pour résumer, le deep learning, sujet de ce chapitre, est une approche de machine learning qui permet aux systèmes informatiques de s'améliorer avec l'expérience et les données. Le deep learning est la seule approche viable pour construire des systèmes intelligents qui peuvent fonctionner dans des environnements complexes du monde réel. Le deep learning est un type particulier du machine learning qui atteint une grande puissance et une grande flexibilité en apprenant à représenter le monde comme une hiérarchie imbriquée de concepts et de représentations, chaque concept étant défini par rapport à des concepts plus simples, et les représentations plus abstraites étant calculées en fonction de concepts moins abstraits. La figure 3.1 illustre la relation entre ces différentes disciplines de l'intelligence artificielle.

3.3 Introduction au deep learning

Le concept de deep learning est né de l'étude sur les *réseaux de neurones artificiels (ANN: Artificial Neural Networks)* [45]. Les ANN sont devenues un domaine de recherche actif au cours des dernières décennies [46–50]. Pour construire un réseau de neurones "standard", il est essentiel d'utiliser les neurones pour produire des activations à valeur réelle et, en ajustant les poids, les réseaux de neurones se comportent comme prévu. Cependant, selon les types des problèmes, le processus

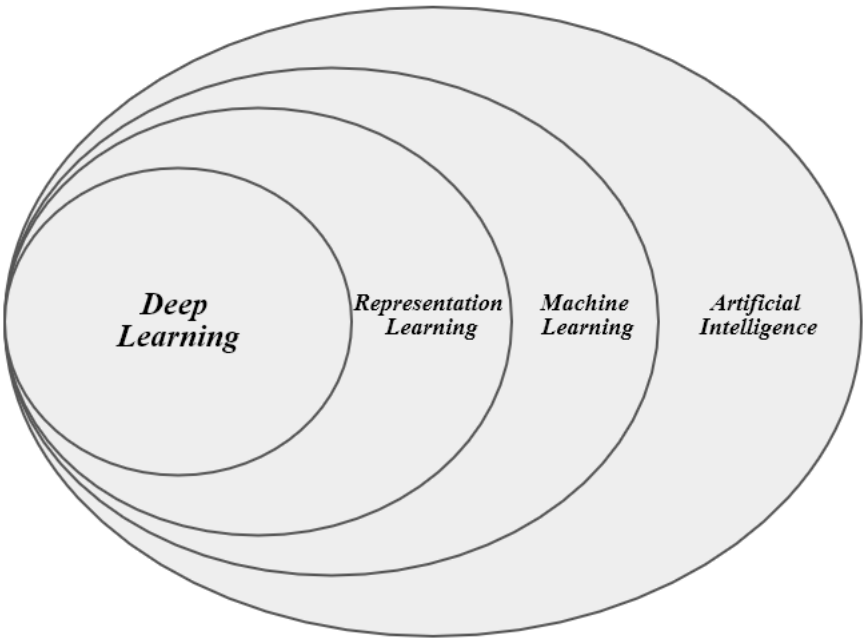


Figure 3.1: Un diagramme de Venn montrant à quel point le deep learning est une sorte d'apprentissage par représentation, qui est à son tour une sorte de machine learning, utilisé pour de nombreuses approches de l'intelligence artificielle, mais pas toutes.

de construction d'un réseau de neurones peut prendre de longues chaînes causales d'étapes de calcul.

La *rétro-propagation* est un algorithme efficace de descente de gradient qui a joué un rôle important dans les réseaux de neurones depuis 1980. Il permet d'entraîner les ANN avec une approche d'apprentissage supervisé. Bien que la précision d'apprentissage soit élevée, la performance de la rétro-propagation appliquée aux données de test pourrait ne pas être satisfaisante. Comme la rétro-propagation est basée sur des informations de gradient local avec un point initial aléatoire, l'algorithme se retrouve souvent piégé dans l'optima local. De plus, si la taille des données d'apprentissage n'est pas assez grande, les réseaux de neurones seront confrontés au problème de sur-apprentissage.

En 2006, Hinton [51] a proposé une nouvelle méthode d'apprentissage (appelée *layer-wise-greedy-learning*) qui a marqué la naissance des techniques de deep learning.

L'idée de base de cet algorithme repose sur un apprentissage non supervisé, qui doit être effectué comme une étape de *pré-entraînement* en réseau avant l'apprentissage ultérieur couche par couche. En extrayant des caractéristiques des entrées, la dimension des données est réduite et on obtient ainsi une représentation compacte. Ensuite, en exportant les caractéristiques vers la couche suivante, tous les échantillons seront étiquetés et le réseau sera affiné avec les données étiquetées.

La raison de la popularité du deep learning est double : d'une part, le développement de grandes techniques d'analyse des données indique que le problème du sur-apprentissage des données d'apprentissage peut être partiellement résolu ; d'autre part, la procédure de pré-entraînement avant l'apprentissage non supervisé va attribuer des valeurs initiales non aléatoires au réseau. Par conséquent, un meilleur minimum local peut être atteint après le processus d'entraînement et un taux de convergence plus rapide peut être obtenu.

Jusqu'à présent, les recherches sur les techniques de deep learning ont suscité beaucoup d'attention et une série de résultats passionnants ont été rapportés dans la littérature. Depuis 2009, la compétition d'ImageNet a attiré un grand nombre de groupes de recherche sur le domaine de traitement d'images dans le monde entier, tant dans le milieu universitaire que dans l'industrie. En 2012, le groupe de recherche dirigé par Hinton a remporté la compétition d'ImageNet en utilisant des approches de deep learning [52]. Le groupe de Hinton a participé à la compétition pour la première fois et ses résultats ont été supérieurs de 10% à ceux de la deuxième place. Google et Baidu ont tous deux mis à jour leurs moteurs de recherche d'images basés sur l'architecture de deep learning de Hinton avec de grandes améliorations dans la précision de la recherche. Baidu a également créé l'*Institute of Deep Learning* (IDL) en 2013 et a invité *Andrew Ng*, professeur associé à l'université de Stanford, en tant que scientifique en chef. En mars 2016, un match de *Go Game* a été organisé en Corée du Sud par le projet de deep learning de Google (appelé *DeepMind*) entre le joueur d'IA *AlphaGo* et l'un des joueurs les plus forts du monde, *Lee Se-dol* [53]. Il s'est avéré qu'AlphaGo, adoptant des techniques de deep learning, a fait preuve d'une force surprenante et a battu Lee Se-dol par 4:1. De plus, les algorithmes de deep learning ont également montré des performances exceptionnelles dans la prédiction de l'activité de molécules médicamenteuses potentielles et des effets des mutations de l'ADN non codant sur l'expression des gènes.

Avec le développement rapide des techniques de calcul, un Framework puissant a été fourni par les ANN avec des architectures profondes pour l'apprentissage supervisé. D'une manière générale, l'algorithme du deep learning consiste en une architecture hiérarchique comportant de nombreuses couches dont chacune constitue une unité de traitement de l'information non linéaire. Dans ce chapitre, nous discutons des architectures profondes que nous avons utilisées dans nos travaux.

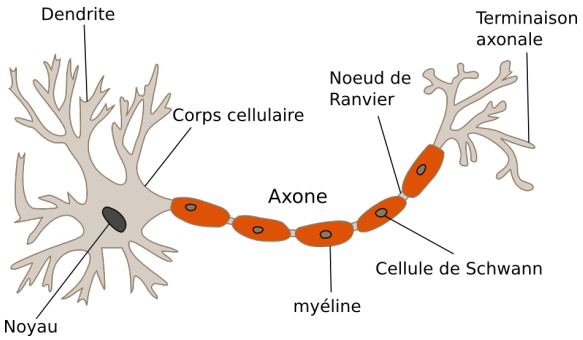


Figure 3.2: Illustration du modèle d'un neurones biologique.

3.4 Perceptron

Comment l'homme pense, parle, calcule, apprend? Deux approches ont été essentiellement explorées. La première approche consiste à commencer par une analyse logique des tâches de la cognition humaine, et à essayer de les reconstruire par programme. Cette approche a été aidée par l'intelligence artificielle symbolique traditionnelle et la psychologie cognitive. Cette approche est appelée cognitivisme. Cependant, dans la seconde approche, et étant donné que la pensée est produite par le cerveau, elle consiste à étudier d'abord le fonctionnement du cerveau, c'est cette approche qui a conduit à l'étude des réseaux de neurones (connexionnisme). Le connexionnisme est la démarche qui consiste à essayer de rendre compte de la cognition humaine par le biais des réseaux de neurones.

La deuxième approche a donc conduit à la définition et à l'étude des réseaux de neurones, qui sont des réseaux complexes d'unités de calcul élémentaires interconnectées. Il existe deux courants de recherche sur les réseaux de neurones : le premier est motivé par l'étude et la modélisation des phénomènes d'apprentissage naturels pour lesquels la pertinence biologique est importante ; le second est motivé par l'obtention d'algorithmes efficaces qui ne se préoccupent pas de la pertinence biologique. Nous adoptons le point de vue du second groupe. En effet, bien que les réseaux de neurones aient été définis sur la base de considérations biologiques, pour la plupart d'entre eux, et en particulier ceux étudiés dans ce chapitre, de nombreuses caractéristiques biologiques ne sont pas prises en compte.

Les neurones reçoivent des signaux (impulsions électriques) à travers des extensions très ramifiées de leur corps cellulaire (dendrites), et envoient des informations à travers de longues extensions (axones). Les impulsions électriques sont régénérées lorsqu'elles se déplacent le long de l'axone. La durée de chaque impulsion est de l'ordre de la milliseconde et son amplitude est d'environ 100 mV [54]. La figure 3.2 illustre un modèle biologique du neurone. Le contact entre deux neurones, de

l'axone à une dendrite, se fait par l'intermédiaire de synapses. Lorsqu'une impulsion électrique atteint la terminaison d'un axone, des neuro-médiateurs sont libérés et se lient aux récepteurs post-synaptiques des dendrites. Chaque neurone intègre en permanence jusqu'à un millier de signaux synaptiques. Ces signaux ne fonctionnent pas de manière linéaire, ce qui signifie qu'il y a un effet de seuil.

3.4.1 Neurone

Un neurone est une fonction de l'agrégation pondérée de ses nombreuses entrées : $\{x_0, \ldots, x_N\}$,

$$y = f(\sum_{i}^{N} w_i x_i + b), \tag{3.1}$$

où w_i est le poids de l'entrée x_i, f est une *fonction d'activation*, et b est le biais. Ceci est généralement exprimé plus simplement en notation matricielle, où chaque neurone est constitué d'un vecteur d'entrée $x = (x_0, \ldots, x_N)$, de poids $w = (w_0, \ldots, w_N)$ et d'un biais b, dont la sortie est,

$$y = f(w^T x + b). \tag{3.2}$$

Si nous supposons que la fonction f est une variante de la fonction *Heaviside*,

$$f = \begin{cases} 1 & if\, x \geqslant 0 \\ 0\,(or -1) & if\, x < 0 \end{cases}, \tag{3.3}$$

alors le neurone est aussi appelé perceptron, un simple classifieur binaire, et l'une des premières méthodes d'apprentissage connexionniste, inventée par Rosenblatt en 1957 [55]. Un réseau perceptron est un réseau de neurones mono-couche (c'est-à-dire un classifieur linéaire), tel que celui illustré à la figure 3.3. Le perceptron a également une interprétation géométrique [56], comme le montre la figure 3.3 et la figure 3.4. En 2D, par exemple, cela équivaut à l'équation de la droite. En supposant que notre perceptron n'a qu'une seule entrée, c'est-à-dire $N = 1$, si nous définissons $a \equiv w_0$, $x \equiv x_0$, $c \equiv b$, et $f(x) = x$,

$$y = ax + c. \tag{3.4}$$

En général, un perceptron définit un *hyperplan*, un collecteur de séparation de dimension $d - 1$ pour un espace d'entrée de dimension d, une droite en deux dimensions, ou un plan en trois dimensions. Les réseaux de neurones sont un classifieur discriminant, et chaque neurone peut être visualisé comme un hyperplan fonctionnant comme une frontière de décision unique.

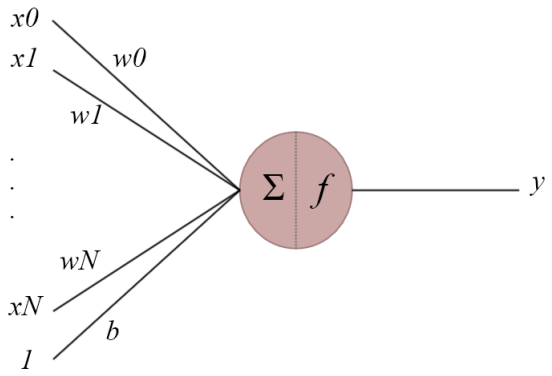


Figure 3.3: Une illustration d'un réseau de neurones artificiel typique *neurone*.

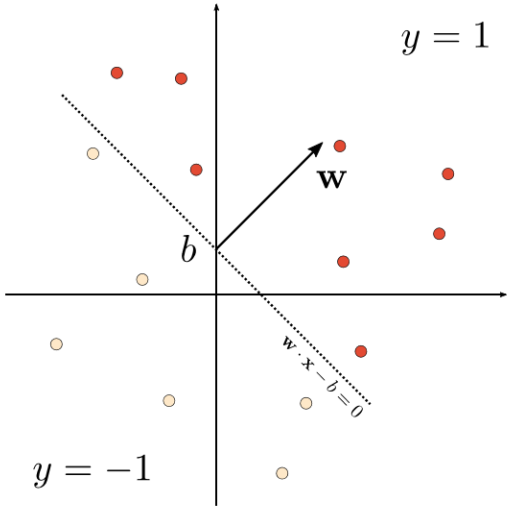


Figure 3.4: L'interprétation d'un perceptron comme un hyperplan orienté.

3.4.2 Limites des réseaux de neurones mono-couches

Un réseau de neurones mono-couche, tel que le perceptron, n'est qu'un classifieur linéaire et, en tant que tel, est inefficace pour apprendre une grande variété de tâches. Dans le livre Perceptrons [57], les auteurs ont notamment montré que les perceptrons monocouches ne pouvaient pas apprendre à modéliser des fonctions aussi simples que la fonction XOR, parmi d'autres problèmes de classification non linéairement séparables. Comme indiqué dans la figure 3.5, aucune droite unique ne peut séparer même un échantillonnage éparse de la fonction XOR — c'est-à-dire qu'elle n'est pas *linéairement séparable*. Au contraire, seule une composition de lignes est capable de séparer et de classer correctement cette fonction, ainsi que d'autres problèmes non linéairement séparables.

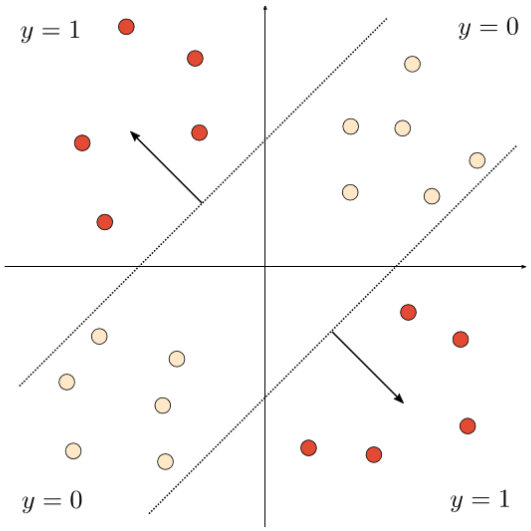


Figure 3.5: Une illustration de l'incapacité d'une seule ligne (c'est-à-dire un perceptron) à classer correctement la fonction XOR. Au lieu de cela, la composition de deux lignes est nécessaire pour séparer correctement ces échantillons, c'est-à-dire plusieurs couches.

À l'époque, il n'était pas évident de savoir comment entraîner les réseaux comportant plus d'une couche de neurones, car les méthodes d'apprentissage des poids des neurones, la règle d'apprentissage du perceptron [58] ou la règle du delta [59] pour les neurones généraux, ne s'appliquaient qu'aux réseaux à une seule couche.

3.5 Perceptron multicouche (MLP)

Les *perceptrons multi-couches (MLP: Multi-Layer Perceptrons)*, également appelés réseaux de neurones supervisés, *Feedforward supervised neural networks*, sont la quintessence des réseaux profonds. Ce sont des fonctions paramétriques définies par la composition de nombreuses fonctions paramétriques. Chacune de ces fonctions composantes a de multiples entrées et de multiples sorties. Dans la terminologie des réseaux de neurones, nous désignons chaque sous-fonction comme une couche du réseau, et chaque sortie scalaire de l'une de ces fonctions comme une unité ou parfois comme une caractéristique. Même si chaque unité met en œuvre une cartographie ou une transformation relativement simple de son entrée, la fonction représentée par l'ensemble du réseau peut devenir arbitrairement complexe.

Tous les algorithmes de deep learning ne peuvent pas être compris en termes de définition d'une fonction déterministe unique comme les MLP, mais tous ont la propriété commune de contenir de nombreuses couches de nombreuses unités. Nous

pouvons considérer le nombre d'unités de chaque couche comme étant la largeur d'un modèle de machine learning, et le nombre de couches comme sa profondeur. Les MLP fournissent un exemple conceptuellement simple d'un algorithme qui saisit les nombreux avantages qui découlent du fait d'avoir une largeur et une profondeur importantes. Les MLP sont également la technologie clé qui sous-tend la plupart des applications commerciales contemporaines du deep learning pour les grands ensembles de données.

Les réseaux de neurones nous permettent d'apprendre de nouveaux types de non-linéarité. Une autre façon de voir cette idée est que les réseaux de neurones nous permettent d'apprendre les caractéristiques fournies à un modèle linéaire. De ce point de vue, les réseaux de neurones nous permettent d'automatiser la conception des caractéristiques — une tâche qui, jusqu'à récemment, était réalisée progressivement et collectivement, grâce aux efforts combinés de toute une communauté de chercheurs.

Les MLP ont été parmi les premiers et les plus performants des algorithmes d'apprentissage non linéaire [60]. Ces réseaux apprennent au moins une fonction définissant les caractéristiques, ainsi qu'une fonction (typiquement linéaire) de mise en correspondance entre les caractéristiques et la sortie. Les couches du réseau qui correspondent aux caractéristiques plutôt qu'aux sorties sont appelées couches cachées. Cela s'explique par le fait que les valeurs correctes des caractéristiques sont inconnues. Les caractéristiques doivent être créées par l'algorithme d'apprentissage. L'entrée et la sortie du réseau sont par contre observées ou visibles dans les données d'entraînement. Il est à noter que dans la littérature, les DNN peuvent être confondus avec d'autres modèles de deep learning, mais dans la plupart des cas, les DNN font référence aux MLP ayant plus d'une couche cachée.

3.5.1 Fonctions d'activation

Comme nous l'avons vu avec le perceptron, la fonction d'activation pour un réseau monocouche peut fournir un moyen de pousser les sorties de chaque neurone vers un classement binaire. Cependant, la fonction d'activation est beaucoup plus importante dans les réseaux de neurones multi-couches. Sans une fonction d'activation non linéaire, même un grand réseau de neurones multicouche n'aurait que le pouvoir de représentation d'un classifieur linéaire — la composition des fonctions linéaires est une fonction linéaire. Pour cette raison, la *fonction d'activation* f est une fonction non linéaire appliquée à la sortie d'un neurone pour permettre aux réseaux multi-couches d'apprendre des fonctions non linéaires complexes,

$$y = f(\mathbf{w}^T\mathbf{x} + b). \tag{3.5}$$

Dans le domaine des réseaux de neurones, les fonctions d'activation ont été classiquement choisies comme une fonction sigmoïde, c'est-à-dire une fonction mettant en correspondance les entrées négatives avec les sorties négatives et les entrées positives avec les sorties positives avec une transition douce autour de $a = 0$. C'est une propriété agréable à avoir, puisque la fonction pousse encore les sorties du réseau vers un classement binaire, la fonction est non linéaire (donc la composition des fonctions est non triviale), et la fonction a des gradients bien définis. Parmi les exemples de fonctions sigmoïdes couramment utilisées, on peut citer la fonction logistique,

$$f(a) = \frac{1}{1 + e^{-a}}, \tag{3.6}$$

et la tangente hyperbolique,

$$f(a) = tanh(a). \tag{3.7}$$

Un problème avec les fonctions d'activation sigmoïdales est le fait que les gradients sont très faibles dans une grande partie du domaine de la fonction. Pour cette raison, et pour obtenir de meilleurs résultats empiriques, les réseaux de neurones modernes ont tendance à utiliser la fonction d'activation *rectified linear unit (ReLU)*,

$$f(a) = max(0, a). \tag{3.8}$$

3.5.2 Nombre de couches cachées !

Il a été prouvé que les réseaux de neurones comportant au moins une couche (infiniment large) *cachée* sont un approximateur universel — c'est-à-dire qu'un tel réseau de neurones peut théoriquement représenter n'importe quelle fonction [61, 62]. Cela contraste fortement avec les limites des réseaux de neurones sans couches cachées.

Cependant, dans la pratique, nous constatons qu'un réseau ne comportant qu'une seule couche cachée, même d'une très grande largeur, puisse apprendre à représenter des fonctions complexes aussi bien que des réseaux comportant de nombreuses couches cachées. En effet, l'observation selon laquelle les réseaux comportant de nombreuses couches cachées, ou les réseaux profonds, atteignent empiriquement une meilleure précision que les réseaux comportant peu de couches cachées, ou les réseaux peu profonds, représente une partie des progrès réalisés dans l'apprentissage avec les réseaux de neurones ces dernières années. [63]

3.6 Machine de Boltzmann restreinte (RBM)

La *machine de Boltzmann restreinte (RBM: Restricted Boltzmann Machine)* est un modèle de deep learning non supervisé, il est largement utilisé en deep learning en raison de leur importance historique et de leur simplicité. Le concept de RBM a été proposé pour la première fois par *Smolensky*, et est devenu très populaire depuis que Hinton a publié son travail [51] en 2006. Les modèles RBM ont été utilisés pour générer des modèles stochastiques des réseaux de neurones artificiels qui peuvent apprendre la distribution de probabilité par rapport à leurs entrées. Les modèles RBM consistent en une variante de machines de Boltzmann (BM). Les BM peuvent être interprétées comme des réseaux de neurones avec des unités de traitement stochastiques connectées de manière bidirectionnelle. Comme il est difficile d'apprendre les aspects d'une distribution de probabilité inconnue, les RBM ont été proposées pour simplifier la topologie du réseau et pour améliorer l'efficacité du modèle. Il est bien connu qu'une RBM est un cas particulier de *champs aléatoires de Markov* avec des unités stochastiques visibles dans une couche et des unités stochastiques observables dans l'autre couche.

Une RBM est composée de deux couches : la couche visible v de $V > 1$ unités et la couche cachée h de $H > 1$ unités. Les variables observables sont représentées dans les unités visibles, tandis que les variables latentes sont représentées par les unités cachées. En effet, les variables observables décrivent les données, tandis que les variables latentes décrivent le comportement des unités cognitives indéterminées [64]. La connexion des deux couches forme une interaction symbolique, il n'y a pas de connexion entre les unités d'une même couche alors que chaque unité d'une couche est connectée à toutes les unités de l'autre couche. La somme totale de tous les poids sur toutes les connexions représente l'*énergie* d'une RBM. En général, il existe deux versions de RBM : la *RBM Bernoulli-Bernoulli* (BB-RBM) et la *RBM Gaussian-Bernoulli* (GB-RBM).

La modèle BB-RBM n'accepte que les données binaires pour la couche visible, tandis que le modèle GB-RBM est plus approprié pour les données en valeur réelle. En effet, la couche visible accepte les vecteurs de valeur réelle, tandis que la couche cachée accepte les vecteurs binaires. La fonction d'énergie du modèle GB-RBM et du modèle BB-RBM sont définies dans Eq. 3.9 et Eq. 3.10, respectivement.

$$E(v,h)_{GB-RBM} = -\sum_{j=1}^{H} b_j^h h_j + \sum_{i=1}^{V} \frac{(v_i - b_i^v)^2}{2\sigma_i^2} - \sum_{i=1}^{V}\sum_{j=1}^{H} \frac{v_i}{\sigma_i} h_j w_{ij} \tag{3.9}$$

$$E(v,h)_{BB-RBM} = -\sum_{j=1}^{H} b_j^h h_j + \sum_{i=1}^{V} b_i^v v_i - \sum_{i=1}^{V}\sum_{j=1}^{H} v_i h_j w_{ij} \tag{3.10}$$

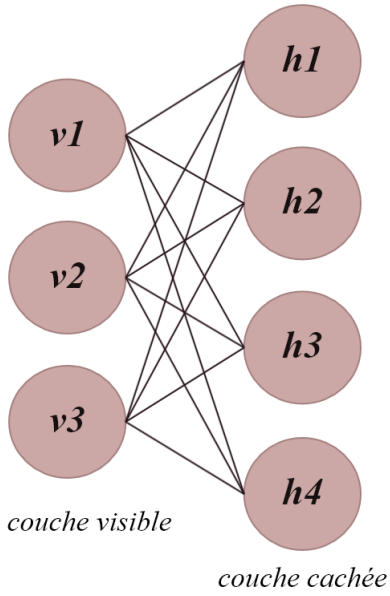


Figure 3.6: Un schéma conceptuelle de la machine de Boltzmann restreinte.

où v_i et h_j désignent les entrées pour les unités $i^{ème}$ et $j^{ème}$ dans les couches visible et cachée, respectivement ; w_{ij} représente la valeur de poids sur la connexion entre les unités i et j ; b_i^v et b_j^h sont les biais pour l'unité visible i et l'unité cachée j, respectivement. σ_i représente l'écart-type du bruit gaussien pour l'unité visible i. Par la suite, nous pouvons calculer la distribution de probabilité commune pour v et h comme suit,

$$p(v, h; \theta) = \frac{\exp(-E(v, h; \theta))}{Z} \tag{3.11}$$

où θ représente les poids et les biais, tandis que Z indique la fonction de partition, définie par,

$$Z = \sum_v \sum_h \exp(-E(v, h; \theta)) \tag{3.12}$$

Comme le montre la figure 3.6, les neurones sont limités pour former un graphique bipartite dans une RBM. On peut voir qu'il y a une connexion complète entre les unités visibles et les unités cachées, alors qu'aucune connexion n'existe entre les unités d'une même couche [65]. Pour entraîner une RBM, l'échantillonneur *Gibbs* est adopté. En partant d'un état aléatoire dans une couche et en effectuant un échantillonnage Gibbs, nous pouvons générer des données à partir d'une RBM. Une fois que les états des unités d'une couche sont donnés, toutes les unités des autres couches seront mises à jour. Ce processus de mise à jour se poursuivra jusqu'à ce que la distribution d'équilibre soit atteinte. Ensuite, les poids au sein d'une

RBM sont obtenus en maximisant la probabilité de cette RBM. Plus précisément, en prenant le gradient de la log-vraisemblance des données d'entraînement, les poids peuvent être mis à jour selon,

$$\frac{\partial \log p(v^0)}{\partial w_{ij}} = \left\langle v_i^0 h_j^0 \right\rangle - \left\langle v_i^\infty h_j^\infty \right\rangle \tag{3.13}$$

où $\left\langle v_i^0 h_j^0 \right\rangle$ et $\left\langle v_i^\infty h_j^\infty \right\rangle$ sont les corrélations lorsque les unités visibles et cachées sont dans la couche la plus basse et la couche la plus haute, respectivement. La preuve détaillée se trouve dans[51]. Il convient de noter que le processus d'entraînement sera plus efficace si l'on utilise l'algorithme de *contrastive divergence* (CD) basé sur les gradients. L'algorithme de CD pour l'entraînement des RBM a été développé par Hinton en 2002 [66]. La procédure de l'algorithme de CD à k-étapes est donnée dans l'algorithme 1.

Algorithm 1 k-étapes contrastive divergence pour les RBM

1: **Entrée:** $RBM(v_1, \ldots, V_H, h_1 \ldots, h_H)$, période d'apprentissage T, taux d'apprentissage ϵ
2: **Sortie:** La matrice de poids de la RBM ω, l'approximation du gradient Δw_{ij}, Δb_i^v et Δb_j^h pour $i = 1, \ldots, V$, $j = 1, \ldots, H$
3: Initialiser ω avec des valeurs aléatoires distribuées de manière uniforme $\in [0, 1]$, $\Delta w_{ij} = \Delta b_i^v = \Delta b_j^h = 0$ pour $i = 1, \ldots, V$, $j = 1, \ldots, H$
4: **for** t=1:T **do**
5: échantillonner $v_i^{(t)} P(v_i|h^{(t)})$ quand $i = 1, \ldots, V$
6: échantillonner $h_j^{(t+1)} P(h_j|v^{(t)})$ quand $j = 1, \ldots, H$
7: **for** $i = 1, \ldots, V$, $j = 1, \ldots, H$ **do**
8: $\Delta\omega_{ij} = \Delta\omega_{ij} + \epsilon \times P(h_j = 1|v^{(0)}) \times v_i^{(0)} - \epsilon \times P(h_j = 1|v^{(T)}) \times v_i^{(T)}$
9: $\Delta b_i^v = \Delta b_i^v + \epsilon \times (v_i^{(0)} - v_i^{(T)})$
10: $\Delta b_j^h = \Delta b_j^h + \epsilon \times [P(h_j = 1|v^{(0)}) - P(h_j = 1|v^{(T)})]$
11: $\omega = \omega + \epsilon \times \Delta\omega_{ij}$
12: **end for**
13: **end for**

En supposant que la différence entre le modèle et la distribution cible n'est pas importante, nous pouvons utiliser les échantillons générés par la chaîne de Gibbs pour approximer le gradient négatif. Idéalement, plus la longueur de la chaîne augmente, plus sa contribution à la probabilité diminue et tend vers zéro [67]. Cependant, dans [68], nous pouvons constater que l'estimation du gradient ne peut pas représenter le gradient lui-même. De plus, la plupart des composantes de la CD et le gradient de log-vraisemblance correspondant ont des signes égaux [69]. C'est pourquoi un algorithme plus pratique appelé *persistent contrastive divergence* a été proposé dans [70]. Dans cette approche, les auteurs ont suggéré de retracer

les états des chaînes persistantes plutôt que de rechercher la valeur initiale de la chaîne de Markov de Gibbs pour un vecteur donné. Les états des unités cachées et visibles dans la chaîne persistante sont renouvelés après la mise à jour de chaque poids. De cette façon, même un petit taux d'apprentissage ne causera pas beaucoup de différence entre les mises à jour et les états de la chaîne persistante tout en apportant des estimations plus précises.

Aujourd'hui, les modèles RBM jouent un rôle important dans diverses applications telles que la modélisation de sujets, la réduction de la dimensionnalité, le filtrage collaboratif, la classification et l'apprentissage de caractéristiques. Par exemple, une RBM peut être utilisé pour encoder les données et ensuite appliqué à l'apprentissage non supervisé pour la régression. En outre, les RBM peuvent être utilisés comme un modèle génératif. Nous pouvons calculer la distribution conjointe des unités visibles et cachées $P(v, h)$ avec la loi bayésienne. La probabilité conditionnelle d'une seule unité $P(h|v)$ peut également être calculée avec les RBM. Par conséquent, une RBM peut également être utilisé comme un modèle discriminant.

3.7 Réseau de croyance profond (DBN)

Comme mentionné dans la section précédente, les variables visibles et cachées ne sont pas indépendantes l'une de l'autre [65]. Pour explorer les dépendances entre ces variables, en 2006, Hinton a construit les *réseaux de croyance profonds (DBN: Deep Belief Networks)* en empilant un ensemble de RBM. Plus précisément, les DBN sont composés de plusieurs couches de variables stochastiques et latentes et peuvent être considérés comme une forme spéciale du modèle génératif probabiliste bayésien. Comparés aux réseaux de neurones artificiels, les DBN sont plus efficaces, surtout lorsqu'ils sont appliqués à des problèmes de données non étiquetées.

Le schéma du modèle est présenté dans la figure 3.7. On peut voir sur cette figure que dans un DBN, toutes les deux couches adjacentes forment une RBM. La couche visible de chaque RBM est connectée à la couche cachée de la RBM précédente et les deux couches supérieures sont non directionnelles. La connexion dirigée entre la couche supérieure et la couche inférieure est de type descendant. Les différentes couches de RBM dans un DBN sont entraînées de manière séquentielle : les RBM inférieures sont entraînées en premier, puis les plus élevées. Une fois que les caractéristiques sont extraites par la RBM supérieure, elles sont propagées vers les couches inférieures [71]. Par rapport à un modèle simple, le modèle empilé augmente la limite supérieure de la log-vraisemblance, ce qui implique des capacités d'apprentissage plus importantes [63].

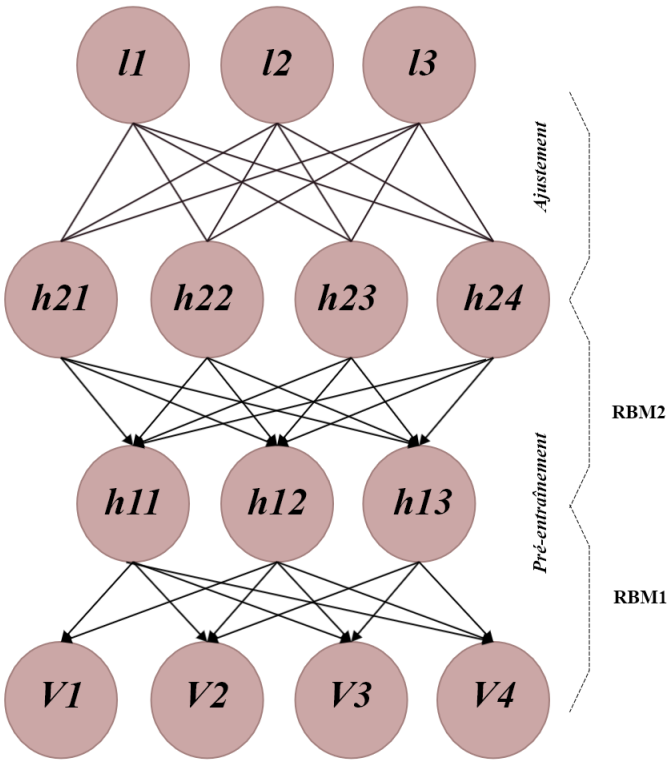


Figure 3.7: Un schéma conceptuelle du réseau de croyance profond.

Le processus d'entraînement d'un DBN peut être divisé en deux étapes : l'étape de pré-entraînement et l'étape d'ajustement. Dans la phase de pré-entraînement, un apprentissage non-supervisé est effectué de bas en haut pour l'extraction des caractéristiques ; tandis que dans la phase d'ajustement, un algorithme ascendant, basé sur un apprentissage supervisé, est effectué pour un ajustement ultérieur des paramètres du réseau. Nous notons que l'amélioration des performances des DBN peut être largement attribuée à l'étape de pré-entraînement au cours de laquelle les poids initiaux du réseau sont appris à partir de la structure des données d'entraînement. Par rapport aux poids initialisés de manière aléatoire, ces poids sont plus proches de l'optima global et peuvent donc apporter une meilleure performance.

L'algorithme CD introduit dans la section précédente peut être utilisé pour pré-entraîner un DBN. Cependant, les performances sont généralement insatisfaisantes, surtout lorsque les données d'entrée sont clampées. Pour surmonter ce problème, un algorithme d'apprentissage couche par couche a été introduit, qui optimise les poids d'un DBN à une complexité temporelle linéaire à la taille et à la profondeur du réseau [51]. Dans l'algorithme, *greedy layer-by-layer*, d'apprentissage couche par couche, les RBM qui constituent le DBN sont entraînées de manière séquentielle. Plus précisément, la couche visible de la RBM la plus basse est entraînée en premier

avec $h^{(0)}$ comme entrée. Les valeurs de la couche visible sont ensuite importées dans les couches cachées où les probabilités d'activation $P(h|v)$ des variables cachées sont calculées. La représentation obtenue dans la RBM précédente sera utilisée comme données d'apprentissage pour la prochaine RBM et ce processus d'entraînement continue jusqu'à ce que toutes les couches soient parcourues. Comme dans cet algorithme, l'approximation de la fonction de vraisemblance n'est nécessaire qu'en une seule étape, le temps d'apprentissage a été considérablement réduit. Le problème de sous-apprentissage (*underfitting*) qui se pose généralement dans les réseaux profonds peut également être surmonté lors du processus de pré-entraînement. Cet algorithme de pré-entraînement est également appelé *the greedy layer-by-layer unsupervised training algorithm*. Pour plus de clarté, nous avons fourni sa procédure d'implémentation dans l'algorithme 2.

Algorithm 2 L'algorithme Greedy layer-by-layer pour les DBN

1: **Entrée:** Vecteur visible d'entrée v_{in}, période d'apprentissage T, taux d'apprentissage ϵ, nombre de couches J
2: **Sortie:** La matrice de poids ω^i de couche i, $i = 1, 2, \ldots, J-2$.
3: Initialiser ω avec des valeurs aléatoires distribuées de manière uniforme $\in$ [0, 1], $h^0 = v_{in}$; où h^0 indique la valeur des unités dans la couche d'entrée. h^i représente la valeur des unités de la $i^{ème}$ couche. $couche = 1$;
4: **for** t=1:T **do**
5: **for** $couche < L$ **do**
6: L'échantillonnage de Gibbs h^{couche} en utilisant $P(h^{couche}|h^{couche-1})$
7: Calculer le CD dans l'algorithme 1
8: $\omega(couche)$ est réalisé en utilisant $\omega(t+1) = \omega(t) + \epsilon \times \Delta\omega$
9: **end for**
10: **end for**

Lors de la phase d'ajustement, les DBN sont entraînés avec des données étiquetées par l'algorithme *up-down* qui est une version contrastive de l'algorithme *wake-sleep* [72]. Pour connaître les limites des catégories du réseau, un ensemble d'étiquettes est mis en place à la couche supérieure pour le processus d'entraînement des poids. L'algorithme de rétro-propagation est également utilisé pour affiner les poids avec des données étiquetées [73].

Pour résumer, le processus d'entraînement d'un DBN comprend une procédure de pré-entraînement non supervisée, couche par couche, effectuée de manière ascendante et un processus de fin-tuning supervisé, de haut en bas. Le processus de pré-entraînement peut être considéré comme un apprentissage de caractéristiques grâce auquel une meilleure valeur initiale pour les poids peut être obtenue, et l'algorithme up-down est ensuite utilisé pour ajuster l'ensemble du réseau. Il

convient de mentionner qu'avec les DBN, les données non étiquetées sont traitées efficacement. De plus, les problèmes de sur-apprentissage et de sous-apprentissage peuvent également être évités [71].

3.8 Réseau de neurones convolutif (CNN)

Les *réseaux de neurones convolutifs (CNN: Convolutional Neural Networks)*, sont des modèles de deep learning, qui ont montré des performances satisfaisantes dans le traitement des données bidimensionnelles avec des topologies de type grille, comme les images et les vidéos [74]. L'architecture des CNN s'inspire de l'organisation du cortex visuel des animaux. Dans les années 1960, Hubel et Wisel [75] ont proposé un concept appelé champs réceptifs. Ils ont découvert que les arrangements complexes des cellules étaient contenus dans le cortex visuel animal chargé de la détection de la lumière dans les sous-régions du champ visuel qui se chevauchent et sont petites. De plus, le modèle de calcul *Neocognitron* a été introduit dans [76] avec des transformations d'images organisées hiérarchiquement. Cependant, le Neocognitron diffère des réseaux CNN par le fait qu'il ne nécessite pas de poids partagé.

Le concept des réseaux CNN s'inspire des *réseaux de neurones à retardement (TDNN: Time Delay Neural Networks)*. Dans un réseau TDNN, les poids sont partagés dans une dimension temporelle, ce qui entraîne une réduction des calculs. Dans les réseaux CNN, la convolution a remplacé la multiplication matricielle générale dans les réseaux de neurones standards. De cette façon, le nombre de poids est réduit, ce qui diminue la complexité du réseau. En outre, les images, en tant qu'entrées brutes, peuvent être directement importées dans le réseau, évitant ainsi la procédure d'extraction de caractéristiques dans les algorithmes d'apprentissage standard. Il convient de noter que les réseaux CNN sont la première architecture de deep learning véritablement réussie grâce à l'apprentissage efficace des couches hiérarchiques. La topologie des réseaux CNN exploite les relations spatiales pour réduire le nombre de paramètres dans le réseau, et la performance est donc améliorée en utilisant les algorithmes de rétro-propagation standard. Un autre avantage du modèle CNN est qu'il nécessite un pré-traitement minimal.

Avec le développement rapide des techniques de calcul, les techniques de calcul accéléré par GPU ont été exploitées pour entraîner plus efficacement les CNN. Aujourd'hui, les CNN ont déjà été appliqués avec succès à la reconnaissance de l'écriture manuscrite, à la détection des visages, à la reconnaissance du comportement, à la reconnaissance de la parole, aux systèmes de recommandation, à la classification des images et au traitement du langage naturel.

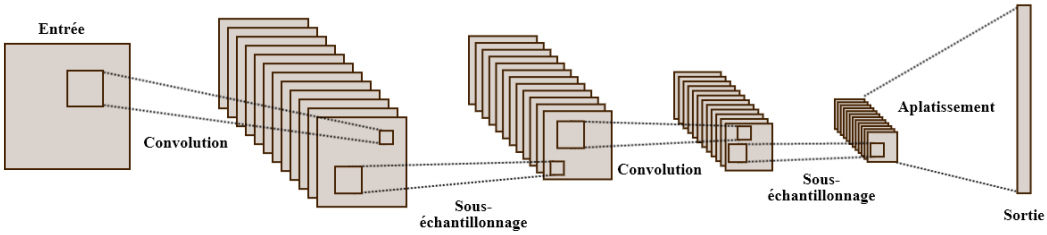


Figure 3.8: Un schéma du réseau de neurones convolutif. (La profondeur des matrices représente le nombre de filtres utilisés. La taille du vecteur de sortie est déterminé après l'aplatissement des matrices de la couche précédente et la concaténations des vecteurs résultants.)

Trois facteurs jouent un rôle clé dans le processus d'entraînement d'un CNN : l'absence d'interaction, le partage des paramètres et la représentation équivoque [63]. À la différence des réseaux de neurones traditionnels où la relation entre les unités d'entrée et de sortie est dérivée par la multiplication matricielle, les réseaux CNN réduisent la charge de calcul grâce à une faible interaction où les noyaux sont rendus plus petits que les entrées et utilisés pour l'image entière. L'idée de base du partage des paramètres est qu'au lieu d'apprendre un ensemble distinct de paramètres à chaque endroit, nous n'avons besoin d'apprendre qu'un seul ensemble d'entre ces paramètres, ce qui implique une meilleure performance du CNN. Le partage des paramètres a également doté le CNN d'une propriété attrayante appelée équivariance, ce qui signifie qu'à chaque fois que l'entrée change, la sortie change de la même manière. Par conséquent, moins de paramètres sont nécessaires pour le CNN par rapport aux autres algorithmes de réseaux de neurones traditionnels, ce qui entraîne une réduction de la mémoire et une amélioration en termes d'efficacité. Les composants d'une couche CNN standard sont présentés à la figure 3.8, et un schéma conceptuel d'un CNN standard est présenté à la figure 3.9.

Comme le montre la figure 3.9, le CNN est un réseau de neurones multicouche qui se compose de deux types de couches différentes, à savoir des couches de convolution et des couches de sous-échantillonnage [52, 63, 71]. Les couches de convolution et de sous-échantillonnage sont connectées alternativement et forment la partie centrale du réseau. Comme le montre la figure 3.8, l'image d'entrée est convoluée avec des filtres entraînables à tous les décalages possibles pour produire des cartes de caractéristiques dans la première couche de convolution. Une couche de poids de connexion est incluse dans chaque filtre. Normalement, quatre pixels de la carte de caractéristiques forment un groupe. En passant par une fonction sigmoïde, ces pixels produisent des cartes de caractéristiques supplémentaires dans la première couche de sous-échantillonnage. Cette procédure se poursuit et nous pouvons ainsi obtenir les

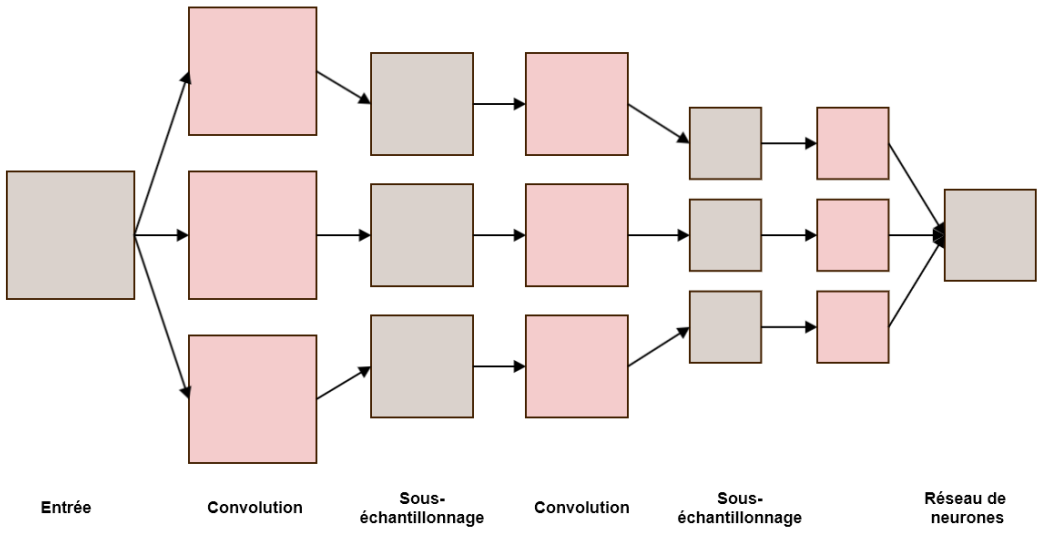


Figure 3.9: Un schéma conceptuelle du réseau de neurones convolutif en utilisant trois filtres.

cartes de caractéristiques dans les couches de convolution et de sous-échantillonnage suivantes. Enfin, les valeurs de ces pixels sont aplaties dans un vecteur unique qui va être considérer comme entrée du réseau MLP [74].

Généralement, les couches de convolution sont utilisées pour extraire des caractéristiques lorsque l'entrée de chaque neurone est liée au champ réceptif local de la couche précédente. Une fois que toutes les caractéristiques locales sont extraites, la relation de position entre elles peut être déterminée. Une couche de sous-échantillonnage est essentielle pour la cartographie des caractéristiques. Ces couches de cartographie des caractéristiques partagent les poids et forment un plan. De plus, pour obtenir une invariance d'échelle, la fonction sigmoïde est sélectionnée comme fonction d'activation en raison de sa légère influence sur le noyau de la fonction. Il convient également de noter que les filtres de ce modèle sont utilisés pour connecter une série de champs réceptifs qui se chevauchent et transformer l'entrée du lot d'images 2D en une seule unité dans la sortie.

Cependant, lorsque la dimension des entrées est égale à celle de la sortie du filtre, il sera difficile de maintenir l'invariance de la traduction avec des filtres supplémentaires. En raison de la dimensionnalité élevée, l'application d'un classifieur peut entraîner un sur-apprentissage. Pour résoudre ce problème, un processus de *pooling*, également appelé sous-échantillonnage est introduit pour réduire la taille globale du signal. Le sous-échantillonnage a déjà été appliqué avec succès pour la réduction de la taille des données dans la compression audio. Dans le filtre 2D, le sous-échantillonnage a également été utilisé pour augmenter l'invariance de position.

La procédure d'entraînement d'un CNN est similaire à celle d'un réseau de neurones standard utilisant la rétro-propagation. Plus précisément, Lecun et al. [77] ont introduit un gradient d'erreur pour entraîner les CNN. Dans la première étape, l'information est propagée vers l'avant à travers différentes couches. Les caractéristiques principales sont obtenues en appliquant des filtres numériques à chaque couche. Les valeurs de la sortie sont ensuite calculées. Au cours de la deuxième étape, l'erreur entre les valeurs attendues et les valeurs réelles de la sortie est calculée. La matrice de pondération est ensuite ajustée pour minimiser cette erreur, ce qui permet d'affiner le réseau. Contrairement aux autres algorithmes standard de classification des images, le pré-traitement n'est pas fréquemment effectué dans les CNN. Au lieu de définir des paramètres, comme c'est le cas avec les réseaux de neurones traditionnels, il suffit d'entraîner les filtres dans les CNN. De plus, dans l'extraction des caractéristiques, les CNN sont indépendants des connaissances préalables et des interférences humaines.

En 1998, la méthode du *max-pooling* a été proposée dans *LeNets* pour le sous-échantillonnage [78]. En résumant les statistiques des sorties proches, une fonction de pooling est utilisée pour remplacer la sortie du réseau à une certaine position. En utilisant la méthode max-pooling, nous pouvons obtenir la sortie maximale dans un voisinage rectangulaire. La procédure de pooling peut également rendre la représentation invariante aux traductions de l'entrée. Maintenant, en ajoutant une couche max-pooling entre les couches convolutionnelles, l'abstraction spatiale augmente avec l'augmentation de l'abstraction des caractéristiques.

Comme mentionné dans [79], le pooling est utilisée pour obtenir l'invariance dans les transformations d'images. Ce procédé permet d'obtenir une meilleure robustesse au bruit. Il est souligné que les performances des différentes méthodes de pooling dépendent de plusieurs facteurs, tels que la résolution à laquelle les éléments de bas niveau sont extraits et les liens entre les cardinalités de l'échantillon. En 2011, Boureau [80] a constaté que même si les caractéristiques sont très différentes, il est possible de les regrouper tant que leurs emplacements sont proches. En outre, il a constaté qu'il est possible d'obtenir de meilleures performances en effectuant le regroupement avant la phase de pooling. Dans [81], il est montré que de meilleures performances de pooling peuvent être obtenues en apprenant les champs réceptifs de manière plus adaptative. Plus précisément, en utilisant le concept de *sur-complétude*, un algorithme d'apprentissage efficace est proposé pour accélérer le processus d'entraînement basé sur une sélection incrémentale de caractéristiques.

3.9 Conclusion

Dans ce chapitre, nous avons présenté l'histoire de deep learning et ses architectures et ses algorithmes d'entraînement (pour les algorithmes d'entraînement du perceptron et le MLP, voir Annexe. A). Dans les chapitres suivants, les architectures de deep learning suivantes sont utilisées : le réseau de neurones profond (DNN), la machine de Boltzmann restreinte (RBM), le réseau de croyance profond (DBN), le réseau de neurones convolutif (CNN). Dans le chapitre suivant, nous présentons les corpus de reconnaissance automatique du locuteur que nous avons utilisés dans notre recherche.

Partie III

Contributions

4

Corpus de Parole pour la RAL

Sommaire

4.1 Introduction

Les corpus de parole sont considérés comme la matière première pour le développement et l'évaluation des systèmes de RAL. Cependant, la grande majorité des corpus de parole, destinés à la recherche sur la RAL, ne sont pas gratuits, et les autres qui sont librement accessibles ne sont pas suffisamment importantes pour le développement et l'évaluation des systèmes de RAL. Ce manque de corpus open source et de grande taille représente un obstacle majeur à la promotion de la recherche sur les systèmes de RAL, en particulier dans les laboratoires disposant de moins de ressources. Pour surmonter ce problème, nous avons développé un

corpus de parole open source, appelé *FSCSR* [1], avec plus de 530 locuteurs de plusieurs langues. Une partie du corpus proposé est collectée et enregistrée sous notre supervision, et la partie restante est collectée à partir du site web Voxforge [1] en utilisant une nouvelle approche semi-automatique. Dans ce chapitre, la section 4.2 présente un aperçu sur les corpus de parole existants. Dans la section 4.3, nous présentons en détails le corpus de parole proposé, et la procédure complète de collecte des données. Nous présentons aussi les résultats des expériences menées sur le corpus, en utilisant les approches *i*-vector/PLDA et GMM/UBM. Dans la section 4.4, nous présentons également un corpus de parole gratuit et open source pour la RAL appelé *THUYG* [82].

4.2 Aperçu des corpus de parole

Afin de promouvoir la recherche sur les systèmes de reconnaissance du locuteur, l'Institut national américain des normes et de la technologie (NIST) a coordonné une série d'évaluations de RAL avec des corpus de parole et des protocoles d'évaluation standard [83]. Ces évaluations, qui ont débuté en 1997, ont posé des tâches difficiles qui incitent les équipes de recherche de la communauté scientifique à améliorer continuellement leurs systèmes. De plus, une grande variété de corpus de parole ont été développés et proposés à la communauté des locuteurs [84, 85]. La diversité de ces corpus de parole est principalement due à plusieurs facteurs, tels que : le nombre et la diversité des locuteurs, les langues parlées (par exemple : monolingue [86] ou multilingue [87]), le nombre de séances par locuteur et la durée entre chaque deux séances consécutives, le type de parole (par exemple : phrase fixe, chiffres affichés, phrases lues, conversation), le matériel d'enregistrement et le canal utilisé pendant l'enregistrement du corpus de parole (par exemple : microphone à large bande, combinés téléphoniques variables), le type d'environnement (par exemple : environnements contrôlés / environnements non contrôlés à la maison, au bureau ou dans la rue), etc.

Les corpus de parole les plus utilisés dans la communauté de reconnaissance des locuteurs sont les suivants : TIMIT et ses dérivés, SIVA, PolyVar, POLYCOST, YOHO, KING-92 et Mixer 6 [85]. Il est certain que l'existence de ces corpus de parole standard permet aux chercheurs d'évaluer et de comparer les performances de leurs systèmes de manière statistiquement cohérente et, par conséquent, de contribuer de manière significative à promouvoir la recherche sur les systèmes de RAL. Cependant, la grande majorité des grands corpus de parole ne sont pas

[1] http://www.voxforge.org/

gratuits pour les chercheurs, ce qui représente souvent un obstacle majeur à la recherche sur les systèmes de RAL dans les laboratoires disposant de ressources insuffisants. Pour faire face à ce problème, nous [1] et [82] proposons deux corpus de parole open sources et gratuits pour la RAL : FSCSR et THUYG.

4.3 Corpus FSCSR

4.3.1 Procédure de collecte des données

Le corpus de parole proposé est composé de deux parties. La première partie est collectée et enregistrée, sous notre supervision, par des locuteurs volontaires arabes, et la partie restante est collectée auprès de Voxforge en utilisant une nouvelle approche semi-automatique.

4.3.1.1 Collecte d'enregistrements vocaux auprès des locuteurs bénévoles arabes

La première partie du corpus de parole est composée de fichiers de parole enregistrés par des locuteurs volontaires arabes, âgés de 18 à 30 ans. Chaque locuteur a enregistré au moins deux sessions séparées par environ deux à trois semaines. Le type de parole comprend le monologue libre et le texte lu. Afin de couvrir un large éventail d'environnements acoustiques réels, les locuteurs ont été enregistrés à partir de différents endroits, par exemple à la maison, au bureau, etc. L'équipement d'enregistrement était composé de trois smartphones.

4.3.1.2 Collecte d'enregistrements vocaux auprès de Voxforge

VoxForge est un projet ambitieux qui vise à collecter des données de parole transcrite auprès de *donateurs* de voix humaine, afin de développer des modèles acoustiques pour les moteurs de reconnaissance vocale. Tout locuteur volontaire peut s'inscrire sur le site de Voxforge et mettre ses propres enregistrements vocaux à la disposition des chercheurs et des développeurs de moteurs de RAP. Bien que l'objectif principal du projet Voxforge soit de construire un corpus de parole sous licence GPL [2] pour le développement de moteurs de RAP. De nombreux chercheurs travaillent sur les systèmes de RAL ont utilisé les enregistrements vocaux de Voxforge pour l'évaluation de leurs systèmes [88, 89].

[2]La licence publique générale GNU, ou GNU General Public License (son seul nom officiel en anglais, communément abrégé GNU GPL, voire simplement « GPL »), est une licence qui fixe les conditions légales de distribution d'un logiciel libre du projet GNU.

Cependant, Voxforge est un site web ouvert où chacun peut enregistrer ses enregistrements vocaux autant de fois qu'il le souhaite en utilisant différents noms d'utilisateur, ce qui rend donc l'utilisation directe des enregistrements vocaux enregistrés peu fiable pour l'évaluation des systèmes de RAL. Par conséquent, l'utilisation des fichiers de parole de Voxforge pour l'évaluation des systèmes de RAL doit être précédée d'une étape de sélection qui **(i)** élimine les fichiers de parole enregistrés par les même locuteurs et enregistrés sous plusieurs nom d'utilisateur, et **(ii)** élimine les locuteurs redondants enregistrés sous des noms d'utilisateurs différents. Pour cette raison, nous proposons une approche de sélection efficace, qui consiste, comme le montre la figure 4.1, en une première étape, manuelle, à vérifier que les fichiers de parole enregistrés sous le même nom d'utilisateur sont réellement destinés au même locuteur, et une seconde étape semi-automatique visant à éliminer les locuteurs enregistrés redondants sous des noms d'utilisateur différents.

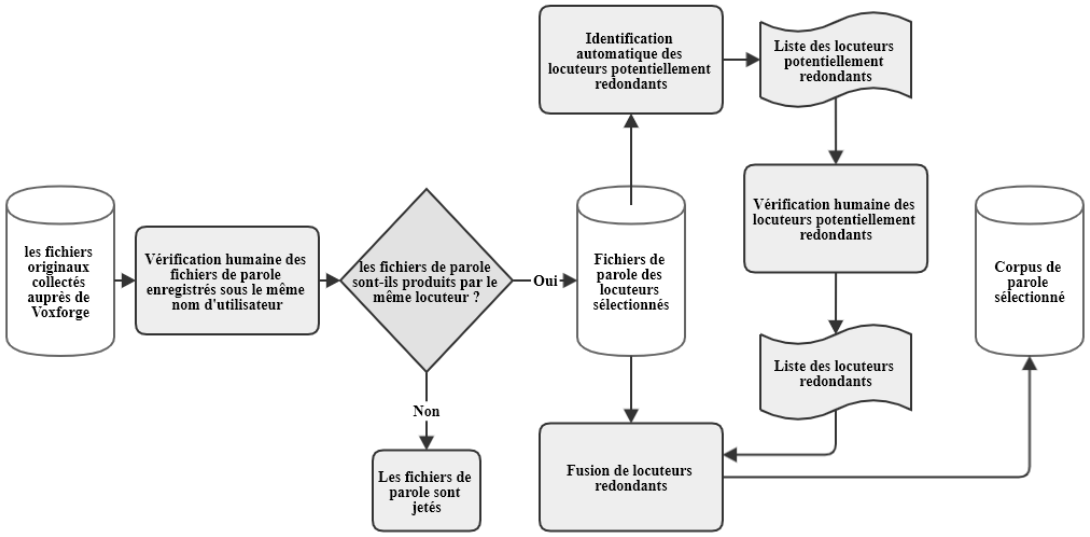


Figure 4.1: Un organigramme de l'approche de sélection proposée.

Au cours de la première étape, les fichiers de parole enregistrés sous le même nom d'utilisateur sont écoutés et vérifiés par au moins trois opérateurs, si les fichiers de parole ne proviennent pas du même locuteur ils sont rejetés. Dans la deuxième étape, c'est-à-dire l'étape semi-automatique, les fichiers de parole des locuteurs sélectionnés sont présentés à un système automatique qui produit une liste de locuteurs potentiels redondants. La liste produite est ensuite vérifiée par au moins trois opérateurs et utilisée pour fusionner les fichiers de parole des locuteurs redondants.

L'idée de base du système automatique qui détecte les locuteurs potentiellement redondants est décrite dans la figure 4.2. Tout d'abord, les fichiers de parole de chaque locuteur sélectionné S_i sont divisés en deux ensembles, un ensemble

d'entraînement $\dot{S}_i = \{F_1^{(i)}, \ldots, F_7^{(i)}\}$ composé de sept fichiers de parole, et un ensemble de test $\ddot{S}_i = \{F_8^{(i)}, \ldots, F_{T^{(i)}}^{(i)}\}$ composé des autres fichiers de parole. Ensuite, les vecteurs caractéristiques du $i^{ème}$ locuteur sont extraits de leurs fichiers de parole d'entraînement S_i. Ensuite, ils sont utilisés pour construire un modèle de référence GMM, $\Lambda^{(i)}$, pour ce locuteur. Ensuite, chaque fichier de parole de test $F_{1\leq t\leq T_i}^{(i)}$ de chaque locuteur est comparé à tous les modèles entraînés $\{\Lambda^{(i)}|1 \leq i \leq N\}$ et le locuteur S_j dont le modèle correspond le mieux $F_{1\leq t\leq T^{(i)}}^{(i)}$ est déterminé :

$$S_j = \arg \max_{1\leq n\leq N} P(F_t^{(i)}|\Lambda^{(n)}) \tag{4.1}$$

Enfin, si le locuteur identifié S_j est différent du locuteur cible S_i, les locuteurs S_i et S_j sont annotés comme locuteurs potentiellement redondants et ajoutés à la liste des locuteurs redondants potentiels.

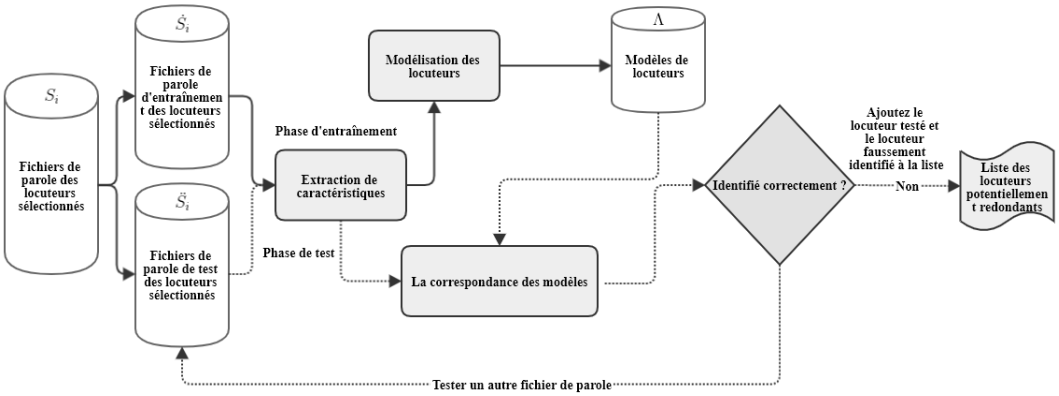


Figure 4.2: Un organigramme du processus d'identification automatique des locuteurs potentiellement redondants.

4.3.2 Description du corpus

Le corpus de parole FSCSR est constitué de plus de 11 heures de signaux de parole actifs, parlés par 536 locuteurs de différentes langues (anglais, français, arabe, italien, allemand et espagnol). Les fichiers de parole sont numérisés à 16 kHz avec une quantification à 16 bits et stockés en format *wav*. Les quantités relatives de locuteurs du FSCSR par langue et par sexe sont illustrées dans la figure 4.3.

L'ensemble du corpus de paroles, comme le montre le tableau 4.1, a été divisé en trois ensembles de données : le premier ensemble de données est composé de 100 locuteurs (50 masculins et 50 féminins), consacré à l'entraînement du modèle UBM, le deuxième ensemble de données est composé de locuteurs clients et le troisième ensemble de données est composé de locuteurs clients et imposteurs. Les signaux de parole des locuteurs clients sont divisés en deux ensembles : l'ensemble des locuteurs d'entraînement et l'ensemble des locuteurs de test.

Tableau 4.1: L'architecture du corpus FSCSR. (Nbr = Nombre)

Langues	Nbr locuteurs UBM	Nbr locuteurs clients	Nbr Imposteurs
Anglais	24	66	54
Français	12	68	30
Arabe	3	52	8
Allemand	29	-	-
Espagnol	18	78	2
Italienne	14	35	43
Total	100	299	137

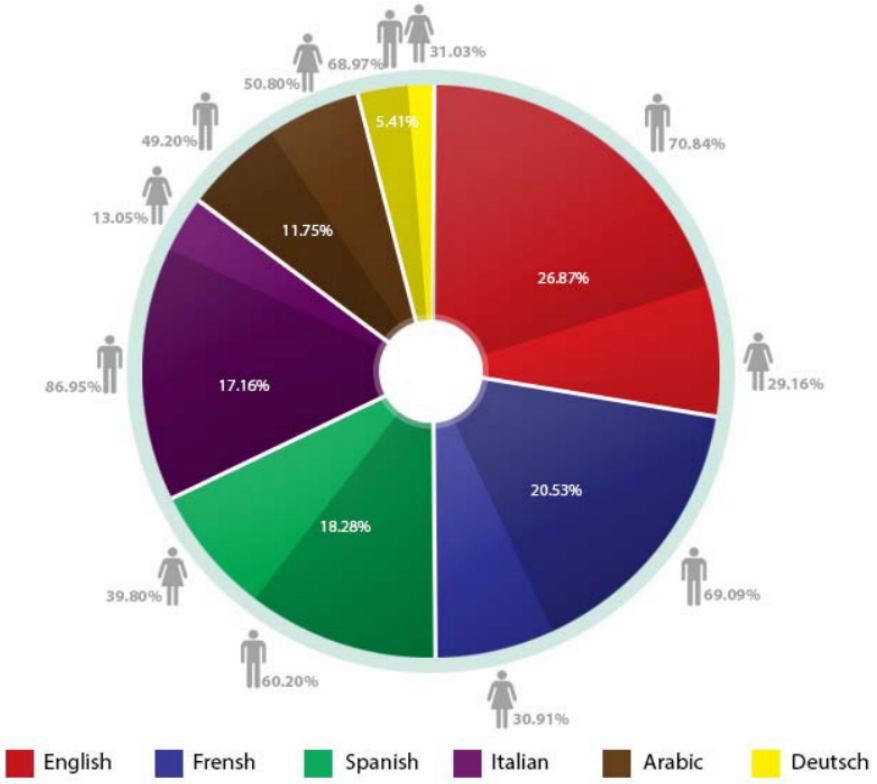


Figure 4.3: Le nombre relatif de locuteurs par langue et par sexe.

4.4 Corpus THUYG

Le corpus de parole THUYG est constitué de plus de 20 heures de signaux de parole prononcés par 371 locuteurs. Les signaux ont été enregistrés dans un bureau silencieux par le même microphone de type carbone. Le taux d'échantillonnage est de 16 kHz et une quantification à 16 bits. Les locuteurs étaient tous des étudiants âgés de 19 à 28 ans, et ils sont de langue maternelle ouïghoure et originaires de 30 comtés. Les phrases ont été extraites de domaines généraux, dont des romans, des journaux

et divers types de livres. Le corpus a été enregistré de janvier à septembre en 2012.

Tableau 4.2: L'architecture du corpus THUYG. (nbr) est pour nombre, et (hrs) est pour hours.

	Locuteurs UBM	Locuteurs clients
Nbr Féminins	100	Entraînement: 87, Test: 1371
Nbr Masculins	100	Entraînement: 66, Test: 990
Nbr Énoncés	4771	Entraînement: 153, Test: 2361
Durée (hrs)	13.15	Entraînement: 1.28, Test: 6.56

Le profil des données du corpus est présenté dans le tableau 4.2. L'ensemble du corpus est divisé en deux groupes de données : le groupe est réservé aux locuteurs UBM qui comprend 4771 énoncés prononcés par 200 locuteurs, et est utilisé pour entraîner des modèles comprenant l'UBM dans le Framework GMM/UBM, la matrice T dans le modèle i-vector, et les paramètres $\{m, U, V, \sum\}$ dans le PLDA. Le deuxième groupe est réservé aux locuteurs clients, il est divisé en deux sous-groupe: un sous-groupe pour l'entraînement et un autre pour le test. Chaque énoncé du sous-groupe d'entraînement est d'une durée de 30 seconds, or la durée des énoncés du groupe de test est de 10 seconds. En outre, la base de données comporte trois signaux sonores : le bruit blanc, le bruit de la cafétéria et le bruit de la voiture. De plus, le bruit est mélangé aux signaux de parole de manière aléatoire, avec un niveau de SNR[3] spécifié.

4.5 Protocole expérimental et résultats

Le signal de parole est divisé en petites trames de 25 ms, et la fenêtre de Hamming est utilisée avec un glissement de 10 ms. En conséquence, des vecteurs caractéristiques MFCC à 39 dimensions sont extraits, y compris le vecteur MFCC statique à 13 dimensions, dans lequel le coefficient $C0$ est remplacé par l'énergie, et leurs première et deuxième dérivées. Afin d'éliminer l'effet de canal des vecteurs caractéristiques, nous avons appliqué la normalisation de la moyenne et de la variance cepstrales, *Cepstral mean and variance normalization* (CMVN) [90].

Par conséquent, nous proposons une évaluation comparative pour le corpus THUYG comme suit:

[3]Le rapport signal/bruit (SNR ou S/N) est une mesure utilisée en science et en ingénierie qui compare le niveau d'un signal souhaité au niveau du bruit de fond. Le SNR est défini comme le rapport entre la puissance du signal et la puissance du bruit, souvent exprimé en décibels.

- L'évaluation est répartie en deux catégories: l'évaluation à ressources limitées et l'évaluation à ressources ouvertes. Dans la première évaluation, tous les modèles sont entraînés avec le corpus THUYG uniquement, tandis que dans la seconde, toutes les données sont autorisées pour entraîner les modèles ;

- L'évaluation dépend du genre, conformément à la convention du plan d'évaluation de NIST [91] ;

- L'évaluation est menée selon trois types de bruit : le bruit blanc, le bruit de la cafétéria et le bruit des voitures. Les données d'inscription et les données de test peuvent être corrompues uniquement par le même type de bruit, mais la corruption peut se faire dans des SNR différents, choisis parmi (0db, 9db, nette), où "nette" signifie aucune corruption par le bruit ;

- La durée du signal de parole d'entraînement est fixée à 30 secondes alors que la durée du signal de parole de test est fixée à 10 secondes.

La première expérience examine les performances du système de base sur la tâche à ressources limitées, c'est-à-dire que seules les données du corpus THUYG sont utilisées pour l'entraînement des modèles. Les résultats sur les locuteurs féminins et masculins sont présentés dans le tableau 4.3, où les données sont bruitées en utilisant le bruit de la cafétéria, et trois conditions de SNR sont rapportées (Nette, 9db et 0db). Comme le montrent le tableau 4.2, la phase de test compte 87 locuteurs féminins et 66 locuteurs masculins ; ces locuteurs sont testés les uns par rapport aux autres, ce qui donne un total de 119,277 essais pour les données féminines et de 65,340 pour les données masculines.

Les résultats du tableau 4.3 montrent qu'en cas d'entraînement et de parole propres, les performances sont plutôt bonnes. Mais en cas de bruit, les taux d'erreurs sont sensiblement augmentés, que le bruit concerne l'entraînement ou la parole de test. Plus le bruit est important, plus la réduction des performances est significative. L'impact du bruit sur la parole de test est plus évident que sur la parole d'entraînement. Par exemple, pour les données féminines, dans le cas de $SNR = 9db$, i-vector/PLDA, le taux d'erreur est de 10,94% EER avec du bruit sur les données d'entraînement, alors que le taux d'erreur est de 17,43% EER avec du bruit sur les données de test. En outre, si le niveau de SNR correspond à celui des paroles d'entraînement et de test, les performances ont tendance à être moins affectées.

Le corpus THUYG est relativement petit pour l'entraînement du modèle UBM dans GMM/UBM et i-vector/PLDA. Dans la tâche de ressource ouverte, des données supplémentaires sont autorisées pour améliorer la qualité du modèle. Les

Tableau 4.3: Présentation des résultats (en % EER) des méthodes suivantes : GMM/UBM et *i*-vector/PLDA, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db.

Locuteurs féminins		Méthodes	
Cible (SNR)	Test (SNR)	GMM/UBM	*i*-vector/PLDA
Nette	Nette	5.14	4.01
Nette	9db	19.85	17.43
Nette	0db	30.15	27.43
9db	Nette	12.98	10.94
9db	9db	10.12	8.83
9db	0db	16.73	15.75
0db	Nette	20.08	18.82
0db	9db	15.74	13.75
0db	0db	18.78	17.58
AVG EER		16.62	14.95

Locuteurs masculins		Méthodes	
Cible (SNR)	Test (SNR)	GMM/UBM	*i*-vector/PLDA
Nette	Nette	5.12	3.87
Nette	9db	19.10	17.90
Nette	0db	29.33	26.58
9db	Nette	12.49	10.66
9db	9db	9.82	7.01
9db	0db	16.11	16.41
0db	Nette	19.99	15.43
0db	9db	15.64	12.77
0db	0db	18.25	16.95
AVG EER		16.21	14.18

corpus Fisher [92] et FSCSR sont utilisés comme données supplémentaires pour l'entraînement de l'UBM. Notons que Fisher est un corpus anglais et que FSCSR est un corpus multilingue, mais il s'avère que les modèles entraînés en utilisant le corpus Fisher donne des résultats meilleurs, comme le montrent le tableau 4.4. Cela démontre d'une part que la RAL est largement indépendante de la langue, et d'autre part qu'une grande base de données d'entraînement est importante pour la construction de systèmes RAL.

Tableau 4.4: Présentation des résultats (en % EER) des méthodes suivantes : GMM/UBM et *i*-vector/PLDA, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db.Le modèle UBM est entraîné en utilisant les corpus FSCSR et Fisher.

Locuteurs féminins		Méthodes			
Cible (SNR)	Test (SNR)	GMM/UBM		*i*-vector/PLDA	
		FSCSR	Fisher	FSCSR	Fisher
Nette	Nette	6.74	4.32	3.88	2.33
Nette	9db	15.39	12.84	13.24	4.45
Nette	0db	25.04	21.62	16.67	8.83
9db	Nette	12.11	8.95	6.40	4.01
9db	9db	10.01	6.17	6.48	3.72
9db	0db	13.52	8.14	11.42	5.76
0db	Nette	16.28	12.54	14.10	6.57
0db	9db	14.66	8.52	8.30	4.74
0db	0db	13.72	9.99	12.34	5.47
AVG EER		14.16	10.34	10.31	5.06
Locuteurs masculins		Méthodes			
Cible (SNR)	Test (SNR)	GMM/UBM		*i*-vector/PLDA	
		FSCSR	Fisher	FSCSR	Fisher
Nette	Nette	9.05	4.16	3.75	2.16
Nette	9db	16.45	12.60	13.38	4.12
Nette	0db	25.61	21.02	15.42	7.77
9db	Nette	13.66	8.27	6.54	3.91
9db	9db	9.98	5.48	6.22	3.66
9db	0db	12.81	7.83	12.96	5.34
0db	Nette	16.22	11.90	11.20	6.01
0db	9db	15.52	8.00	8.36	4.80
0db	0db	16.31	9.35	11.51	4.99
AVG EER		15.07	9.85	9.93	4.75

4.6 Conclusion

Dans ce chapitre, nous avons présenté deux corpus de parole. Le premier était notre proposition : "Le corpus multilingue FSCSR". Une partie de ce corpus de parole proposé est collectée et enregistrée sous notre supervision, et la partie restante est collectée de manière précise auprès de Voxforge en utilisant une nouvelle approche semi-automatique. Le second est le corpus "THUYG", qui est un corpus ouïgour. En outre, nous avons évalué ces deux corpus en présentant les résultats d'une étude

comparative utilisent les deux méthodes de l'état de l'art, à savoir *i*-vector/PLDA et GMM/UBM. Sur la base des résultats, nous avons décidé d'utiliser les résultats présentés dans le tableau 4.4, en utilisant le corpus THUYG, comme système de référence, qui sera comparé avec nos approches présentées dans les chapitres qui suivent. Le corpus FSCSR est utilisé comme corpus de développement.

5

Méthode de Plus Proche Cluster — MFCC/NearC

Sommaire

5.1 Introduction

De nos jours, les modèles stochastiques sont l'état de l'art pour la RAL. Cependant, ils sont coûteux en termes de temps et peuvent nécessiter beaucoup plus de données lors de la phase d'entraînement. Dans ce chapitre, nous proposons une nouvelle méthode d'évaluation des locuteurs basée sur le notion de distance pour la mesure de similarité dans la vérification automatique du locuteur en mode indépendante du texte [2]. L'idée de base de notre approche est de mesurer la similarité entre les locuteurs en utilisant directement des vecteurs caractéristiques (MFCC), afin de préserver et de tirer parti des caractéristiques spécifiques de locuteur. Des

expériences sur deux corpus de reconnaissance automatique du locuteur, THUYG et FSCSR, confirment notre hypothèse. Les résultats montrent que notre approche surpasse largement les approches de l'état de l'art : GMM/UBM et *i*-vector/PLDA dans les conditions d'une parole nette. Dans la section 5.2, nous présentons la méthode proposée. Dans la section 5.3, nous donnons les résultats et nous les discutons, et dans la section 5.4, nous concluons le chapitre.

5.2 Méthodologie

La modélisation du locuteur est la phase la plus importante d'un système de RAL. Dans les modèles stochastiques, nous avons besoin de beaucoup plus de données pour modéliser les locuteurs. En outre, l'adaptation des locuteurs à partir d'un modèle UBM peut atténuer les caractéristiques spécifiques de locuteur. Il en va de même pour la quantification vectorielle, l'idée de regrouper les vecteurs caractéristiques cibles et de ne conserver que les centroïdes des groupes [28, 93–95] peut négliger les vecteurs caractéristiques spécifiques du locuteur. Par conséquent, une modélisation efficace du locuteur doit prendre en compte la contrainte de conserver toutes les caractéristiques.

L'objectif de ce chapitre est de fournir une nouvelle méthode d'évaluation basée sur le calcul d'une distance pour la mesure de la similarité dans la vérification automatique du locuteur en mode indépendante du texte, avec l'idée de cibler et de préserver les caractéristiques spécifiques du locuteur. Notre proposition est basée sur trois méthodes, dans lesquelles chacune est une version améliorée de la précédente.

5.2.1 Méthode (i): la première étape

La première méthode proposée repose sur la mesure de la similarité entre les vecteurs de S_{test} (segment de parole du locuteur test) et de S_{cible} (segment de parole du locuteur cible), en utilisant l'équation de calcul de la distance définie comme suit,

$$d(S_{test}, S_{cible}) = \frac{1}{N} \sum_{i=1}^{N} d(S_{test}^{(i)}, S_{cible}), \tag{5.1}$$

avec :

$$d(S_{test}^{(i)}, S_{cible}) = \min_{j=1,\ldots,M} d(S_{test}^{(i)}, S_{cible}^{(j)}), \tag{5.2}$$

où N et M se réfèrent respectivement au nombre total de vecteurs de S_{test}, et au nombre total de vecteurs de S_{cible}.

Cette méthode consiste à renvoyer la distance entre un vecteur $S_{test}^{(i)}$ et le vecteur $S_{cible}^{(j)}$ le plus proche. Ensuite, elle retourne la moyenne des distances résultantes.

5.2.2 Méthode (ii): le regroupement en clusters

Dans note première proposition, notre hypothèse initiale considère que la caractéristique d'un locuteur peut être représentée par un vecteur, et qu'entre-temps, un vecteur peut comprendre plus d'une caractéristique. Cependant, le signal de parole a un nombre inconnu de caractéristiques, et par conséquent nous pouvons avoir des vecteurs très similaires. Pour cette raison, nous avons choisi l'algorithme de mise en clusters *k-means* [96] pour partitionner les vecteurs MFCC en clusters, dans l'espoir de représenter une caractéristique ou un ensemble de caractéristiques similaires par un cluster.

Étant donné un ensemble de vecteurs $S = \{s^{(1)}, \ldots, s^{(n)}\}$, la mise en clusters k-means vise à partitionner les n vecteurs en $K(\leq n)$ ensemble $C = \{C^{(1)}, \ldots, C^{(K)}\}$ de manière à minimiser la somme des carrés à l'intérieur d'un cluster, *within-cluster sum of squares (WCSS)*, c'est-à-dire la variance. Formellement, l'objectif est de trouver,

$$\arg\min_{C} \sum_{i=1}^{K} \sum_{s \in C_i} \|s - \mu_i\| = \arg\min_{C} \sum_{i=1}^{K} |C_i| \, \mathbf{Var}\, C_i, \tag{5.3}$$

où μ_i est la moyenne des points en Ci.

Nous avons donc appliqué l'algorithme de regroupement k-means aux vecteurs S_{cible}. Par la suite, nous avons proposé une deuxième méthode pour calculer la distance entre les vecteurs S_{test} et les vecteurs S_{cible}, comme suit,

$$d(S_{test}, S_{cible}) = \frac{1}{K} \sum_{k=1}^{K} d(S_{test}, C^{(k)}), \tag{5.4}$$

où K désigne le nombre total de clusters S_{cible}, et C désigne un cluster. En effet, les vecteurs de S_{test} sont comparés à chaque cluster $C^{(k)}$ de S_{cible},

$$d(S_{test}, C^{(k)}) = \frac{1}{N} \sum_{i=1}^{N} d(S_{test}^{(i)}, C^{(k)}), \tag{5.5}$$

où la distance entre le $i^{ème}$ vecteur de S_{test} et le $k^{ème}$ cluster de S_{cible} est définie comme suit,

$$d(S_{test}^{(i)}, C^{(k)}) = \min_{j=1,\ldots,L} d(S_{test}^{(i)}, C_{(j)}^{(k)}), \tag{5.6}$$

où L est le nombre total de vecteurs dans le cluster $C^{(k)}$.

Dans Eq. 5.6 nous prenons le vecteur le plus proche dans le cluster $C^{(k)}$ de $S_{test}^{(i)}$, puis nous calculons la distance entre les vecteurs S_{test} et le $k^{ème}$ cluster comme mentionné dans Eq. 5.5. Enfin, dans Eq. 5.4, nous calculons la distance moyenne de toutes les distances résultantes de Eq. 5.5. Nous notons que la méthode (i) est un cas particulier de la méthode (ii) lorsque $k = 1$.

5.2.3 Méthode (iii): le plus proche cluster

Le signal de parole du locuteur peut comprendre non seulement les caractéristiques spécifiques du locuteur, mais aussi les caractéristiques communes entre locuteurs. C'est pourquoi nous avons proposé une troisième façon de calculer la distance entre les vecteurs S_{test} et les vecteurs S_{cible}, qui consiste à renvoyer la distance entre les vecteurs S_{test} et le cluster le plus proche $C^{(k)}$. En effet, nous remplaçons Eq. 5.4 par l'équation suivante,

$$d(S_{test}, S_{cible}) = \min_{k=1,\ldots,K} d(S_{test}, C^{(k)}) \tag{5.7}$$

Le résumé de la méthode proposée est donné dans l'algorithme 3.

Algorithm 3 Le plus proche cluster: *The Nearest Cluster (NearC)*

Input: les vecteurs des matrices S_{cible} et S_{test}; le nombre de clusters k.
Output: la distance *dist* entre S_{cible} et S_{test}.
procedure NEARC(S_{test}, S_{cible}, k)
    $dist = infinity$
    $clusters = kmeans(S_{cible}, k)$ ▷ $kmeans()$ renvoie une liste de clusters
    for each *cluster* in *clusters* **do**
        if $dist > distanceTestCluster(S_{test}, cluster)$ **then**
            $dist = distanceTestCluster(S_{test}, cluster)$
        end if
    end for
    return *dist*
end procedure
procedure DISTANCETESTCLUSTER(S_{test}, *cluster*)
    $sum = 0$
    for each v in S_{test} **do**
        $sum = sum + distanceVectorCluster(v, cluster)$
    end for
    return *sum / cluster.length*
end procedure
procedure DISTANCEVECTORCLUSTER(v, *cluster*)
    $d = infinity$
    for each u in *cluster* **do**
        if $d > distance(v, u)$ **then**
            $d = distance(v, u)$ ▷ $distance()$ renvoie la distance entre deux vecteurs
        end if
    end for
    return d
end procedure

5.3 Résultats expérimentaux

5.3.1 Environnement expérimental

L'approche proposée a été évaluée sur (i) PC, Processeur: Intel Core^{TM} i7-7700HQ 2.80 GHz 2.81 GHz ; RAM installée: 16.0GB, (ii) OS: Microsoft Windows 10 Professionnel, (iii) Langage de programmation: Python 3.6.

5.3.2 Protocole expérimental

L'approche proposée a été évaluée sur le corpus THUYG. Les 39 vecteurs MFCC sont calculés (13 vecteurs MFCC + Δ + $\Delta\Delta$). Nous avons conservé le même protocole expérimental expliqué dans le chapitre 4. Par conséquent, le signal de parole de chaque locuteur dans l'ensemble d'entraînement est converti en sa matrice MFCC correspondante $S_{cible}(39 \times 3000)$. Il en va de même pour l'ensemble de test, chaque locuteur dispose d'un certain nombre de segments de test $S_{test}(39 \times 1000)$. Pour chaque segment de test, nous le comparons à l'ensemble des segments de locuteurs cibles, ce qui donne un total de 119,277 essais pour les locuteurs féminins et 63,361 essais pour les locuteurs masculins.

5.3.3 Résultats et discussion

La première expérience a consisté à choisir la meilleure distance en termes de taux d'erreur (EER). Nous calculons la distance en utilisant la méthode (i), avec $distance = \{Euclidien, Manhattan, Cosinus\}$, voir Eq. 5.8, et en utilisant les signaux de la parole nette.

$$\begin{cases} Euclidienne & d(x,y) = \sqrt{\sum_{i=0}^{n-1}(x_i - y_i)^2} \\ Manhattan & d(x,y) = \sum_{i=0}^{n-1}|x_i - y_i| \\ Cosinus & d(x,y) = 1 - similarity(x,y) \\ & similarity(x,y) = \frac{\sum_{i=0}^{n-1} x_i y_i}{\sqrt{\sum_{i=0}^{n-1}(x_i)^2} * \sqrt{\sum_{i=0}^{n-1}(y_i)^2}} \end{cases} \tag{5.8}$$

Tableau 5.1: Présentation des résultats (en % EER) de la comparaison des distances à l'aide de la méthode (i) sur le corpus THUYG / Locuteurs féminins, dans les conditions nettes.

Distance	Méthode (i)
Euclidien	8.20
Manhattan	8.91
Cosinus	5.01

Le tableau 5.1 montre que la distance en cosinus est plus performante que les distances euclidienne et de Manhattan. Ces résultats nous ont encouragés à considérer le cosinus comme la principale distance dans notre travail. Par la suite, nous avons testé la méthode (ii), avec $k > 1$, voir le tableau 5.2.

Tableau 5.2: Présentation des résultats (en % EER) de la méthode (ii) en utilisant le corpus THUYG / Locuteurs féminins, avec $k \in [2, 7]$, dans les conditions nettes.

Nombre de clusters	Méthode (ii)
k = 2	5.01
k = 3	4.39
k = 4	4.10
k = 5	4.42
k = 6	4.91
k = 7	5.04

Le taux d'erreur passe de 5,01% EER à 4,10% EER lorsque les vecteurs S_{cible} sont regroupés en quatre clusters. Cependant, lorsque le nombre de clusters augmente, l'impact des clusters les plus proches peut être négligé. C'est pourquoi nous proposons dans la méthode (iii) de ne renvoyer que la distance entre les vecteurs S_{test} et le cluster de S_{cible} le plus proche à S_{test}.

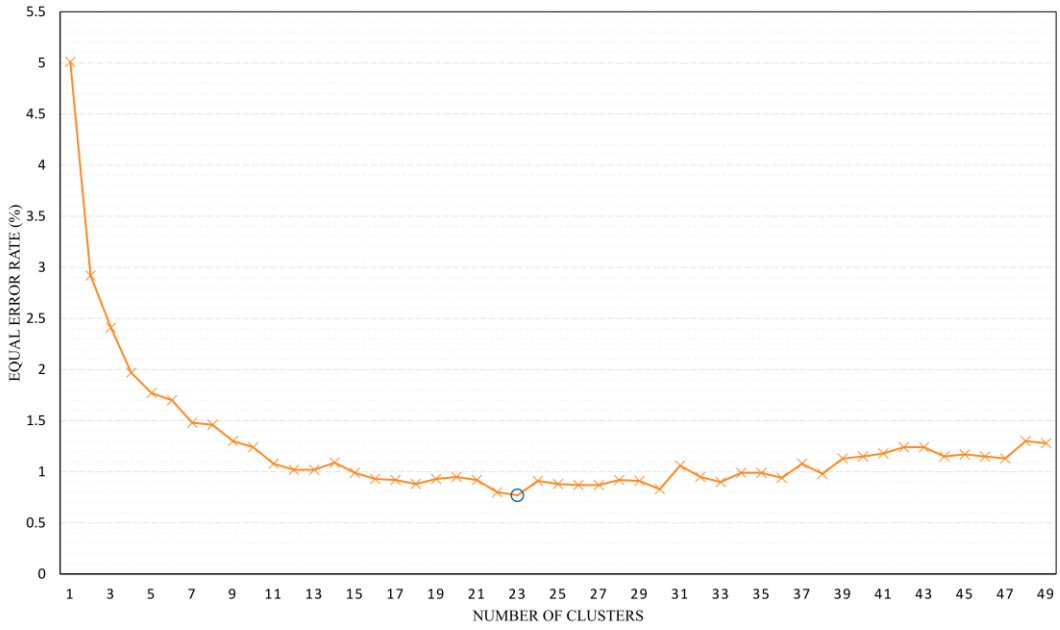


Figure 5.1: Méthode (iii): MFCC/NearC, Le taux d'erreur égal est tracé par rapport au nombre de clusters sur le corpus THUYG / locuteurs féminins, le EER minimal est de 0,77% pour $k = 23$ (point entouré d'un cercle).

La figure 5.1 montre que les résultats ont été largement améliorés par le regroupement des vecteurs S_{cible} en deux clusters. En effet, le taux d'erreur a

été réduit de 5,01% EER à 2,92% EER par rapport à la méthode (i). En outre, le taux d'erreur minimal était de 0,77% EER, avec $k = 23$.

Les résultats recueillis sont impressionnants lorsque l'on dispose de 10 secondes pour chaque segment de parole testé. Cela est dû parce qu'un segment de parole de 10 secondes contient la majorité des caractéristiques du locuteur. Par conséquent, une procédure de vérification du locuteur sera difficile avec des segments de test de courte durée. Dans ce contexte, nous avons divisé les segments de test en segments de courte durée $(0-1sec., 0-2sec., \ldots, 0-9sec.)$. Les résultats sont présentés dans le tableau 5.3.

Tableau 5.3: Présentation des résultats (en % EER) des méthode: GMM/UBM, *i*-vector/PLDA, et la méthode proposée MFCC/NearC en utilisant le corpus THUYG avec $k = 23$ et en utilisant différentes durées de segments de test.

Durée (seconds)	1	2	3	4	5	6	7	8	9	10
GMM/UBM	14.28	12.54	12.12	9.49	8.15	7.98	5.62	4.32	4.31	4.25
i-vector/PLDA	7.72	5.23	4.77	4.32	4.02	3.15	2.25	2.28	2.26	2.26
MFCC/NearC	1.16	1.20	1.21	0.99	1.02	0.98	0.99	1.03	0.98	0.88

Les résultats présentés dans le tableau 5.3 ont démontré l'efficacité de la méthode proposée. En effet, pour un segment de test d'une seconde seulement, notre approche est nettement plus performante que l'approche *i*-vector/PLDA (en utilisant des segments de test de 10 secondes) avec une diminution significative du taux d'erreur à 1,16% EER. L'idée de regrouper les vecteurs S_{cible} est intéressante, elle permet de montrer que certains clusters contiennent les caractéristiques spécifiques du locuteur. Ce taux d'erreur faible obtenu en utilisant, seulement, une seconde nous conforte dans notre démarche de raisonnement.

La deuxième expérience visait à comparer les résultats de la méthode MFCC/NearC avec le système de référence (GMM/UBM et *i*-vector/PLDA, voir, chapitre 4 tableau 4.4). Pour cette raison, nous avons appliqué le même protocole expérimental que dans le chapitre 4. Les résultats sont présentés dans le tableau 5.4.

Dans le tableau 5.4, nous constatons que les résultats obtenus dans le corpus des locuteurs féminins sont systématiquement corrélés avec les résultats obtenus dans le corpus des locuteurs masculins. C'est pourquoi nous avons transformé les deux tableaux de résultats en un seul, tableau 5.5. Ces résultats montrent que la méthode proposée est plus performante que les méthodes traditionnelles lorsque les énoncés de l'entraînement et de test sont dans des conditions nette. En outre, les résultats ne sont pas éloignés de ceux donnés par les méthodes traditionnelles lorsque les énoncés d'entraînements et de tests sont prononcés dans le même niveau

SNR. Sans cela, les résultats ne sont pas aussi bons que ceux donnés par les méthodes traditionnelles dans les autres cas.

En comparant nos résultats à ceux des méthodes traditionnelles, il faut souligner que NearC est considérée comme une méthode de modélisation des modèles *Templates*, que nous comparons à des modèles stochastiques. Il est bien connu que les modèles stochastiques sont plus performants que les modèles *Templates*, notamment en reconnaissance automatique du locuteur, en raison de leur capacité à traiter des données bruitées. Cependant, notre méthode a l'avantage de traiter la parole propre avec moins de données.

Tableau 5.4: Présentation des résultats (en % EER) des méthodes suivantes : GMM/UBM et *i*-vector/PLDA, et MFCC/NearC, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db. Le module UBM est entraîné en utilisant le corpus Fisher.

Locuteurs féminins		Méthodes		
Cible (SNR)	Test (SNR)	GMM/UBM	*i*-vector/PLDA	MFCC/NearC
Nette	Nette	4.32	2.33	0.77
Nette	9db	12.84	4.45	10.63
Nette	0db	21.62	8.83	15.28
9db	Nette	8.95	4.01	9.42
9db	9db	6.17	3.72	5.01
9db	0db	8.14	5.76	13.55
0db	Nette	12.54	6.57	17.74
0db	9db	8.52	4.74	15.02
0db	0db	9.99	5.47	9.85
AVG EER		10.34	5.10	10.81

Locuteurs masculins		Méthodes		
Cible (SNR)	Test (SNR)	GMM/UBM	*i*-vector/PLDA	MFCC/NearC
Nette	Nette	4.16	2.16	1.02
Nette	9db	12.60	4.12	10.80
Nette	0db	21.02	7.77	16.11
9db	Nette	8.27	3.91	9.63
9db	9db	5.48	3.66	5.22
9db	0db	7.83	3.34	13.87
0db	Nette	11.90	6.01	17.93
0db	9db	8.00	4.80	15.45
0db	0db	9.35	4.99	10.07
AVG EER		9.85	4.52	11.12

D'une manière générale, l'approche *i*-vector/PLDA bénéficie du vaste ensemble de données utilisé pour entraîner l'UBM, ce qui se traduit clairement par une erreur moyenne de 4.95% EER. Bien que la méthode que nous proposons, MFCC/NearC, ne nécessite pas de données UBM, l'erreur moyen enregistré (10.94% EER) n'est pas éloigné de l'erreur moyen enregistré par GMM/UBM (10.13% EER).

Tableau 5.5: Présentation des résultats (en % EER) des méthodes suivantes : GMM/UBM et *i*-vector/PLDA, et MFCC/NearC, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db. Le module UBM est entraîné en utilisant le corpus Fisher. (Le taux d'erreur qui prend en considération à la fois les locuteurs masculins et féminins est calculé selon l'équation suivante : $EER = \frac{(N_m \times EER_m)+(N_f \times EER_f)}{N_m+N_f}$, avec N, m et f font référence respectivement aux nombre total de locuteurs, masculin, et féminin.)

Locuteurs		Méthodes		
Cible (SNR)	Test (SNR)	GMM/UBM	*i*-vector/PLDA	MFCC/NearC
Nette	Nette	4.25	2.26	0.88
Nette	9db	12.74	4.31	10.70
Nette	0db	21.36	8.37	15.64
9db	Nette	8.66	3.97	9.51
9db	9db	5.87	3.69	5.10
9db	0db	8.01	5.58	13.69
0db	Nette	12.26	6.33	17.82
0db	9db	8.30	4.77	15.21
0db	0db	9.71	5.26	9.94
AVG EER		10.13	4.95	10.94

5.4 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode d'évaluation des locuteurs pour la tâche de vérification automatique du locuteur. Nous avons démontré que des résultats intéressants peuvent être obtenus en utilisant, directement, les vecteurs caractéristiques du locuteur (MFCC). Ainsi, notre méthode est considéré comme une méthode des modèles *Templates*, donc nous n'avons pas de phase d'entraînement du modèle UBM, et la décision est basée uniquement sur le calcul de distance. Les expériences ont été menées sur le corpus THUYG. Les résultats obtenus ont démontrés que la méthode proposé "MFCC/NearC" surpasse les méthodes traditionnelles GMM/UBM et *i*-vector/PLDA dans les conditions où les paroles d'entraînements et de tests sont nettes. De plus, notre méthode reste

plus performante en utilisant peu de données en test dans les conditions (Nette - Nette). Cependant, notre méthode n'est pas aussi performante que les méthodes traditionnelles dans des conditions bruyantes. Pour cela, nous proposons, dans le chapitre suivant, une nouvelle approche de deep learning pour surmonté ce problème.

6

Caractéristiques Profondes du Locuteur — DeepSF/NearC

Sommaire

6.1 Introduction

Dans le chapitre précédent, nous avons présenté une nouvelle méthode d'évaluation pour la vérification automatique du locuteur appelée "NearC". La méthode NearC a l'avantage sur les méthodes traditionnelles dans des conditions de parole nettes. Cependant, son problème majeur réside dans son incapacité à traiter des données bruyantes. Pour cette raison, nous avons cherché à améliorer la méthode NearC en utilisant des techniques de deep learning.

Le deep learning a suscité beaucoup plus d'intérêt chez les chercheurs en traitement de la parole, et il a été introduit récemment dans le domaine de la RAL. Dans la plupart des cas, les modèles de deep learning sont adaptés des applications de reconnaissance automatique de la parole et appliqués à la RAL, et ils ont montré leur capacité à concurrencer les modèles des approches traditionnelles. Néanmoins, l'utilisation de deep learning dans la RAL est toujours liée à la reconnaissance de la parole.

Dans cette étude, nous proposons une nouvelle façon d'utiliser les réseaux de neurones profonds (DNN) dans la RAL, dans le but de faciliter à ce réseau de neurones d'apprendre la distribution des caractéristiques. Pour ce faire, nous transformons les vecteurs caractéristiques (MFCC) en vecteurs améliorées que nous dénommons DeepSF (*Deep Speaker Features*), ou caractéristiques profondes du locuteur [3]. Des expériences ont été menées sur le corpus THUYG, et les résultats obtenus sont très intéressants qui surpassent à la fois la méthode *i*-vector/PLDA et notre système de base MFCC/NearC dans des conditions nettes et de bruit.

Les sections suivantes sont organisées comme suit : dans la section 6.2, nous présentons les travaux liés à notre étude. Dans la section 6.3, nous présentons la méthode proposée. Dans la section 6.4, nous présentons les résultats et nous les discutons, et dans la section 6.5, nous concluons le chapitre.

6.2 Travaux connexes

Le succès des techniques du deep learning dans la reconnaissance automatique de la parole [97–99] et dans le traitement d'images [100, 101], a incité les chercheurs à utiliser ses techniques dans la RAL. C'est pourquoi plusieurs approches de deep learning ont été étudiées. D'une part, les machines Boltzmann restreintes (RBM) et les réseaux de croyances profondes (DBN) ont été utilisés comme approches génératives. D'autre part, les réseaux de neurones profonds (DNN) ont été utilisés comme approches discriminantes.

Un réseau RBM est un réseau de neurones bidirectionnel, dans lequel il n'y a pas de connexion entre les unités d'une même couche. Il a été proposé en premier lieu pour la RAL par [102]. Le DBN est un réseau de neurones profond non supervisé qui consiste en modèles RBM empilés, il a été introduit par [51], et utilisé pour la vérification automatique du locuteur afin d'extraire les *i*-vectors [103], et aussi comme une phase de pré-entraînement pour les DNN [104].

Un DNN est un réseau de neurones artificiel supervisé avec plusieurs couches cachées, et une couche de sortie *softmax*[1]. Il a été utilisé dans la RAL dans le but de discriminer entre les locuteurs ou de générer des *i*-vectors. Dans le premier cas, la couche de sortie se compose des étiquettes des locuteurs [104]. Dans le second cas, la couche de sortie est constituée de phonèmes de la parole. En effet, un DNN entraîné pour la modélisation acoustique en reconnaissance automatique de la parole [105–108] remplace le modèle UBM. Ensuite, les caractéristiques de chaque locuteur sont transmises au réseau DNN afin d'extraire les caractéristiques dites *Bottleneck* (BN), qui peuvent être utilisées de deux manières : comme vecteurs caractéristiques, ou en les concaténant avec les MFCC [109].

6.3 Méthodologie

La figure 6.1 représente le schéma fonctionnel de la méthode proposée. Dans cette section, nous expliquons le processus de transformation des vecteurs MFCC en vecteurs caractéristiques profond du locuteur DeepSF. Pour ce faire, nous utilisons :

- K-Means : pour regrouper les vecteurs MFCC d'un locuteur en deux clusters ($K = 2$) ;
- Réseau de croyance profond (DBN) : pour initialiser les poids du réseau de neurones profond (DNN) ;
- Réseau de neurones profond (DNN) : pour classifier les vecteurs MFCC d'un locuteur en deux classes, et extraire les DeepSF ;
- Le plus proche cluster (NearC) : pour évaluer les locuteurs pour la tâche de vérification automatique du locuteur.

6.3.1 Extraction et regroupement de caractéristiques

Le signal vocal est divisé en petites trames de 25 ms avec un glissement de 10 ms. Ensuite, chaque trame est représentée par un vecteur caractéristique MFCC, qui comprend des MFCC d'ordre 13 et leurs premières et secondes dérivées.

Afin de permettre au DNN d'apprendre la distribution des caractéristiques, nous devrions séparer les fonctionnalités similaires en clusters. Cependant, le DNN a

[1]La fonction softmax est une fonction qui transforme un vecteur de K valeurs réelles en un vecteur de K valeurs réelles dont la somme est égale à 1. Les valeurs d'entrée peuvent être positives, négatives, nulles ou supérieures à un, mais la softmax les transforme en valeurs comprises entre 0 et 1, de sorte qu'elles peuvent être interprétées comme des probabilités.

besoin de beaucoup plus de données pour éviter le sur-apprentissage. Pour cette raison, nous avons décidé de regrouper les caractéristiques du locuteur en deux groupes seulement. Par conséquent, nous avons choisi d'appliquer le méthode K-Means aux vecteurs MFCC du locuteur avec $K = 2$.

6.3.2 Entraînement du réseau de neurones profond (DNN)

L'entraînement du réseau de neurones profond (DNN) se déroule en deux phase successives. Tout d'abord, une phase non supervisé est effectuée, au cours de laquelle un DBN est entraîné afin d'utiliser ses poids résultants comme poids initiaux du DNN. Deuxièmement, dans la phase discriminante, le DNN est entraîné de manière supervisée afin d'étiqueter les vecteurs MFCC avec l'étiquette du cluster correspondant.

La RBM représente l'élément de base du DBN, elle comprend deux couches, la couche visible (d'entrée) $v = \{v_1, v_2, ...v_V\}$, et la couche cachée (sortie) $h = \{h_1, h_2, ...h_H\}$, chaque couche est constituée d'unités, dans lesquels chaque unité est connectée à toutes les unités de l'autre couche, et aucune connexion entre les unités de la même couche. Dans ce chapitre, nous utilisons la modèle Gaussien-Bernoulli RBM (GB-RBM), avec $v \in \mathbb{R}$ et $h \in \{0, 1\}$, pour construire les blocs du DBN. Par conséquent, la fonction énergétique de GB-RBM est définie par,

$$E(h, v) = \sum_{i=1}^{V} \frac{(v_i - b_i^v)^2}{2\sigma_i^2} - \sum_{j=1}^{H} b_j^h h_j - \sum_{i=1}^{V}\sum_{j=1}^{H} \frac{v_i}{\sigma_i} h_j w_{ij}, \tag{6.1}$$

où v_i désigne les valeurs d'entrée dans la couche i et h_i désigne les valeurs de sortie dans la couche j. w_{ij} représente les poids entre l'unité visible v_i et l'unité cachée h_j, b_i^v et b_j^h sont les biais des unités v_i et h_j, respectivement. σ_i est la notation de l'écart-type du bruit gaussien pour v_i. Par conséquent, la distribution de probabilité commune associée à la configuration de (v, h) est définie par,

$$p(v, h; \Theta) = \frac{exp(-E(v, h; \Theta))}{Z}, \tag{6.2}$$

Θ désigne les poids et les biais, avec Z est la fonction de partition qui est définie par,

$$Z = \sum_{v}\sum_{h} exp(-E(v, h; \Theta)). \tag{6.3}$$

Le DBN est une concaténation de plusieurs RBM. Une RBM est entraînée de manière non supervisée et utilisée comme extracteur de caractéristiques. Ce modèle a la capacité de traiter des structures de données compliquées et de détecter des caractéristiques qui ne peuvent pas être détectées directement [64]. Par

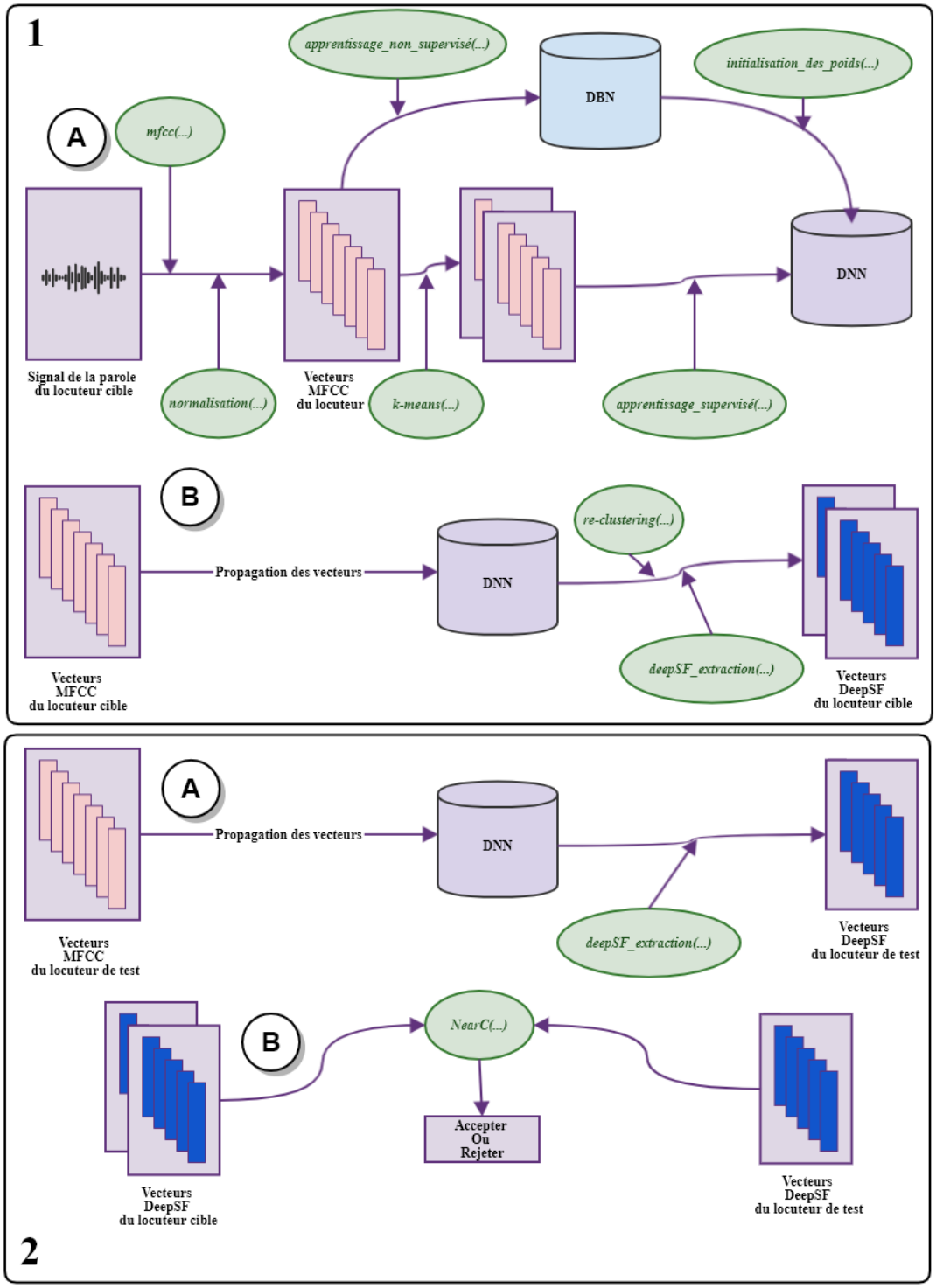


Figure 6.1: Le schéma de la méthode proposée. La phase (1) est la phase d'entraînement, la phase (2) est la phase de test.

conséquent, l'empilement de RBM de manière successive peut représenter des structures plus complexes. En effet, la sortie de la RBM précédente est utilisée pour entraîner la RBM courante afin de détecter des caractéristiques implicites de la RBM précédente, etc. En général, le DBN est entraîné en utilisant l'algorithme couche par couche [51] (voir chapitre 3, Algorithme 2), son avantage est de découvrir des caractéristiques descriptives qui illustrent la corrélation entre les entrées dans chaque couche [110]. L'algorithme d'apprentissage couche par couche permet une meilleure optimisation des poids entre les couches. En outre, l'initialisation des poids du DBN pourrait améliorer les résultats plutôt que si des poids aléatoires sont utilisés. En outre, les avantages de l'apprentissage du DBN sont étendus à sa capacité à diminuer les effets de sur-apprentissage et de sous-apprentissage, où les deux sont des problèmes communs dans les grandes architectures de deep learning. Pour ces raisons, et dans ce chapitre, le DBN est choisi comme initialisateur de poids pour le DNN. En effet, et après l'apprentissage non supervisée du DBN, une couche softmax qui représente les étiquettes des clusters est ajoutée à la sortie du réseau. Le DNN est entraîné de manière supervisée dans le but d'étiqueter chaque vecteur de caractéristique (par le label de l'un des deux clusters), ce qui permet au réseau, après la fin de l'apprentissage, d'apprendre comment les caractéristiques du locuteur sont distribuées.

6.3.3 Extraction de caractéristiques profondes (DeepSF)

Les vecteurs caractéristiques MFCC sont transformés en vecteurs caractéristiques profonds DeepSF à travers les étapes illustré sur la figure 6.1. Tout d'abord, le DBN est entraîné avec les vecteurs MFCC du locuteur cible d'une manière non supervisé. Ensuite, les poids du DNN sont initialisés avec les poids résultants du DBN. En outre, la méthode de regroupement K-Means est appliquée aux vecteurs MFCC du locuteur cible (Figure 6.1: 1-A). Ensuite, une couche softmax est ajoutée au DNN, constituée des étiquettes des clusters, voir la figure 6.2. Par conséquent, le DNN est entraîné de manière supervisée pour apprendre à distinguer entre les clusters résultants de K-Means. À la fin de la phase d'apprentissage, et après un certain nombre d'époques, le DNN peut ne pas étiqueter l'ensemble des vecteurs MFCC, ce qui signifie que la précision d'apprentissage peut ne pas atteindre 100%. Dans ce cas, nous supposons que le regroupement en utilisant K-Means n'a pas regroupé correctement certains vecteurs MFCC. Par conséquent, et en utilisons le même DNN entraîné (Figure 6.1: 1-B), nous procédons à une phase de reclassification, afin de classifier correctement les vecteurs MFCC dans leurs clusters. En conséquence, les vecteurs MFCC du locuteur cible sont à nouveau propagés vers l'avant, uniquement,

dans le DNN, afin qu'il puisse les étiqueter dans leurs clusters. Par la suite, nous retirons la couche softmax du DNN. De cette façon le DNN sera considéré comme un modèle représentant le locuteur cible. Ensuite, nous propageons chaque vecteur MFCC du locuteur cible dans le DNN, et nous renvoyons les valeurs de la dernière couche représentant un vecteur DeepSF.

Pour vérifier un locuteur inconnu, nous utilisons le modèle DNN du locuteur proclamé. Par conséquent, nous générons les vecteurs DeepSF du locuteur test (Figure 6.1: 2-A). Enfin, nous procédons à la méthode d'évaluation NearC qui vérifie le degré de similarité entre les vecteurs DeepSF du locuteur cible et les vecteurs DeepSF locuteur test (Figure 6.1: 2-B).

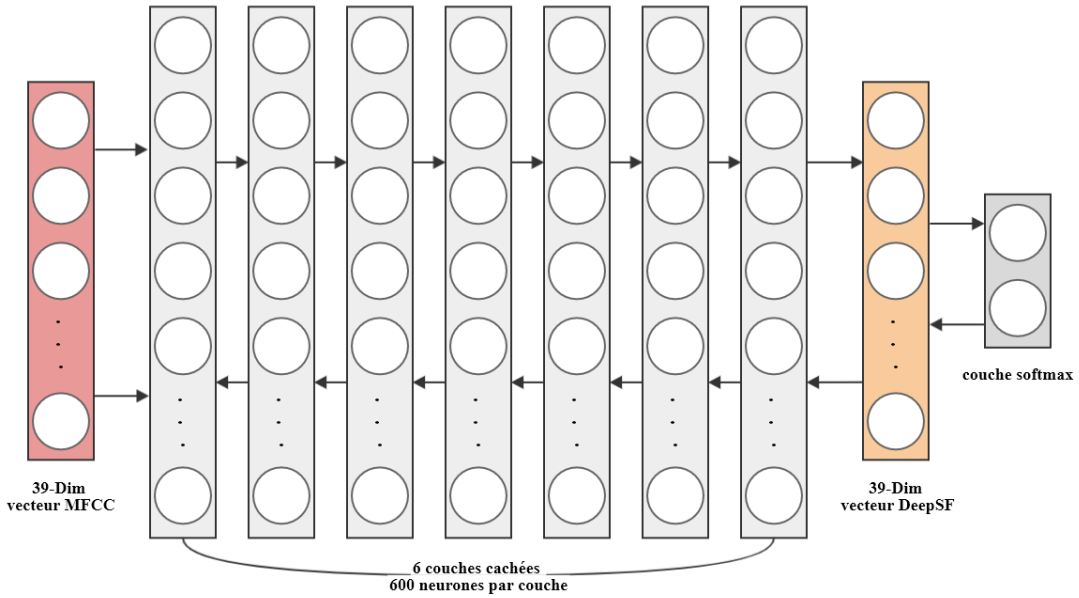


Figure 6.2: L'architecture du réseau de neurone profond (DNN) utilisé dans la méthode proposée.

6.4 Résultats expérimentaux

6.4.1 Environnement expérimental

La méthode proposée a été évaluée sur ces configurations: (i) PC, CPU: Intel Core^{TM} i7-7700HQ 2,80 Ghz; GPU: Nvidia GeForce GTX 1050 4 Go V-RAM, cœurs CUDA: 640, Bande passante mémoire: 112,13 Go/s; Mémoire cache 6 Mo, RAM: 16,0 Go. (ii) Système d'exploitation: Microsoft Windows 10 Professionnel, (iii) Langage de programmation: Python 3.6; Frameworks: TensorFlow (pour RBM et DBN) et Keras (pour DNN).

6.4.2 Protocole expérimental

Dans ce travail, nous utilisons des architectures de deep learning de grandes tailles, ce qui est coûteux en termes de calcul. c'est pourquoi l'entraînement a été accéléré à l'aide d'un processeur graphique. Les MFCC à 39 dimensions sont utilisés, y compris le vecteur MFCC statique à 13 dimensions, dans lequel le coefficient C0 est remplacé par l'énergie, concaténée aux vecteurs Δ et $\Delta\Delta$. Par la suite, une normalisation CMVN a été appliquée afin de supprimer l'effet de canal.

Après de nombreuses expériences, nous avons choisi d'utiliser l'architecture DNN suivante : entrée (39 unités) ; couches cachées (6 couches cachées, 600 unités par couche) + (1 couche Bottleneck de 39 unités) ; sortie (couche sortie softmax, 2 unités) 6.2. La tailles des couches d'entrée et de Bottleneck correspond à la taille du vecteur MFCC, et la taille de la couche de sortie correspond au nombre de clusters K. Afin d'initialiser les poids du DNN, les vecteurs MFCC du locuteur cible sont envoyés au premier réseau RBM du DBN, ils sont entraînés en 10 époques et les sorties de cette RBM sont envoyées à la RBM suivante qui est également entraînée en 10 époques, etc. Enfin, nous utilisons les poids du DBN pour initialiser les poids du DNN.

Les vecteurs MFCC du locuteur cible sont regroupés selon la méthode k-Means. Étant donné que les vecteurs MFCC du locuteur comprennent un nombre inconnu de caractéristiques, ce qui est difficile à définir, et en raison de la contrainte de données insuffisantes, nous avons décidé de limiter le nombre de clusters à deux ($k = 2$), où nous supposons que chaque cluster comprend un certain nombre de caractéristiques similaires. Par la suite, nous utilisons le DNN comme classifieur de vecteurs MFCC, où chaque vecteur prend le label de l'un des deux clusters. La fonction d'activation utilisée dans cette architecture est la fonction ReLU pour les couches cachées et la fonction Sigmoïde pour la couche softmax avec un dropout de $0,5$.

Pour compiler le modèle, nous avons utilisé l'optimiseur stochastique de descente de gradient (SGD) avec ces hyper-paramètres: $0,005$ pour le taux d'apprentissage, $1e-6$ pour la décroissance du taux d'apprentissage, et $0,9$ pour le moment. Le modèle est entraîné en 100 époques, et le modèle avec la meilleur précision d'apprentissage est retourné.

6.4.3 Résultats et discussion

Le tableau 6.1 représente les résultats, sur le corpus THUYG, en utilisant trois méthodes: (i) i-vector/PLDA, (ii) MFCC/NearC et (iii) DeepSF/NearC. Les expériences ont été menées sous trois SNR de bruit : Nette, 9db et 0db. Il a été

Tableau 6.1: Présentation des résultats (en % EER) des méthodes suivantes : *i*-vector/PLDA, la méthode de référence MFCC/NearC, et la méthode proposée DeepSF/NearC, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db.

Locuteurs féminins		Méthodes		
Cible (SNR)	Test (SNR)	*i*-vector/PLDA	MFCC/NearC	DeepSF/NearC
Nette	Nette	2.33	0.77	0.43
Nette	9db	4.45	10.63	0.88
Nette	0db	8.83	15.28	1.15
9db	Nette	4.01	9.42	0.91
9db	9db	3.72	5.01	1.41
9db	0db	5.76	13.55	2.87
0db	Nette	6.57	17.74	3.07
0db	9db	4.74	15.02	2.76
0db	0db	5.47	9.85	2.27
AVG EER		5.10	10.81	1.75

Locuteurs masculins		Méthodes		
Cible (SNR)	Test (SNR)	*i*-vector/PLDA	MFCC/NearC	DeepSF/NearC
Nette	Nette	2.16	1.02	0.55
Nette	9db	4.12	10.80	1.14
Nette	0db	7.77	16.11	1.20
9db	Nette	3.91	9.63	0.88
9db	9db	3.66	5.22	1.51
9db	0db	5.34	13.87	2.43
0db	Nette	6.01	17.93	3.19
0db	9db	4.80	15.45	3.09
0db	0db	4.99	10.07	2.40
AVG EER		4.52	11.12	1.82

prouvé dans le chapitre 5 que la méthode MFCC/NearC a largement surpassé l'*i*-vector/PLDA dans les conditions de parole nettes. En effet, elle a enregistrée 0,88% EER, tandis que l'*i*-vector/PLDA a enregistrée 2,26% EER. Néanmoins, la méthode MFCC/NearC ne fonctionne pas bien dans des conditions de bruit, et cela est dû au fait que la méthode MFCC/NearC essaie de cibler les caractéristiques spécifiques du locuteur, qui sont corrompues par le bruit. Il est clair que lorsque les données d'entraînement et de test passent par le même SNR, l'erreur est bien plus faible que si elles sont dans des SNR différents.

Tableau 6.2: Présentation des résultats (en % EER) des méthodes suivantes : *i*-vector/PLDA, la méthode de référence MFCC/NearC, et la méthode proposée DeepSF/NearC, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db. (Le taux d'erreur qui prend en considération à la fois les locuteurs masculins et féminins est calculé selon l'équation suivante : $EER = \frac{(N_m \times EER_m)+(N_f \times EER_f)}{N_m+N_f}$, avec N, m et f font référence respectivement aux nombre total de locuteurs, masculin, et féminin.)

Locuteurs		Méthodes		
Cible (SNR)	Test (SNR)	*i*-vector/PLDA	MFCC/NearC	DeepSF/NearC
Nette	Nette	2.26	0.88	0.48
Nette	9db	4.31	10.70	0.99
Nette	0db	8.37	15.64	1.17
9db	Nette	3.97	9.51	0.90
9db	9db	3.69	5.10	1.45
9db	0db	5.58	13.69	2.68
0db	Nette	6.33	17.82	3.12
0db	9db	4.77	15.21	2.90
0db	0db	5.26	9.94	2.33
AVG EER		4.95	10.94	1.78

Comme dans le chapitre précédent, nous continuons à constater que les résultats obtenus dans le corpus des locuteurs féminins sont systématiquement corrélés avec les résultats obtenus dans le corpus des locuteurs masculins. C'est pourquoi nous avons transformé les deux tableaux de résultats en un seul, tableau 6.2. Ce tableau montre que la méthode proposée, DeepSF/NearC, a surpassée les méthodes MFCC/NearC et *i*-vector/PLDA. En effet, le méthode DeepSF/NearC a enregistrée un taux d'erreur de 0,48% EER dans les conditions nettes. Ce taux d'erreur est considéré comme le taux le plus faible enregistré dans toutes les expériences (conditions nettes et bruitées). En outre, il semble que la méthode DeepSF/NearC ne soit pas autant affectée par le bruit par rapport aux méthodes MFCC/NearC et *i*-vector/PLDA. En effet, lorsque le DNN est entraîné par des segments de paroles nettes, DeepSF/NearC a enregistrée 1,17% EER. Ce taux d'erreur est considéré comme le taux le plus élevé enregistré par notre méthode dans des conditions de tests bruitées, et des conditions de paroles nettes. Au contraire, les taux d'erreur les plus faibles pour les *i*-vectors/PLDA et MFCC/NearC, en même conditions, sont 4,31% EER et 10,70% EER respectivement. Par ailleurs, Les résultats montrent que la méthode MFCC/NearC a enregistrée des taux d'erreur inférieurs lorsque les données d'entraînement et de test sont dans le même niveau SNR (Nette-Nette:

0,88% EER, 9db-9db: 5,10% EER et 0db-0db: 9,94% EER). Cela signifie que la méthode MFCC/NearC est pertinente lorsque les données d'entraînement et de test sont dans les mêmes conditions de bruit et dans les mêmes SNR. De plus, le taux d'erreur change en corrélation avec le degré de SNR ; lorsque le signal de la parole est plus bruité, le taux d'erreur est plus élevé et vice versa, d'où la limitation de la méthode MFCC/NearC. Alors que, dans la méthode DeepSF/NearC, le taux d'erreur varie entre 0,48% EER et 3,12% EER. D'une manière générale, la méthode proposée a permis de réduire largement l'erreur moyenne de 4,95% EER (i-vector/PLDA) à 1,78% EER.

Comme nous l'avons fait dans le le chapitre précédent, nous divisons les segments de test en segments courts $(0-1sec., 0-2sec., \ldots, 0-9sec.)$. Les résultats sont présentés dans le tableau 6.3.

Tableau 6.3: Présentation des résultats (en % EER) des méthode: i-vector/PLDA, MFCC/NearC et la méthode proposée DeepSF/NearC en utilisant le corpus THUYG avec différentes durées de segments de test, et dans des conditions de paroles nettes.

Durée (seconds)	1	2	3	4	5	6	7	8	9	10
i-vector/PLDA	7.72	5.23	4.77	4.32	4.02	3.15	2.25	2.28	2.26	2.26
MFCC/NearC	1.16	1.20	1.21	0.99	1.02	0.98	0.99	1.03	0.98	0.88
DeepSF/NearC	0.82	0.80	0.79	0.58	0.60	0.55	0.56	0.58	0.52	0.48

Nous remarquons qu'avec, seulement, une seconde du segment de test, la méthode DeepSF/NearC a pu surpasser les deux méthodes i-vector/PLDA et MFCC/NearC. En effet, le taux d'erreur d'une second du segment de test pour la méthode proposée est de 0,82% EER, or l'i-vector/PLDA et MFCC/NearC ont enregistrée 7,72% EER et 1,16% EER respectivement. En outre, le taux d'erreur enregistré par DeepSF/NearC avec une seconde du segment de test reste plus faible par rapport aux taux d'erreur des autres méthodes, même si les autres méthodes utilisent 10 secondes du segment de test.

Le réseau de neurones profond (DNN) est connu pour sa capacité à apprendre des caractéristiques complexes et indirectes par le biais de ses couches cachées. En effet, plus on va en profondeur dans les couches, plus le DNN peut représenter des caractéristiques complexes et indirectes. Dans ce chapitre, les résultats conduisent à conclure que le fait de faire apprendre à un DNN la distribution des caractéristiques dans un ensemble de vecteurs MFCC d'un locuteur peut les améliorer en générant des caractéristiques plus puissantes (DeepSF).

6.5 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode pour transformer les vecteurs caractéristiques du locuteur (MFCC) en vecteurs caractéristiques profonds du locuteur (DeepSF), en utilisant des techniques de deep learning. En effet, un DBN est entraîné pour apprendre les MFCC, de manière non supervisée, puis ses poids ont été utilisés pour initialiser le DNN. Par la suite, le DNN est utilisé de manière supervisée pour apprendre la distribution des caractéristiques dans l'ensemble de vecteurs MFCC du locuteur, et pour chaque vecteur MFCC, un vecteur DeepSF est extrait. Pour évaluer le vecteur DeepSF extrait, nous avons utilisé NearC comme méthode d'évaluation. Des expériences ont été menées sur le corpus THUYG pour les ensembles de locuteurs féminins et masculins. Notre approche a démontré ses performances dans toutes les conditions de bruit qui sont utilisées dans ce travail. En effet, dans le cas de la parole nette, le méthode DeepSF/NearC a surpassé la méthode de référence MFCC/NearC avec une diminution de l'erreur de 0,88% EER à 0,48% EER. En outre, le taux d'erreur n'a pas dépassé 3,12% EER dans toutes les conditions de bruit, tandis que le *i*-vector/PLDA et MFCC/NearC ont enregistré 8,37% EER et 17,82% EER respectivement. Il a donc été prouvé que le fait de pousser un DNN à apprendre la distribution des caractéristiques dans l'ensemble des vecteurs caractéristiques du locuteur permet de générer des vecteurs caractéristiques robustes pour la RAL.

Plusieurs applications pourraient découler de ces recherches. En effet, le taux d'erreur le plus faible atteint dans ce chapitre pourrait encourager le secteur bancaire à adopter notre méthode pour déployer ses systèmes biométriques vocaux. En outre, notre méthodologie pourrait être utilisée pour un système de RAL dans les enquêtes criminelles.

7

Intégration des Réseaux de Neurones Convolutifs dans la RAL — CNN/UBM-NearC

Sommaire

7.1 Introduction

Les modèles de deep learning sont désormais considérés comme les approches de pointe dans de nombreux domaines de la reconnaissance des formes. Dans le domaine de la reconnaissance automatique du locuteur, plusieurs architectures ont été étudiées, telles que les réseaux de neurones profonds (DNN), les réseaux de

croyances profondes (DBN), les machines de Boltzmann restreintes (RBM), etc., tandis que les réseaux de neurones convolutifs (CNN) sont les modèles les plus utilisés dans la reconnaissance des images. Le problème est que les CNN sont limités au domaine de l'image en raison de leur structure conçue pour des données bidimensionnelles. Pour surmonter cette limitation, nous visons à représenter les modèles de locuteurs sous forme de matrice. À cette fin, nous utilisons des machines Boltzmann restreintes pour l'extraction des matrices. Ensuite, nous introduisons une nouvelle méthode de modélisation des locuteurs cibles et non-cibles afin d'effectuer une vérification automatique du locuteur à l'aide d'un réseau de neurones convolutif. Ce travail comprend trois contributions principales :

- C'est le premier travail de RAL à utiliser les réseaux CNN sans recourir aux spectrogrammes ;
- La proposition d'une nouvelle méthode pour créer des locuteurs cibles et non-cibles ;
- La proposition d'une nouvelle méthode pour générer des données du locuteur en utilisant l'UBM.

L'architecture de ce chapitre est la suivante : dans la section 7.2, nous présentons les travaux correspondants. Dans la section 7.3, nous présentons la méthode proposée. Dans la section 7.4, nous présentons une discussion et analyse des résultats. Dans la section 7.5, nous concluons le chapitre.

7.2 Travaux connexes

Le réseau de neurones convolutif (CNN) est une architecture de deep learning qui prend une image (ou n'importe quelle forme matricielle) d'entrée, attribue de l'importance aux différents aspects de l'image et est capable de les distinguer les uns des autres. Les CNN ont été utilisées dans la RAL en utilisant des spectrogrammes comme entrée. Dans [111], CNN est utilisé pour la vérification automatique du locuteur afin de capturer les caractéristiques du locuteur et d'éliminer les autres caractéristiques, et dans [112] CNN est utilisé pour l'identification automatique du locuteur, et étendu au problème de classification automatique de locuteurs. La majorité des travaux utilisent des spectrogrammes (images) comme entrée de CNN [113–115], et cela parce que CNN est conçu pour être utilisé en traitement d'images (c'est-à-dire pour des données 2D) [52]. Cependant, nous pensons que l'information représentée sur un spectrogramme est différente de celle des "images naturelles". En effet, un spectrogramme est une illustration visuelle des fréquences spectrales d'un signal qui varient avec le temps.

7.3 Méthodologie

Dans ce travail, nous proposons une nouvelle méthode de vérification automatique du locuteur qui fait appel au deep learning. En partant de la motivation de modéliser les locuteurs avec des matrices représentatives plutôt qu'avec des spectrogrammes, nous utilisons des RBM pour transformer les vecteurs caractéristiques des locuteurs en matrices. Pour la vérification automatique du locuteur, nous avons besoin d'un modèle de locuteur cible, et d'un modèle pour représenter les caractéristiques communes des autres locuteurs. Pour cette raison, nous créons un modèle pour le locuteur cible, et un autre modèle que nous appelons le modèle du locuteur non-cible, comme l'opposé du locuteur cible. En effet, nous générons un ensemble de matrices pour le locuteur cible, et un autre ensemble avec le même nombre de matrices pour le locuteur non-cible. En pratique, chaque locuteur cible a son locuteur non-cible correspondant. Enfin, nous utilisons un CNN pour distinguer les matrices du locuteur cible des matrices du locuteur non-cible.

Dans la suite de cette section, nous présentons en détail la méthodologie de notre travail. Tout d'abord, nous avons utilisé les MFCC pour extraire les vecteurs caractéristiques, puis nous avons utilisé les RBM pour modéliser les locuteurs cibles et non-cibles. Enfin, nous utilisons un CNN pour procéder à la tâche de vérification automatique du locuteur, afin de discriminer entre les locuteurs cibles et non-cibles.

7.3.1 Création de données cibles et non-cibles

Dans cette section, nous expliquons comment nous avons utilisé les RBM pour modéliser les locuteurs cibles et non-cibles, et comment construire les matrices de poids.

7.3.1.1 Création de données UBM

Tout d'abord, le modèle UBM est utilisé pour extraire les matrices de poids. En effet, chaque locuteur dans l'UBM est représenté par une matrice de poids, voir la figure 7.1. En pratique, les poids de la RBM sont initialisés de manière aléatoire par des valeurs comprises entre 0 et 1. Par conséquent, les vecteurs MFCC de chaque locuteur de l'UBM sont transmis à la RBM, qui tente de reconstruire les trames d'entrée. Lorsque la RBM atteint la précision d'apprentissage adéquate, nous renvoyons les valeurs de poids entre la couche visible et la couche cachée, sous forme d'une matrice. Enfin, le nouveau UBM ne comprend que des matrices, au lieu de vecteurs MFCC.

Le nouvel UBM est utilisé pour extraire les données du locuteur cible, et les données opposées au locuteur cible, comme décrit dans les sections suivantes.

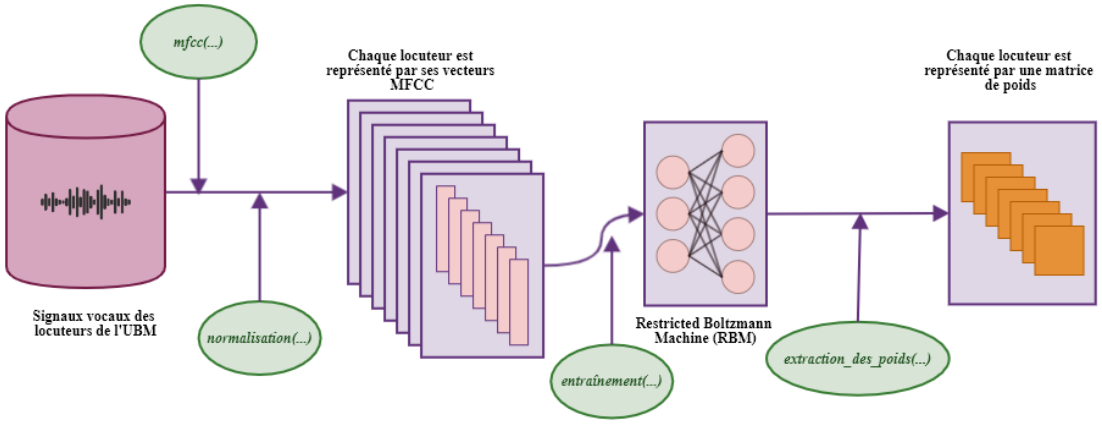


Figure 7.1: Illustration de la génération de matrices à partir de l'UBM ; les paramètres MFCC sont extraits de chaque locuteur UBM, puis normalisés, et en utilisant la RBM, les vecteurs caractéristiques de chaque locuteur sont transformés en une matrice.

7.3.1.2 Création de données du locuteur cible

Afin d'augmenter les données du locuteur cible, nous effectuons une segmentation de ses vecteurs MFCC. Concrètement, nous choisissons au hasard un certain nombre de vecteurs MFCC et nous les rassemblons dans un segment, puis nous répétons ce processus jusqu'à ce que nous obtenions autant de segments que le nombre de matrices UBM, voir la figure 7.2. Ensuite, les vecteurs MFCC de chaque segment sont transmis à toutes les RBM, qui sont initialisées par les matrices UBM. Enfin, chaque RBM produise une matrice de poids, et chaque segment est représenté par le même nombre de matrices de poids que l'UBM.

7.3.1.3 Création de données du locuteur non-cible

La figure 7.3 représente la procédure de construction des données du locuteur non-cible. Afin de construire ces données, nous effectuons une sélection de caractéristiques. Par conséquent, les données UBM initiales sont divisées en deux groupes équitables. Le premier groupe est stocké sans modification. Cependant, nous appliquons la méthode de plus proche cluster (NearC), voir chapitre 5, au second groupe. Fondamentalement, la méthode NearC est une méthode d'évaluation pour la vérification automatique du locuteur, elle consiste à regrouper les vecteurs caractéristiques du locuteur cible, et à calculer la distance entre les vecteurs de MFCC du locuteur de test et tous les clusters du locuteur cible, et à renvoyer la distance minimale. Dans notre cas, nous voulons appliquer cette méthode aux locuteurs du second groupe de l'UBM. Par conséquent, pour chaque locuteur de ce groupe, nous ne gardons que le groupe le plus proche du locuteur cible. En

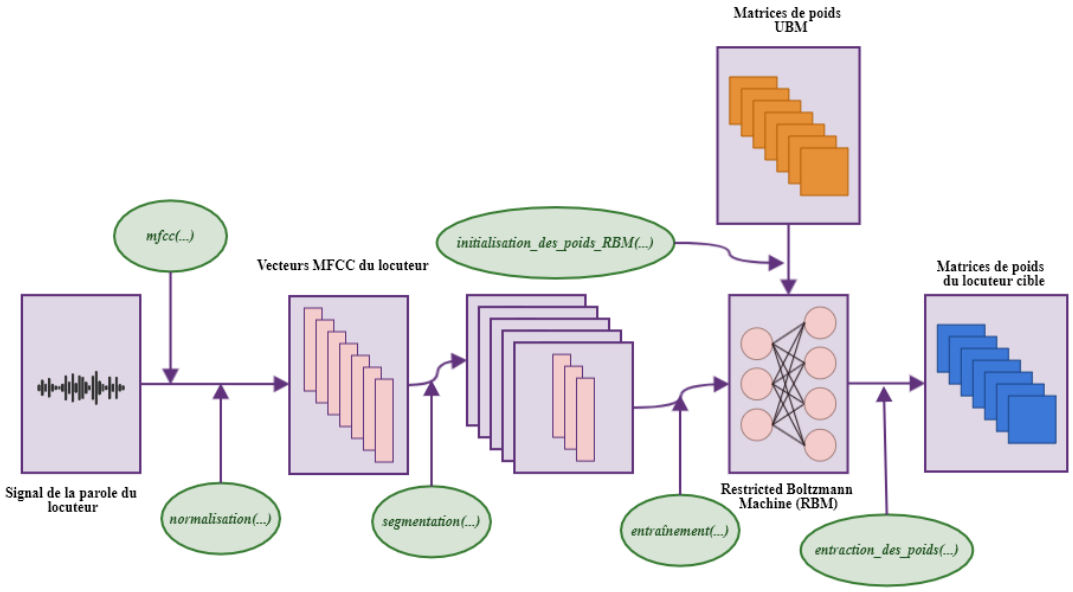


Figure 7.2: Processus de génération des matrices du locuteur cible.

pratique, la distance entre un vecteur i d'un vecteur MFCC du locuteur cible et un cluster C d'un locuteur UBM est définie par,

$$d(S_{cible}^{(i)}, C^{(k)}) = \min_{j=1,\ldots,L} d(S_{ubm}^{(i)}, C_{(j)}^{(k)}), \tag{7.1}$$

d indique la distance en cosinus, S_{cible} représente les vecteurs caractéristiques du locuteur cible, C représente un groupe, k représente le $k^{ème}$ groupe, L indique le nombre de vecteurs MFCC dans le cluster. Alors que la distance entre les vecteurs MFCC du locuteur cible et le cluster k est définie par,

$$d(S_{cible}, C^{(k)}) = \frac{1}{N} \sum_{i=1}^{N} d(S_{ubm}^{(i)}, C^{(k)}), \tag{7.2}$$

avec N signifie le nombre de vecteurs MFCC du locuteur cible. Par conséquent, la distance entre les vecteurs MFCC du locuteur cible et les vecteurs MFCC du locuteur UBM est définie par,

$$d(S_{cible}, S_{ubm}) = \min_{k=1,\ldots,K} d(S_{cible}, C^{(k)}). \tag{7.3}$$

Finalement, nous rassemblons les clusters les plus proches du locuteur cible, et nous les mettons dans le second groupe, afin de former des vecteurs caractéristiques de locuteurs non-cibles, voir la figure 7.3. Ensuite, nous construisons des matrices pour le locuteur non-cible en utilisant la même méthode que pour les données du locuteur cible, mais sans traiter la phase de segmentation, car nous avons déjà le même nombre de segments dans les vecteurs MFCC du locuteur non-cible (100 locuteurs). Par conséquent, les vecteurs MFCC de chaque locuteur sont propagés directement dans la RBM correspondante, voir la figure 7.2.

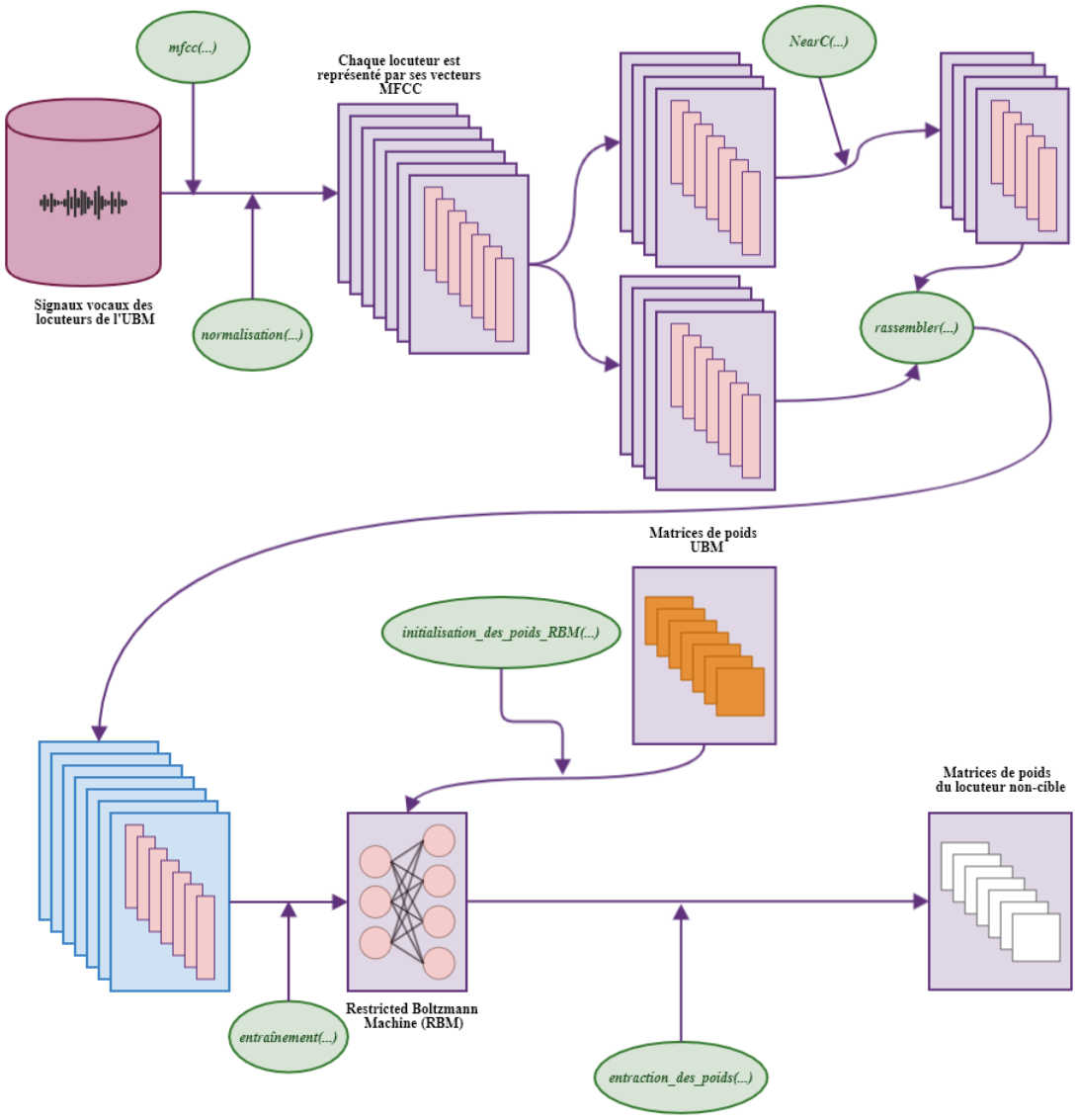


Figure 7.3: Processus de génération des matrices du locuteur non-cible.

7.4 Résultats

7.4.1 Environnement expérimental

La méthode proposée a été évaluée sur ces configurations: (i) PC, CPU: Intel Core^{TM} i7-7700HQ 2,80 Ghz; GPU: Nvidia GeForce GTX 1050 4 Go V-RAM, cœurs CUDA: 640, Bande passante mémoire: 112,13 Go/s; Mémoire cache 6 Mo, RAM: 16,0 Go. (ii) Système d'exploitation: Microsoft Windows 10 Professionnel, (iii) Langage de programmation: Python 3.6; Frameworks: TensorFlow (pour RBM) et Keras (pour CNN).

7.4.2 Protocole expérimental

Dans la phase d'extraction des caractéristiques, nous décomposons le signal en trames de 25 ms, et nous fixons le pas de la fenêtre de hamming à 10 ms. Par conséquent, pour chaque seconde du son de la parole, nous obtenons 100 vecteurs MFCC. En effet, les caractéristiques MFCC à 39 dimensions sont extraites, y compris les MFCC statiques à 13 dimensions, dont le coefficient C0 est remplacé par de l'énergie. Ensuite, ces coefficients sont concaténés avec leurs dérivées premières et secondes respectivement. Par conséquent, les vecteurs des MFCC sont normalisés en utilisant la normalisation (CMVN) [90], ce qui permet de supprimer l'effet du canal.

Tout d'abord, nous avons commencé par transformer les données de l'UBM en matrices ; l'UBM est composé de 100 locuteurs, et chaque locuteur a sa matrice correspondante de MFCC normalisés, dans laquelle chaque colonne représente un vecteur MFCC. Par la suite, chaque matrice est propagée dans la RBM, et chaque colonne est traitée comme un vecteur visible de RBM. Nous voudrions construire des matrices dans lesquelles chaque matrice représente un locuteur de l'UBM. Pour cette raison, nous avons choisi une architecture RBM de 39 unités par couche visible, et 39 unités par couche cachée. Nous avons utilisé l'algorithme de divergence contrastive [116] pour reconstruire les unités visibles. Après la reconstruction des données, la matrice de poids de dimensions 39×39 est extraite pour chaque locuteur UBM.

Ensuite, nous devons extraire les matrices de poids correspondantes de chaque locuteur dans l'ensemble des locuteurs d'entraînement (locuteurs cibles), composée de 3,000 vecteurs caractéristiques MFCC pour chaque locuteur. Mais tout d'abord, nous avons dû augmenter les données de chaque locuteur cible. Pour cette raison, nous avons effectué une phase de sélection aléatoire de 100 vecteurs caractéristiques MFCC parmi les 3,000 vecteurs MFCC de chaque locuteur, et nous avons répété cette procédure pendant 100 étapes, ce qui a donné 100 segments, dans lesquels chaque segment est constitué de 100 vecteurs MFCC. Par la suite, chaque segment est envoyé à une RBM, dans lequel ses poids sont initialisés par une matrice de poids UBM. Par conséquent, 100 RBM sont initialisés avec les 100 matrices de poids UBM, et apprennent les segments du locuteur cible, ce qui donne 10,000 matrices de poids représentant le locuteur cible.

Troisièmement, après la construction des données du locuteur cible, nous devions construire les données du locuteur non-cible. Par conséquent, nous avons divisé les données initiales de l'UBM en deux groupes équitables. Les vecteurs MFCC du premier groupe sont conservés sans aucune modification. En outre, pour le second groupe, nous avons effectué une phase de sélection des vecteurs MFCC.

En effet, nous avons appliqué la méthode NearC, pour choisir le groupe le plus proche au locuteur cible. Le nombre de clusters de chaque locuteur est fixé à 23 (voir chapitre 5), et le cluster le plus proche du locuteur cible est renvoyé. Par conséquent, nous avons rajouté ces clusters au premier groupe, pour construire des vecteurs MFCC du locuteur non-cible. Par conséquent, chaque locuteur cible de l'ensemble d'entraînement a ses données non-cible correspondantes.

Les données non-cibles sont constituées de 100 matrices de vecteurs MFCC des locuteurs, chaque matrice est propagée dans une RBM, dans laquelle ses poids sont initialisés par une matrice UBM. Par conséquent, 100 RBM sont initialisées avec 100 matrices de poids UBM et apprennent les vecteurs MFCC non-cibles, ce qui donne 10,000 matrices de poids non-cibles.

Enfin, et après avoir obtenu un ensemble de données équitables comprenant 10,000 matrices du locuteur cible et 10,000 matrices du locuteur non-cible, nous pouvons mener un processus de vérification automatique du locuteur en utilisant un CNN.

Les dimensions de la matrice d'entrée du CNN est de 39×39, puis nous avons appliqué trois couches de convolution suivies de trois couches de max-pooling, avec les filtres 32, 64 et 128 respectivement. Les tailles du filtre et du pooling sont respectivement (3,3) et (2,2). De plus, les matrices résultantes sont ensuite aplaties et propagées dans un réseau de neurones profond (DNN), qui consiste en 4 couches cachées de 1024 unités, et une sortie sigmoïde qui a effectué une classification binaire. Nous notons que la fonction ReLU est utilisée pour toutes les couches comme fonction d'activation, sauf pour la couche de sortie, où la sigmoïde est utilisée.

Des expériences ont été menées à l'aide de corpus THUYG, en utilisant d'ensemble des locuteurs féminins et masculins, dans lesquels 87 locuteurs sont des femmes et 66 des hommes, et l'ensemble de locuteurs UBM féminins et masculins, dans lesquels 100 locuteurs dans chacun. En suivant le même protocole expérimental que les chapitres précédents.

7.4.3 Résultats et discussion

Le tableau 7.1 présente les résultats obtenus sur le corpus THUYG pour les ensembles de locuteurs féminins et masculins, en utilisant trois conditions de bruit (nette, 9db et 0db). Nous avons appliqué notre méthodologie en trois scénario : dans le scénario (i), M1:CNN/UBM, le réseau CNN est entraîné pour distinguer entre 10,000 matrices du locuteur cible et 10,000 matrices UBM ; tandis que dans scénario (ii) M2:CNN/NearC, nous extrayons de chaque locuteur UBM le cluster le plus proche du locuteur cible, par conséquent, chaque locuteur UBM est représenté

Tableau 7.1: Présentation des résultats (en % EER) des méthodes suivantes : M1:CNN/UBM, M2:CNN/NearC, et M3:CNN/UBM-NearC (M1+M2) en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db.

Locuteurs féminins		Méthodes proposées		
Cible (SNR)	Test (SNR)	CNN/UBM	CNN/NearC	CNN/UBM-NearC
Nette	Nette	2.45	1.75	0.34
Nette	9db	4.77	2.10	0.81
Nette	0db	7.35	2.88	1.02
9db	Nette	4.15	2.11	0.99
9db	9db	3.82	3.17	1.51
9db	0db	6.43	5.88	2.97
0db	Nette	6.78	6.52	3.44
0db	9db	5.19	4.71	2.85
0db	0db	5.92	4.59	2.33
AVG EER		5.21	3.75	1.81
Locuteurs masculins		Méthodes proposées		
Cible (SNR)	Test (SNR)	CNN/UBM	CNN/NearC	CNN/UBM-NearC
Nette	Nette	2.56	1.82	0.42
Nette	9db	4.90	2.33	0.84
Nette	0db	7.48	3.12	1.09
9db	Nette	4.30	2.28	1.05
9db	9db	3.88	3.54	1.50
9db	0db	6.49	6.12	2.98
0db	Nette	6.90	6.77	3.52
0db	9db	5.23	4.80	2.99
0db	0db	6.17	4.55	2.50
AVG EER		5.32	3.93	1.88

par la matrice de poids produite par son cluster; le scénario (iii), M3:CNN/UBM-NearC (M1+M2) nous avons combiné le M1:CNN/UBM et le M2:CNN/NearC, en effet, nous avons divisé divisé l'ensemble des locuteurs UBM en deux groupes, dans le premier groupe nous avons appliqué la méthode M1, et dans le deuxième groupe nous avons appliqué la méthode M2. D'après les résultats reportées par le tableau 7.1, nous remarquons que le méthode CNN/UBM-NearC surpasse les deux autres méthodes proposées dans tous les scénarios. En effet, pour l'ensemble des locuteurs féminins, la méthode CNN/UBM-NearC a enregistré un taux d'erreur moyen de 1,81% EER, tandis que les autres méthodes, CNN/UBM et CNN/NearC ont enregistré respectivement 5,21% EER et 3,75% EER.

Tableau 7.2: Présentation des résultats (en % EER) des méthodes suivantes : *i*-vector/PLDA, DeepSF/NearC, et CNN/UBM-NearC, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db. (Le taux d'erreur qui prend en considération à la fois les locuteurs masculins et féminins est calculé selon l'équation suivante : $EER = \frac{(N_m \times EER_m)+(N_f \times EER_f)}{N_m+N_f}$, avec N, m et f font référence respectivement aux nombre total de locuteurs, masculin, et féminin.)

Locuteurs		Méthodes		
Cible (SNR)	Test (SNR)	*i*-vector/PLDA	DeepSF/NearC	CNN/UBM-NearC
Nette	Nette	2.26	0.48	0.37
Nette	9db	4.31	0.99	0.82
Nette	0db	8.37	1.17	1.05
9db	Nette	3.97	0.90	1.02
9db	9db	3.69	1.45	1.51
9db	0db	5.58	2.68	2.97
0db	Nette	6.33	3.12	3.47
0db	9db	4.77	2.90	2.91
0db	0db	5.26	2.33	2.40
AVG EER		4.95	1.78	1.84

Tableau 7.3: Présentation des résultats (en % EER) des méthode: *i*-vector/PLDA, DeepSF/NearC et la méthode proposée CNN/UBM-NearC en utilisant le corpus THUYG avec différentes durées de segments de test, et dans des conditions de paroles nettes.

Durée (seconds)	1	2	3	4	5	6	7	8	9	10
i-vector/PLDA	7.72	5.23	4.77	4.32	4.02	3.15	2.25	2.28	2.26	2.26
DeepSF/NearC	0.82	0.80	0.79	0.58	0.60	0.55	0.56	0.58	0.52	0.48
CNN/UBM-NearC	1.04	0.97	0.82	0.45	0.39	0.41	0.41	0.41	0.36	0.37

Dans le tableau 7.2, nous présentons les résultats de CNN/UBM-NearC, *i*-vector/PLDA et DeepSF/NearC. Pour la méthode *i*-vector/PLDA, l'UBM est entraîné en utilisant le corpus Fisher (nous pouvons utiliser les données UBM du corpus THUYG, mais les résultats ne sont pas assez bon que ceux obtenus en utilisant le corpus Fisher, voir chapitre 4). Pour la méthode DeepSF/NearC, chapitre 6, nous n'avons pas besoin de données UBM. D'après les résultats obtenus, la méthode proposée surpasse la méthode *i*-vector/PLDA dans toutes les conditions, ce que nous avons déjà réussi à faire dans le chapitre précédent en utilisant DeepSF/NearC. En outre, la méthode proposée a pu surpasser DeepSF/NearC dans les conditions nettes de la parole, en enregistrant des taux d'erreurs de 0.37% EER, tandis que DeepSF/NearC a enregistré 0.48% EER. Cependant, la méthode proposée n'a pas

pu surpasser DeepSF/NearC dans les conditions où la parole des locuteurs cibles est dans des conditions de bruit (9db et 0db).

Nous avons ensuite procédé à d'autres expérimentations pour tester différents segments de test. Les résultats sont présentés dans le tableau 7.3. Nous remarquons que la méthode CNN/UBM-NearC présente les meilleurs résultats à partir de 4 secondes de segment de test.

7.5 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle méthode pour modéliser les locuteurs en utilisant des architectures de deep learning. Nous avons utilisé la machine de Boltzmann restreinte (RBM) pour extraire des matrices de poids, qui sont utilisées comme entrée dans un réseau de neurones convolutif (CNN). Dans ce contexte, nous avons présenté trois contributions principales. Premièrement, nous avons pu mettre en œuvre des CNN dans la RAL sans spectrogrammes. Deuxièmement, nous avons introduit une nouvelle méthode pour modéliser les locuteurs cibles et non-cibles dans un problème de vérification automatique du locuteur. Troisièmement, nous avons proposé une méthode originale pour utiliser le modèle UBM afin de générer des données supplémentaires pour les locuteurs. Des expériences ont été menées sur le corpus THUYG, dans trois conditions de bruit, en utilisant les trois méthodes proposées: M1:CNN/UBM, M2:CNN/NearC, et M3:CNN/UBM-NearC. Les résultats montrent que les méthodes proposées sont plus performantes que les approches de pointe. En outre, la combinaison de M1 et M2 diminue sensiblement le taux d'erreur.

Au vu de ces résultats, nous pouvons conclure que la matrice de poids montre sa capacité à préserver les caractéristiques importantes du locuteur. En outre, nous pensons que la méthode proposée, pour modéliser le locuteur non-cible, peut couvrir un plus grand nombre de locuteurs dont la voix est proche de celle du locuteur cible.

Les résultats actuels confirment notre hypothèse initiale. Cependant, l'apprentissage est une phase qui prend de temps, ce qui ne nous empêche d'utiliser un ensemble de données plus important.

8

Vecteurs du Réseau de Neurones Convolutif pour la RAL — ConvVectors

Sommaire

8.1 Introduction

Dans ce chapitre, nous proposons une nouvelle utilisation des réseaux de neurones convolutifs (CNN) pour le problème de la reconnaissance automatique du locuteur. Bien qu'ils soient particulièrement conçus pour les problèmes de reconnaissance des images, les CNN ont récemment été appliqués au RAL en utilisant des spectrogrammes comme images d'entrée. Nous pensons que cette approche n'est pas optimale car elle peut conduire à deux erreurs cumulatives dans la résolution d'un problème de traitement d'image et de RAL. Dans ce travail, l'objectif est

de développer un réseau de neurones convolutif personnalisé pour la RAL. Pour ce faire, nous proposons une nouvelle approche pour extraire les caractéristiques du locuteur en construisant des filtres CNN liés à ce locuteur. De plus, nous proposons de nouveaux vecteurs pour identifier les locuteurs, que nous appelons dans ce travail "convVectors". Les expériences ont été réalisées en utilisant le corpus THUYG dans trois conditions de bruit : nette, 9db, et 0db. Nous avons comparé la méthode proposée avec nos travaux précédents. Les résultats ont montré que la méthode des convVectors était la plus robuste. C'est une conclusion importante pour comprendre comment les modèles de deep learning peuvent être adaptés au problème de la reconnaissance automatique du locuteur.

L'architecture de ce chapitre est la suivante : dans la section 8.2, nous présentons les travaux correspondants. Dans la section 8.3, nous présentons la méthode proposée. Dans la section 8.4, nous présentons les résultats et discutons. La section 8.5 représente la conclusion de ce chapitre.

8.2 Travaux connexes

Les CNN ont été utilisées dans la RAL en exploitant leurs capacités de traitement des données 2D dans le domaine de traitement d'images. Par conséquent, la plupart des travaux se concentrent sur l'extraction des caractéristiques du locuteur à partir de la représentation visuelle du spectre de fréquences appelée spectrogramme [113–115, 117], tandis que peu de travaux utilisent des formes d'onde brutes [118, 119].

Le défi à relever dans ce domaine est l'adoption des architectures de deep learning pour les caractéristiques de la voix, en particulier en utilisant les CNN qui sont plus adaptées à l'image [52]. À notre connaissance, nos travaux antérieurs (chapitre 7) ont été les premiers à utiliser les CNN sans recourir aux approches du traitement d'images, et aucune étude n'a été menée sur l'adoption des CNN dans la RAL sans transformer le problème de la RAL en problème de traitement d'images.

Dans le chapitre précédent, nous avons utilisé différentes architectures de deep learning pour le système de vérification automatique du locuteur proposé. En effet, nous avons utilisé la RBM pour transformer les vecteurs caractéristiques du locuteur en une matrice de poids. Par la suite, nous avons généré des matrices pour le locuteur cible et pour son opposé, dit non-cible. Nous avons ensuite utilisé le réseau CNN pour distinguer les matrices du locuteur cible des matrices du locuteur non-cible.

Dans ce chapitre, l'objectif est de développer une méthode plus sophistiquée de vérification automatique du locuteur, en utilisant les modèles de deep learning RBM et CNN, afin de former un vecteur représentant le locuteur, que nous appelons

"convVector" (c'est-à-dire un vecteur de réseau de neurone convolutif). Le processus d'extraction des convVectors comporte trois phases: (i) la transformation des données du modèle UBM en matrices ; (ii) la construction des filtres CNN à partir des données du locuteur cible ; (iii) la construction du modèle CNN et l'extraction de convVector pour le locuteur cible.

Les principales contributions de ce travail sont présentées comme suit:

- La proposition d'une approche originale pour extraire les caractéristiques du locuteur en formant des filtres CNN correspondant à ce locuteur ;
- La proposition de nouveaux vecteurs pour identifier les locuteurs : ConvVectors.

8.3 Méthodologie

Dans ce travail, nous proposons une approche de deep learning pour la reconnaissance automatique du locuteur. Nous avons utilisé les RBM pour deux raisons : premièrement, pour construire l'UBM en transformant les vecteurs caractéristiques de chaque locuteur UBM en une matrice de poids (même procédure que pour le chapitre 7) ; et deuxièmement, pour transformer les vecteurs caractéristiques du locuteur en vecteurs binaires. Nous avons ensuite transformé les vecteurs binaires en matrices qui sont utilisées comme filtres pour le modèle CNN. Nous avons donc construit le CNN, en commençant par la couche d'entrée composée de matrices UBM. Par la suite, les couches de convolution sont obtenues en utilisant les filtres obtenu du locuteur cible. Finalement, les vecteurs aplatis sont extraits pour construire le convVector du locuteur.

8.3.1 Création de données UBM

Pour construire l'UBM, nous avons utilisé la même approche que dans nos travaux précédents (chapitre 7). Par conséquent, nous avons transformé les vecteurs MFCC de chaque locuteur dans l'ensemble de données UBM en une matrice carrée. En pratique, les poids de la RBM sont initialisés aléatoirement aux valeurs réelles $\in [0, 1]$. Par conséquent, les vecteurs MFCC de chaque locuteur dans le modèle UBM sont transmis au modèle RBM, qui tente de les reconstruire. Lorsque la RBM atteint la précision d'apprentissage appropriée, nous renvoyons les valeurs de poids entre les couches visible et cachée, qui sont représentées par une matrice. Finalement, l'UBM qui en résulte ne comprend que des matrices, au lieu de vecteurs MFCC, voir la figure 8.1.

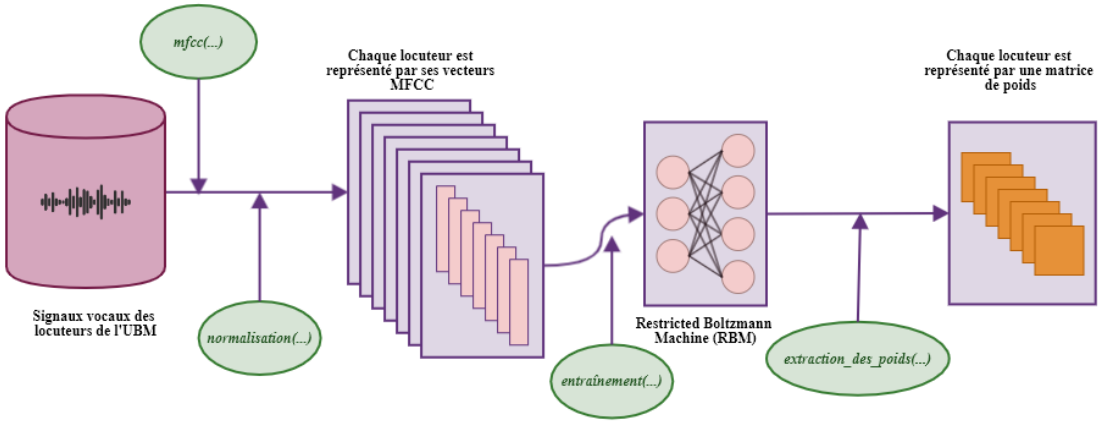


Figure 8.1: Illustration de la génération de matrices à partir de l'UBM ; les paramètres MFCC sont extraits de chaque locuteur UBM, puis normalisés, et en utilisant la RBM, les vecteurs caractéristiques de chaque locuteur sont transformés en une matrice.

8.3.2 Extraction des filtres pour le modèle CNN

Les vecteurs MFCC du locuteur cible sont utilisés pour extraire les filtres pour le modèle CNN. En pratique, les RBM ont été configurées pour n'apprendre qu'un seul vecteur caractéristique pour chaque phase d'apprentissage, puis les valeurs de la couche cachée sont extraites. Par conséquent, chaque vecteur MFCC du locuteur est converti en un vecteur binaire, voir la figure 8.2 A et B. Le nouvel ensemble de vecteurs binaires peut contenir de nombreux vecteurs identiques. Pour cette raison, nous regroupons l'ensemble de vecteurs en utilisant l'algorithme de clustering DB-SCAN [52]. Dans l'algorithme DB-SCAN, nous avons utilisé la distance de Hamming [120] en considérant deux vecteurs binaires similaires si et seulement si leur distance est égale à zéro. Nous obtenons donc des clusters, et chaque cluster comprend des vecteurs binaires similaires. Cependant, certains clusters peuvent inclure plus de vecteurs que d'autres. Pour mettre en évidence cette caractéristique, nous avons étiqueté les clusters, en donnant la priorité au cluster qui contient le plus de vecteurs sur le cluster qui en contient le moins. Ensuite, nous réduisons chaque cluster à un seul vecteur binaire que nous étiquetons avec le nombre de ses occurrences. Finalement, pour construire les filtres du locuteur cible, nous avons

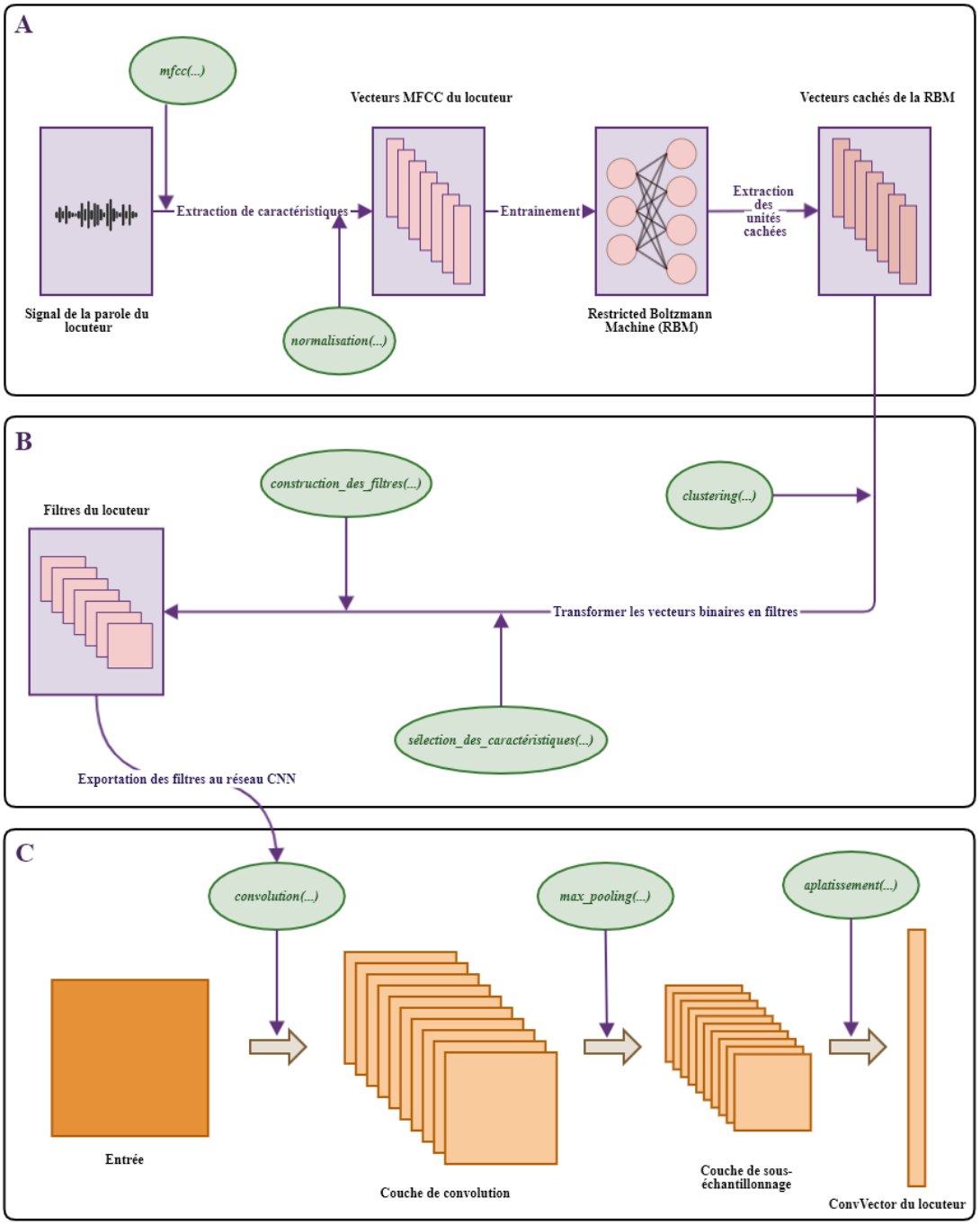


Figure 8.2: Une illustration de la méthode proposée : La partie (A) représente le processus d'extraction des vecteurs de caractéristiques binaires ; le signal vocal du locuteur est transformé en vecteurs caractéristiques MFCC normalisés, et chaque vecteur MFCC est converti en un vecteur binaires en utilisant une RBM. La partie (B) représente le processus de transformation des vecteurs binaires en filtres ; les vecteurs binaires sont regroupés, un ensemble de groupes est sélectionné, et chaque groupe est représenté par un vecteur binaire et transformé en filtre. La partie (C) représente le processus de construction du CNN ; les entrées sont générées par l'UBM (voir Fig. 8.1), les couches de convolution sont calculées en appliquant les filtres obtenus à partir de la partie (B), le max-pooling est appliqué après chaque couche de convolution, et à la fin un vecteur aplati est calculé représentant le convVector du locuteur.

transformé chaque vecteur binaire en une matrice en appliquant une multiplication binaire entre le vecteur binaire h et sa transposition h^T, avec,

$$Filtre = h.h^T \tag{8.1}$$

8.3.3 Architecture CNN proposée

Le processus d'extraction de caractéristiques à l'aide du réseau CNN commence par l'application d'un filtre glissant (également appelé noyau) sur la matrice d'entrée, ce qui donne une couche de convolution. Par conséquent, la dimension de la couche de convolution est réduite par une couche de regroupement. Le processus peut être répété si nécessaire, et les matrices finales sont aplaties pour construire un vecteur unidimensionnel qui représente l'entrée du réseau de neurones entièrement connecté, voir la figure 8.3.

Dans ce travail, nous proposons une utilisation différente des CNN. En effet, notre objectif n'est pas d'utiliser leur capacité d'apprentissage. Cependant, nous avons l'intention d'utiliser la capacité des CNN en termes d'extraction de caractéristiques pour extraire un vecteur représentant le locuteur. Par conséquent, nous développons l'architecture des CNN bloc par bloc. Nous utilisons d'abord des matrices UBM comme entrées de CNN. Ensuite, nous utilisons les filtres extraits du locuteur cible comme filtres du CNN afin d'obtenir une couche de convolution. Troisièmement, nous appliquons le max-pooling pour réduire la dimensionnalité de la couche de convolution. Nous répétons ce processus comme décrit dans le protocole expérimentale. Quatrièmement, chaque matrice résultante est aplatie en concaténant ses lignes pour former un vecteur. Finalement, nous calculons la moyenne des vecteurs des matrices pour former un vecteur représentant le locuteur cible, appelé convVector.

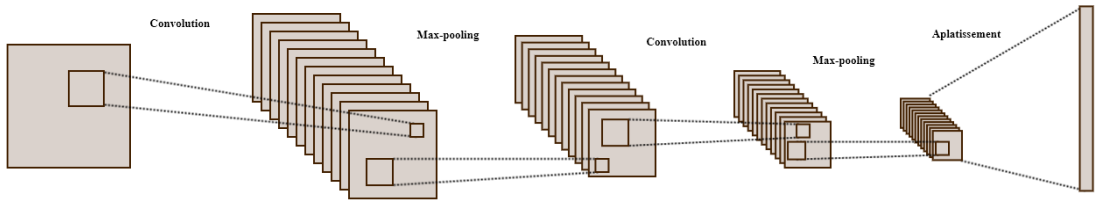


Figure 8.3: Illustration de la structure du réseau de neurones convolutif, elle se compose de : couche d'entrée (à gauche), couches de convolution, couches de sous-échantillonnage, et une couche entièrement connectée (à droite).

8.4 Résultats

8.4.1 Environnement expérimental

La méthode proposée a été évaluée sur ces configurations: (i) PC, CPU: Intel Core^{TM} i7-7700HQ 2,80 Ghz; GPU: Nvidia GeForce GTX 1050 4 Go V-RAM, cœurs CUDA: 640, Bande passante mémoire: 112,13 Go/s; Mémoire cache 6 Mo, RAM: 16,0 Go. (ii) Système d'exploitation: Microsoft Windows 10 Professionnel, (iii) Langage de programmation: Python 3.6; Frameworks: TensorFlow (pour RBM) et Keras (pour CNN).

8.4.2 Protocole expérimental

L'UBM est indépendant du genre, nous dans chaque ensemble avons 100 locuteurs, chacun est représenté par une matrice carrée avec 39 x 39. Le réseau RBM est composé de 39 unités dans chaque couche. Pour former la RBM, nous avons utilisé l'algorithme de divergence contrastive (chapitre 3, Algorithme 1). Ensuite, nous avons extrait la matrice de poids de chaque RBM. Le modèle UBM qui en résulte est constitué de 100 matrices carrées, 39×39, qui constituent la couche d'entrée du réseau CNN.

Ensuite, nous avons extrait les filtres du locuteur cible. Chaque ensemble de vecteurs MFCC du locuteur cible est constitué de 3,000 vecteurs, et chaque vecteur est transformé en filtre. En pratique, nous avons utilisé trois architectures RBM : RBM1 (v = 39, h= 12), RBM2 (v = 39, h= 9), et RBM3 (v = 39, h= 6). Pour chaque vecteur MFCC, nous obtenons trois vecteurs binaires de dimensions 12, 9 et 6 respectivement. Nous obtenons ainsi trois groupes de vecteurs binaires : G1(3000, 12), G2(3000, 9), et G3(3000, 6). Pour chaque ensemble, nous avons procédé à la sélection de vecteurs MFCC comme expliqué dans la section 8.3 ; pour G1, nous avons conservé les 32 vecteurs binaires correspondant aux 32 premiers groupes en donnant la priorité aux vecteurs en fonction de la taille du groupe qu'ils représentent comme expliqué dans la section 8.3.2, pour G2, et G3 nous avons conservé respectivement 64 et 128 vecteurs binaires. Finalement, les vecteurs binaires sont transformés en filtres, ce qui donne trois groupes de filtres : conv(32,12,12), conv(64,9,9) et conv(128,6,6). Dans cette expérience, nous avons utilisé les configurations CNN suivantes :

- La première configuration du réseau CNN :
  - C1 : M(100, 39, 39) + conv(32, 12,12) + max-pooling + aplatissement;
  - C2: M(100, 39, 39) + conv(64, 9, 9) + max-pooling + aplatissement;

- C3: M(100, 39, 39) + conv(128, 6, 6) + max-pooling + aplatissement.

- La deuxième configuration du réseau CNN :

  - C1 + C2: M(100, 39, 39) + conv(32, 12,12) + max-pooling + conv(64, 9, 9) + max-pooling + aplatissement;

  - C1 + C3: M(100, 39, 39) + conv(32, 12,12) + max-pooling + conv(128, 6, 6) + max-pooling + aplatissement;

  - C2 + C3: M(100, 39, 39) + conv(64, 9, 9) + max-pooling + conv(128, 6, 6) + max-pooling + aplatissement.

- La troisième configuration du réseau CNN :

  - C1 + C2 + C3: M(100, 39, 39) + conv(32, 12,12) + max-pooling + conv(64, 9, 9) + max-pooling + conv(128, 6, 6) + max-pooling + aplatissement.

Les expériences ont été menées à partir du corpus THUYG, en utilisant des ensembles de locuteurs féminins et masculins, dans lesquels 87 locuteurs sont des femmes et 66 des hommes. Nous avons suivi le même protocole expérimental que les travaux précédents. Par conséquent, nous avons testé les locuteurs les uns par rapport aux autres, ce qui a donné un total de 119,277 essais pour l'ensemble des locuteurs féminins et 63,361 essais pour l'ensemble des locuteurs masculins. Il convient de noter que les locuteurs testés ont été traités comme des locuteurs cibles dans le processus d'extraction des convVectors, et la distance entre les convVectors cible et de test est calculée en utilisant leur similarité cosinus.

8.4.3 Résultats et discussions

Tableau 8.1: Présentation des résultats (en % EER) de l'utilisation de différentes tailles et combinaisons de couche de convolution : C1=conv(12,12), C2=conv(9,9), et C3=conv(6,6), en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db.

Locuteurs féminins		Architectures CNN						
Cible (SNR)	Test (SNR)	C1	C2	C3	C1+C2	C1+C3	C2+C3	C1+C2+C3
Nette	Nette	1.02	0.98	1.58	0.17	0.13	0.11	0.12
Nette	9db	1.28	1.32	1.92	0.25	0.29	0.31	0.26
Nette	0db	1.65	1.77	2.01	0.42	0.51	0.52	0.39
9db	Nette	1.88	1.92	2.31	1.07	1.10	1.11	1.04
9db	9db	1.77	1.73	3.15	1.08	1.02	0.92	0.95
9db	0db	2.40	2.45	2.88	1.71	1.75	2.07	1.69
0db	Nette	1.95	1.97	2.28	1.58	1.61	1.63	1.60
0db	9db	2.00	2.01	2.34	1.65	1.73	1.72	1.66
0db	0db	2.21	2.19	2.51	1.67	1.61	1.42	1.44
AVG EER		1.80	1.82	2.33	1.07	1.08	1.09	1.02

Locuteurs masculins		Architectures CNN						
Cible (SNR)	Test (SNR)	C1	C2	C3	C1+C2	C1+C3	C2+C3	C1+C2+C3
Nette	Nette	1.11	1.10	1.64	0.19	0.15	0.15	0.15
Nette	9db	1.33	1.39	2.01	0.28	0.32	0.37	0.35
Nette	0db	1.76	1.80	2.12	0.41	0.54	0.59	0.45
9db	Nette	1.89	1.95	2.35	1.11	1.12	1.15	1.13
9db	9db	1.81	1.74	2.16	1.09	1.05	0.98	1.00
9db	0db	2.48	2.40	2.87	1.74	1.77	2.06	1.74
0db	Nette	1.99	2.02	2.33	1.62	1.63	1.68	1.65
0db	9db	2.06	2.11	2.34	1.66	1.75	1.80	1.79
0db	0db	2.26	2.23	2.55	1.68	1.62	1.49	1.45
AVG EER		1.85	1.86	2.26	1.09	1.11	1.14	1.08

Le tableau 8.1 représente les résultats de la méthode proposée pour les ensembles de locuteurs féminins et masculins, en utilisant sept configurations CNN: C1, C2, C3, C1+C2, C1+C3, C2+C3, C1+C2+C3. Les résultats montrent que C1 donne le taux d'erreur moyen le plus faible par rapport à C2 et C3, avec 1,80% EER pour les locuteurs féminines et 1,85% EER pour les locuteurs masculines. D'après les résultats de C1, C2 et C3, il est clair que le taux d'erreur moyen diminue avec

la diminution de la taille de la couche de convolution. Suivant la même logique, la composition de C1 et C2 donne de bien meilleurs résultats que l'utilisation d'une seule couche de convolution. En effet, le taux d'erreur moyen est diminué de 1,80% EER à 1,07% EER pour les locuteurs féminines. Les résultats de C1+C2 sont améliorés en rajoutant C3, c'est-à-dire C1+C2+C3, à 1,02% EER pour les locuteurs féminins, et à 1,08% EER pour l'ensemble locuteurs masculins, qui sont les meilleurs résultats que nous ayons observés.

D'après les résultats, il est clair que la configuration CNN (C1+C2+C3) donne les meilleurs résultats. Les résultats de cette méthode sont ensuite comparés aux résultats de l'approche *i*-vector/PLDA (chapitre 4), à l'approche DeepSF/NearC (chapitre 6), et aux travaux du chapitre précédent (l'approche CNN/UBM-NearC). Pour l'*i*-vector/PLDA, l'UBM implique 2048 composants gaussiens, et 400 est utilisé comme dimension de l'*i*-vector. Pour DeepSF/NearC, l'UBM n'est pas impliqué, les vecteurs MFCC sont transformés en vecteurs DeepSF et la la méthode NearC (chapitre 5) est utilisée pour l'étape d'évaluation. Pour l'approche CNN/UBM-NearC, nous avons utilisé la même configuration UBM que celle utilisée dans ce travail (chapitre 7).

Le tableau 8.2 montre les résultats des différentes méthodes utilisant les ensembles de données féminins et masculins, dans trois conditions de bruit (nette, 9db et 0db). Dans l'un de nos travaux précédents (chapitre 6), l'approche DeepSF/NearC surpasse l'approche *i*-vector/PLDA en diminuant le taux d'erreur de 4,95% EER à 1,78% EER. Dans notre travail précédent (chapitre 7), les résultats montrent que l'approche CNN/UBM-NearC ne fonctionne mieux que l'approche DeepSF/NearC que lorsque la parole des locuteurs cibles est nette. D'autre part, la méthode proposée va au-delà des rapports précédents, en montrant que les convVectors sont ceux qui ont obtenu les résultats les plus robustes. Lorsque l'on compare les résultats des convVectors à ceux d'études plus anciennes, il convient de noter que le taux d'erreur le plus faible est indiqué lorsque les données des locuteurs cibles et tests sont de parole nette, c'est-à-dire 0,13% EER. De plus, les convVectors diminuent le taux d'erreur moyen de 1,84% EER à 1,04% EER par rapport au système de base du chapitre précédent.

Nous avons ensuite procédé à d'autres expérimentations pour tester différents segments de test. Les résultats sont présentés dans le tableau 8.3. À cause de l'architecture de la méthode proposée, nous ne pouvons pas avoir des segments de test inférieur à 3 secondes. Nous remarquons que la méthode ConvVectors donne les meilleurs résultats. De plus, nous remarquons qu'on peut avoir le plus faible taux d'erreur juste en utilisant 5 secondes de segment de test.

Tableau 8.2: Présentation des résultats (en % EER) des méthodes suivantes : *i*-vector/PLDA, DeepSF/NearC, CNN/UBM-NearC, et la méthode proposée ConvVectors, en utilisant le corpus THUYG, dans trois conditions de SNR : nette, 9db, et 0db.(Le taux d'erreur qui prend en considération à la fois les locuteurs masculins et féminins est calculé selon l'équation suivante : $EER = \frac{(N_m \times EER_m)+(N_f \times EER_f)}{N_m+N_f}$, avec N, m et f font référence respectivement aux nombre total de locuteurs, masculin, et féminin.)

Locuteurs		Méthodes			
Cible	Test	*i*-vector/PLDA	DeepSF/NearC	CNN/UBM-NearC	ConvVectors
Nette	Nette	2.26	0.48	0.37	**0.13**
Nette	9db	4.31	0.99	0.82	**0.30**
Nette	0db	8.37	1.17	1.05	**0.42**
9db	Nette	3.97	**0.90**	1.02	1.08
9db	9db	3.69	1.45	1.51	**0.97**
9db	0db	5.58	2.68	2.97	**1.71**
0db	Nette	6.33	3.12	3.47	**1.62**
0db	9db	4.77	2.90	2.91	**1.72**
0db	0db	5.26	2.33	2.40	**1.44**
AVG EER		4.95	1.78	1.84	**1.04**

Les résultats des expériences montrent que la méthode proposée bénéficie d'un soutien évident. En outre, une autre découverte originale est que les filtres cibles pourraient former les caractéristiques spécifiques du locuteur en améliorant les caractéristiques communes, qui sont illustrées par les matrices UBM.

Tableau 8.3: Présentation des résultats (en % EER) des méthode: *i*-vector/PLDA, DeepSF/NearC, CNN/UBM-NearC et la méthode proposée ConvVectors en utilisant le corpus THUYG avec différentes durées de segments de test, et dans des conditions de paroles nettes.

Durée (seconds)	1	2	3	4	5	6	7	8	9	10
i-vector/PLDA	7.72	5.23	4.77	4.32	4.02	3.15	2.25	2.28	2.26	2.26
DeepSF/NearC	0.82	0.80	0.79	0.58	0.60	0.55	0.56	0.58	0.52	0.48
CNN/UBM-NearC	1.04	0.97	0.82	0.45	0.39	0.41	0.41	0.37	0.36	0.37
ConvVectors	-	-	0.48	0.16	0.13	0.14	0.13	0.13	0.13	0.13

8.5 Conclusion

Dans ce chapitre, nous avons démontré que les CNN pourraient être utilisées directement dans la reconnaissance automatique du locuteur sans réduire le problème

de la RAL à celui de la reconnaissance d'images. Le défi vient du besoin de données bidimensionnelles qui, dans le cas de la parole, a déjà été abordé en présentant le signal de parole sous forme d'image en utilisant soit des spectrogrammes soit des formes d'onde, et en résolvant ainsi un problème de reconnaissance d'images. Nous avons proposé plutôt des filtres CNN bidimensionnels pour les locuteurs, dérivés des MFCC. De plus, nous avons proposé un nouveau type de vecteurs pour identifier les locuteurs, appelés convVectors.

Les expériences ont été menées sur le corpus THUYG, dans trois conditions de bruit : nette, 9db et 0db. Dans l'ensemble, nos résultats montrent une efficacité des filtres CNN proposés sur la préservation des caractéristiques du locuteur dans les convVectors en améliorant les performances par rapport aux méthodes de traditionnelles pour la RAL. L'amélioration rapportée du système de base est de 43,10%. Comme les phases d'apprentissage et de test prennent du temps, notre infrastructure de calcul ne nous a pas permis d'utiliser des ensembles de données plus importants. Des études futures avec une infrastructure de calcul plus puissante pourraient être nécessaires pour valider les conclusions qui peuvent être tirées de ce travail.

Partie IV

Conclusion

9 Conclusions et Perspectives

Sommaire

9.1 Résultats de la thèse et discussion

Le tableau 9.1 présente un récapitulatif des résultats des méthodes proposées dans ce travail de thèse : MFCC/NearC (chapitre 5), DeepSF/NearC (chapitre 6), CNN/UBM-NearC (chapitre 7) et ConvVectors (chapitre 8). En plus des méthodes traditionnelles: GMM/UBM, et *i*-vector/PLDA. En utilisant trois conditions de bruit de SNR : nette, 9db et 0db. Sur le corpus THUYG.

Dans le chapitre 5, nous avons proposé une nouvelle méthode d'évaluation de mesure de similarité pour la vérification automatique du locuteur (MFCC/NearC). Le but de cette méthode est de cibler les caractéristiques spécifiques des locuteurs. Afin d'évaluer les locuteurs dans le système de vérification, nous calculons la distance entre le locuteur de test et l'ensemble des vecteurs que nous avons considéré comme contenant des caractéristiques spécifiques du locuteur cible. Les résultats ont confirmé notre hypothèse initiale, car les expériences expérimentales ont enregistré un taux d'erreur de 0,88% EER dans des conditions de parole nette. Cependant, les résultats dans des conditions de bruit sont loin de ce qui a été enregistré par les méthodes traditionnelles, ce qui est évident puisque la méthode MFCC/NearC est

Tableau 9.1: Présentation des résultats (en % EER) des méthodes suivantes : GMM/UBM, *i*-vector/PLDA, MFCC/NearC(chapitre 5), DeepSF/NearC (chapitre 6), CNN/UBM-NearC (chapitre 7), et les ConvVectors (chapitre 8), en utilisant le corpus THUYG, dans trois conditions de SNR : Nette, 9db, et 0db.

Locuteurs		Méthodes					
Cible (SNR)	Test (SNR)	GMM/UBM	*i*-vector/PLDA	MFCC/NearC	DeepSF/NearC	CNN/UBM-NearC	ConvVectors
Nette	Nette	4.25	2.26	0.88	0.48	0.37	**0.13**
Nette	9db	12.74	4.31	10.70	0.99	0.82	**0.30**
Nette	0db	21.36	8.37	15.64	1.17	1.05	**0.42**
9db	Nette	8.66	3.97	9.51	**0.90**	1.02	1.08
9db	9db	5.87	3.69	5.10	1.45	1.51	**0.97**
9db	0db	8.01	5.58	13.69	2.68	2.97	**1.71**
0db	Nette	12.26	6.33	17.82	3.12	3.47	**1.62**
0db	9db	8.30	4.77	15.21	2.90	2.91	**1.72**
0db	0db	9.71	5.26	9.94	2.33	2.40	**1.44**
AVG EER		10.13	4.95	10.94	1.78	1.84	**1.04**

considérée comme une approches de modèles Template. La méthode MFCC/NearC donne de bons résultats dans des conditions de parole nette mais aussi lorsque le locuteur cible et le locuteur de test sont tous deux dans le même niveau de SNR, ce qui est confirmé en enregistrant un taux d'erreur de 5,10% EER lorsque $SNR = 9db$ et un taux d'erreur de 9,94% EER lorsque $SNR = 0db$. Ces résultats illustrent la propriété des modèles Template où le locuteur de test est considéré comme un modèle incomplet du locuteur cible.

Dans le chapitre 6, nous avons proposé une approche de deep learning pour améliorer la méthode précédente (MFCC/NearC). Le problème de la méthode précédente résidait dans les vecteurs caractéristiques MFCC qui, lorsqu'ils sont corrompus par le bruit, ne préservent pas les caractéristiques spécifiques du locuteur. Pour cette raison, nous avons proposé dans chapitre 6 une amélioration des vecteurs caractéristiques (MFCC) et de les transformer en vecteurs caractéristiques profonds (DeepSF). Par la suite, nous avons utilisé la méthode NearC comme méthode d'évaluation. La méthode proposée DeepSF/NearC légèrement meilleur que la méthode MFCC/NearC (10,94% EER) et les méthodes traditionnelles (GMM/UBM : 10,13% EER, *i*-vector/PLDA : 4,95% EER) dans toutes les conditions nettes et de bruit en enregistrant un taux d'erreur moyen de 1,78% EER. Dans ce chapitre, nous avons confirmé l'hypothèse initiale selon laquelle certains vecteurs caractéristiques représentent les caractéristiques spécifiques du locuteur. De plus, DeepSF/NearC a démontré la puissance de la méthode NearC, en améliorant les vecteurs caractéristiques MFCC du locuteur.

Dans le chapitre 7, nous avons proposé une nouvelle méthode qui permet d'intégrer les réseaux neuronaux convolutifs (CNN) dans la reconnaissance automatique du locuteur. Cependant, le problème, ici, réside dans la nécessité de disposer de données bidimensionnelles pour l'entrée du réseau CNN, ce qui n'est pas possible avec les vecteurs caractéristiques MFCC. Pour cette raison, nous avons proposé une approche (CNN/UBM-NearC) de deep learning qui permet de générer les entrées pour le réseau CNN. Les résultats expérimentaux montrent que la méthode CNN/UBM-NearC ne surpasse la méthode DeepSF/NearC que lorsque la parole des locuteurs cibles est nette. Mais globalement, la méthode DeepSF/NearC surpasse la méthode CNN/UBM-NearC en enregistrant un taux d'erreur moyen de 1,78% EER alors que la méthode CNN/UBM-NearC a enregistré un taux d'erreur moyen de 1,84% EER.

Dans le chapitre 8, nous suivons la même voie que le chapitre 7, qui est l'intégration des CNN dans la reconnaissance automatique du locuteur. Dans le chapitre précédent, nous avons développé les données d'entrée pour le réseau CNN.

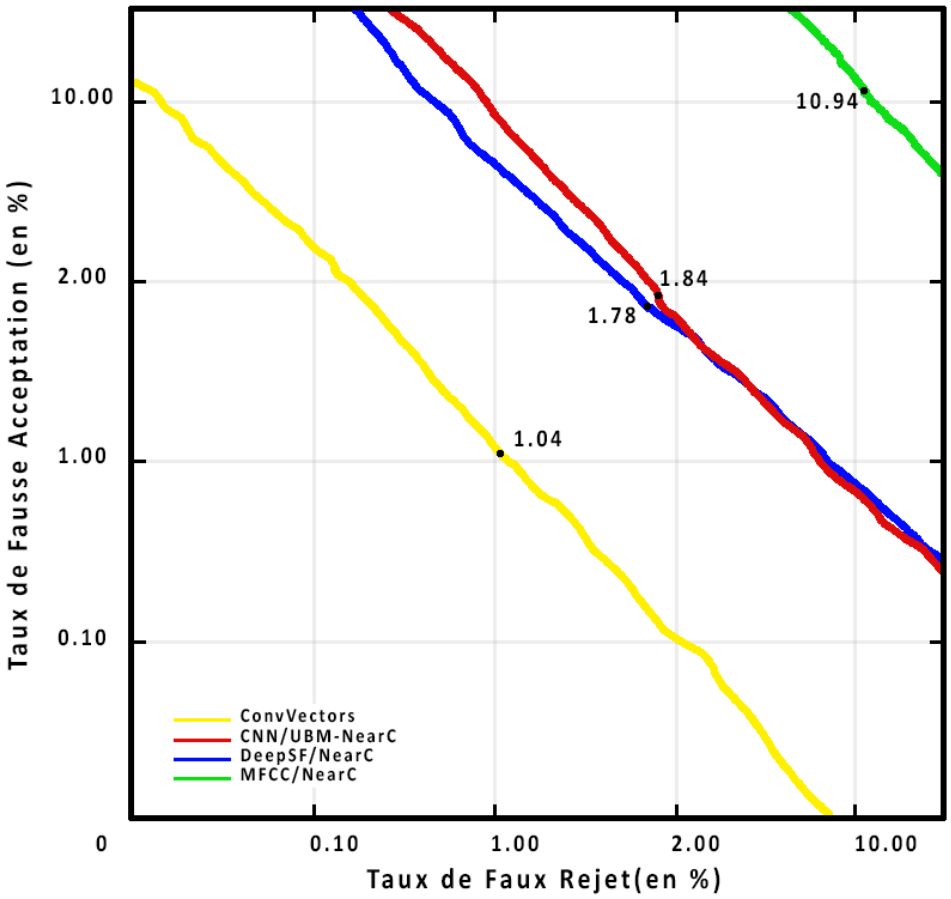


Figure 9.1: Le compromis d'erreur de détection d'échantillon (DET) représentant la performance des méthodes proposées: MFCC/NearC, DeepSF/NearC, CNN/UBM-NearC et ConvVectors.

Dans ce chapitre, nous avons proposé une nouvelle configuration du réseau CNN conçue pour le locuteur cible, afin d'extraire des vecteurs d'identification appelés convVectors. Les expériences montrent que la méthode convVectors a surpassé toutes les méthodes proposées et les méthodes traditionnelles en enregistrant le taux d'erreur le plus faible 1,04% EER. De plus, en parole nette, la méthode a enregistré un taux d'erreur de 0,13% EER. En outre, la méthode DeepSF/NearC conserve l'erreur minimale lorsque le SNR du locuteur cible est de 9db et que la parole du locuteur de test est nette, en enregistrant une erreur moyenne de 0,90% EER, alors que la méthode convVectors avait enregistré un taux d'erreur de 1,08% EER.

Le tableau 9.2 montre les résultats de ce protocole expérimental de toutes les méthodes étudiées dans cette thèse dans des conditions de parole nette Pour les méthodes traditionnelles (GMM/UBM et *i*-vector/PLDA), on constate que le taux d'erreur diminue nettement de la première seconde à la dixième seconde. En effet,

Tableau 9.2: Présentation des résultats (en % EER) des méthodes suivantes : GMM/UBM, *i*-vector/PLDA, MFCC/NearC(chapitre 5), DeepSF/NearC (chapitre 6), CNN/UBM-NearC (chapitre 7), et les ConvVectors (chapitre 8)en utilisant le corpus THUYG avec différentes durées de segments de test, et dans des conditions de paroles nettes.

Durée (seconds)	1	2	3	4	5	6	7	8	9	10
GMM/UBM	14.28	12.54	12.12	9.49	8.15	7.98	5.62	4.32	4.31	4.25
i-vector/PLDA	7.72	5.23	4.77	4.32	4.02	3.15	2.25	2.28	2.26	2.26
MFCC/NearC	1.16	1.20	1.21	0.99	1.02	0.98	0.99	1.03	0.98	0.88
DeepSF/NearC	0.82	0.80	0.79	0.58	0.60	0.55	0.56	0.58	0.52	0.48
CNN/UBM-NearC	1.04	0.97	0.82	0.45	0.39	0.41	0.41	0.37	0.36	0.37
ConvVectors	-	-	0.48	0.16	0.13	0.14	0.13	0.13	0.13	0.13

la marge de diminution du taux d'erreur pour GMM/UBM est de 10,03% EER et de 5,46% EER pour l'*i*-vector/PLDA. En revanche, on constate que les marges d'erreur de la méthode MFCC/NearC (Chapitre 5) et de la méthode DeepSF/NearC (Chapitre 6) ne sont pas importantes par rapport aux marges d'erreur des méthodes traditionnelles. En effet, les marges d'erreur sont de 0,33% EER et 0,34%EER pour les méthodes MFCC/NearC et DeepSF/NearC respectivement. Cela montre que ces méthodes sont efficaces même avec des durées de segment de test plus courtes.

Pour les méthodes CNN/UBM-NearC (Chapitre 7) et convVectors (Chapitre 8), on remarque qu'à partir d'une durée de segment de test de 5 secondes, les deux méthodes atteignent leur taux d'erreur le plus faible. Pour les convVectors, et en raison de leur architecture, nous ne pouvons pas avoir des segments de test de moins de 3 secondes.

9.2 Conclusion générale

L'objectif principal de ce travail de recherche était de développer des architectures de deep learning spécifiques au domaine de la reconnaissance automatique du locuteur. Dans l'introduction de cette thèse, nous avons soulevé des questions de recherche auxquelles nous répondrions comme suit:

1. *L'extraction de caractéristiques est la première phase d'un système de reconnaissance automatique du locuteur; cette phase consiste à représenter une petite durée de signal vocal par un vecteur appelé vecteur caractéristique. Le problème qui apparaît est lorsque plusieurs locuteurs ont des vecteurs caractéristiques similaires, par exemple le vecteur caractéristique qui représente le silence. Les questions que l'on peut se poser sont les suivantes: Pourrait-on extraire les*

caractéristiques spécifiques du locuteur ? Pourrait-on utiliser la distribution et la répartition des vecteurs caractéristiques communs dans les vecteurs caractéristiques du locuteur comme une caractéristique ?

➥ Dans le chapitre 5, nous avons regroupé les vecteurs caractéristiques du locuteur cible en 23 clusters. Au cours du processus de vérification du locuteur, nous avons considéré la distance entre le locuteur de test et le locuteur cible comme étant la distance entre le locuteur de test et le cluster le plus proche du locuteur cible. Les résultats dans des conditions de parole nette suggèrent que le cluster le plus proche peut contenir les caractéristiques spécifiques du locuteur. De plus, les résultats insatisfaisants obtenus dans des conditions de bruit ont renforcé l'hypothèse. En effet, les vecteurs caractéristiques corrompues ont dû être améliorés par un modèle de deep learning afin de découvrir les caractéristiques spécifiques du locuteur (chapitre 6). De plus, les vecteurs caractéristiques du locuteur sont divisés en deux groupes par un algorithme de regroupement, et un modèle de deep learning est établi pour les distinguer. Ce que nous comprenons comme une nouvelle caractéristique du locuteur. En effet, nous pensons que lorsqu'un modèle de deep learning est entraîné à distinguer deux ensembles de vecteurs caractéristiques, il est en fait entraîné à apprendre comment les vecteurs caractéristiques sont distribués dans les deux ensembles.

2. *La vérification automatique du locuteur est une branche de la reconnaissance automatique du locuteur qui consiste à comparer un segment de parole à un modèle de locuteur. Les approches traditionnelles utilisent le modèle dit* modèle générique des locuteurs *(*UBM: Universal Background Model*) afin de modéliser une large population de locuteurs, puis d'en déduire le modèle de locuteur cible. Ensuite, le segment de test est comparé au modèle UBM et au modèle du locuteur, et voir si le segment et proche du modèle de locuteur ou bien du modèle UBM. Le problème ici est que nous avons besoin d'une énorme quantité de données pour modéliser les locuteurs. Les questions qui se posent sont les suivantes : serait-il possible de modéliser les locuteurs sans utiliser une énorme quantité de données ? Pourrait-on modéliser les locuteurs sans s'appuyer sur un modèle de générique des locuteurs ?*

   ➥ Dans le chapitre 5, nous avons utilisé une méthode pour évaluer les locuteurs directement à partir de vecteurs caractéristiques. La méthode fonctionne bien dans des conditions de parole nette. Cependant, les vecteurs caractéristiques ont dû être améliorés à l'aide d'un modèle de deep learning (chapitre 6) pour

pouvoir fonctionner également dans des conditions de bruit. Dans les deux chapitres, nous avons démontré que nous pouvons modéliser les locuteurs sans utiliser modèle générique des locuteurs. Dans la tâche de vérification automatique du locuteur, nous n'avons besoin que d'un seul segment de parole du locuteur pour construire son modèle.

3. *L'objectif de la thèse est d'intégrer des modèles de deep learning dans la reconnaissance automatique du locuteur. L'un des modèles les plus populaires est celui des réseaux de neurones convolutifs (CNN). Ils ont été utilisés dans la reconnaissance d'images et sont considérés comme la méthode de pointe. Cependant, la nature des réseaux CNN, qui ne sont conçus que pour des données bidimensionnelles, rend difficile leur intégration dans plusieurs domaines de la reconnaissance des formes. Les réseaux CNN ont été intégrés dans la reconnaissance automatique de la parole, mais en se reposant sur des spectrogrammes et des formes d'onde, qui sont une représentation de la parole sous forme d'image. Les questions que nous pouvons poser sont les suivantes: pourrions-nous intégrer les réseaux CNN dans la reconnaissance automatique du locuteur sans utiliser d'images ? Si oui, comment construire les données bidimensionnelles ? Pourrions-nous représenter les caractéristiques spécifiques du locuteur en utilisant des réseaux CNN ?*

   ➥ Dans le chapitre 7, nous avons proposé une méthode pour construire des données bidimensionnelles pour un réseau de neurones convolutif. En effet, nous avons utilisé des locuteurs UBM et les vecteurs caractéristiques des locuteurs pour construire deux ensembles opposés de matrices pour la tâche de vérification. Par conséquent, nous pourrions utiliser le réseau CNN sans utiliser de spectrogrammes ou de formes d'onde. De plus, dans le chapitre 8, nous avons adapté le réseau CNN pour le locuteur cible, en utilisant ses vecteurs caractéristiques comme matière première pour extraire les filtres (noyaux) du réseau CNN. Les résultats ont démontré que les filtres ont représenté les caractéristiques spécifiques du locuteur en donnant les meilleurs résultats tout au long de la thèse.

Ce travail de recherche a nécessité une énorme réflexion et de travail acharné, dans lequel nous avons essayé plusieurs méthodes de plus que ce qui a été présenté dans cette thèse. Nous avons commencé à travailler sur ce sujet depuis janvier 2016, en tant que projet de fin d'étude. Au début, nous avons été confrontés à plusieurs problèmes comme le manque de corpus de reconnaissance des locuteurs gratuits. En outre, le deep learning n'a pas été exploré dans la reconnaissance automatique du

locuteur comme aujourd'hui. Pour cette raison, ce travail est considéré comme l'une des contributions importantes dans le domaine de la reconnaissance automatique du locuteur. Le travail le plus intéressant que nous avons fait dans le cadre de cette thèse est que nous avons pu extraire les caractéristiques du locuteur les plus pertinentes à partir de son signal continu de parole.

9.3 Perspectives

Nos travaux de recherche ont donné de nombreux résultats prometteurs qui ont plusieurs implications pratiques dans différents domaines, et encouragent la poursuite de certaines perspectives intéressantes, parmi lesquelles nous suggérons:

- Comme la méthode NearC/MFCC donne de bons résultats dans des conditions de parole nette, et ne nécessite pas beaucoup de données et de temps de traitement, nous suggérons de l'utiliser pour les applications mobiles d'accès vocal ;
- Pour les applications de sécurité qui ne disposent pas d'ordinateurs à haute capacité de calcul, nous suggérons l'utilisation de la méthode DeepSF/NearC, car elle présente un taux d'erreur plus faible et ne nécessite pas beaucoup de données pour la phase de d'entraînement ;
- Pour les applications de sécurité qui disposent d'ordinateurs puissants, nous suggérons l'utilisation de la méthode convVectors, car elle présente le taux d'erreur le plus faible et nécessite beaucoup plus de données et de temps pour la phase d'entraînement.

Des études futures avec une infrastructure de calcul plus puissante peut être nécessaire pour valider les conclusions qui peuvent être tirées de ce travail. De plus, nous voulons tester les résultats du dernier chapitre sur un corpus plus large et de procéder à des évaluations internationales telles que les évaluations du NIST[1]. Nous avions aussi l'intention de transformer les convVectors en vecteurs plus sophistiqués.

[1] https://sre.nist.gov/

Annexes

A

Perceptron et MLP : Algorithmes d'Entraînement

Sommaire

A.1 Entraînement d'un réseau de neurones monocouche: la règle Delta

La règle delta pour les réseaux de neurones monocouches est une méthode de descente par gradient, qui utilise la dérivée des poids du réseau par rapport à l'erreur de sortie pour ajuster les poids afin de mieux classer les exemples d'entraînement.

L'entraînement est effectuée sur un ensemble de données d'entraînement X, où chaque échantillon $x^n \in X$ est un vecteur $x^n = (x_0^n, \ldots, x_N^n)$. Supposons que pour un échantillon d'entraînement donné x^n, le $i^{ème}$ neurone de notre réseau de neurones monocouche produise y_j^n, la sortie cible (souhaitée) t_j^n et les poids

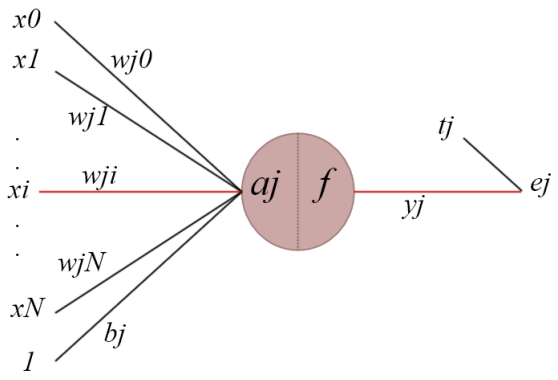


Figure A.1: Illustration détaillée d'un réseau de neurones monocouche pouvant être entraîné avec la règle du delta.

$w = (w_j^0, \ldots, w_j^M)$, comme indiqué dans la figure A.1. Nous pouvons considérer le biais comme un poids supplémentaire avec une entrée unitaire, et donc nous pouvons omettre le biais explicite de la dérivation.

Nous voulons savoir comment changer un poids donné w_{ji} étant donné la sortie du nœud j pour un échantillon de données d'entrée donné x^n,

$$y_j^n = f(a_j^n), \tag{A.1}$$

où l'activation a_j^n est définie par,

$$a_j^n = \sum_i w_{ji} x_i^n. \tag{A.2}$$

Pour ce faire, nous devons utiliser l'erreur de la prédiction pour chaque sortie y_j et l'échantillon d'entraînement x^n par rapport à l'étiquette connue t_j,

$$e_j^n = y_j^n - t_j^n. \tag{A.3}$$

Pour cette dérivation, nous supposons que l'erreur pour un échantillon unique est calculée par la somme des erreurs au carré de chaque sortie.

$$E^n = \frac{1}{2} \sum_j (e_j^n)^2. \tag{A.4}$$

La règle de dérivation des fonctions composées nous permet de calculer la sensibilité de l'erreur à chaque poids w_{ji} dans le réseau„

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial e_j^n} \frac{\partial e_j^n}{\partial y_j^n} \frac{\partial y_j^n}{\partial a_j^n} \frac{\partial a_j^n}{\partial wji} \tag{A.5}$$

en dérivant Eq. A.4 par rapport à e_j^n,

$$\frac{\partial E^n}{\partial e_j^n} = e_j^n, \tag{A.6}$$

Eq. A.3 par rapport à y_j^n,

$$\frac{\partial e_j^n}{\partial y_j^n} = 1, \tag{A.7}$$

Eq. A.1 par rapport à a_j^n,

$$\frac{\partial y_j^n}{\partial a_j^n} = f'(a_j^n), \tag{A.8}$$

et enfin Eq. A.2 par rapport w_{ji},

$$\frac{\partial a_j^n}{\partial wji} = \frac{\partial}{\partial w_{ji}}(\sum_i w_{ji}x_i) = x_i, \tag{A.9}$$

puisqu'un seul des termes de la somme est lié au poids spécifique w_{ji}. La sensibilité est donc,

$$\frac{\partial E^n}{\partial w_{ji}} = e_j^n f'(a_j^n)x_i. \tag{A.10}$$

En général, on définit alors ce que l'on appelle le gradient local, l'erreur, ou simplement *delta*,

$$\delta_j^n = \frac{\partial E^n}{\partial a_j^n} = \frac{\partial E^n}{\partial e_j^n}\frac{\partial e_j^n}{\partial y_j^n}\frac{\partial y_j^n}{\partial a_j^n} = e_j^n f'(a_j^n), \tag{A.11}$$

de telle sorte que Eq. A.10 puisse être réécrite,

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j^n x_i. \tag{A.12}$$

La règle du delta ajuste chaque poids w_{ji} proportionnellement à la sensibilité,

$$\Delta w_{ji} = -\gamma\frac{\partial E^n}{\partial w_{ji}}, \tag{A.13}$$

où γ est une constante appelée *taux d'apprentissage*. En utilisant le delta défini dans Eq. A.11, cela s'écrit simplement,

$$\Delta w_{ji} = -\gamma\delta_j^n x_i. \tag{A.14}$$

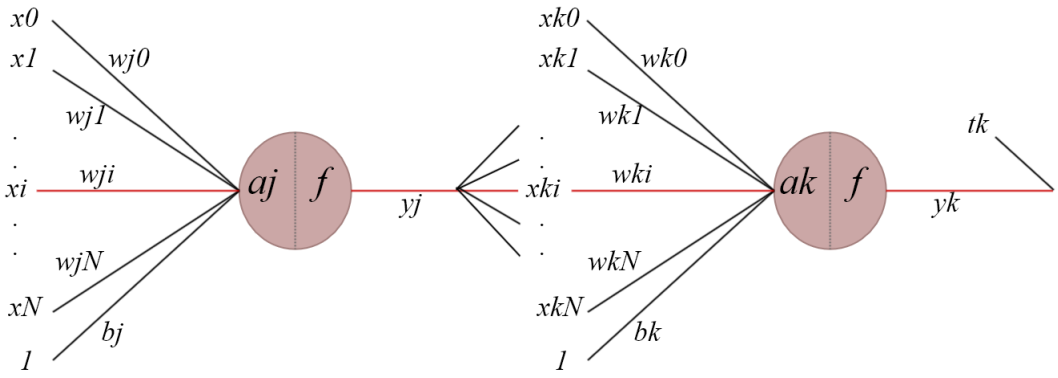


Figure A.2: Illustration détaillée d'un réseau de neurones avec une seule couche cachée. Le $k^{ème}$ neurone est connecté au $j^{ème}$ neurone de la couche précédente.

A.1.1 Rétro-propagation

Bien que la règle delta (Section A.1) fonctionne, elle est limitée dans ce qu'elle peut réaliser par l'architecture à couche unique dans laquelle elle fonctionne. On sait depuis longtemps que les réseaux généraux d'unités, tels que les MLP, fournissent une capacité de calcul beaucoup plus riche, mais l'absence de règles d'entraînement applicables n'a fait de ces réseaux qu'une simple curiosité intellectuelle. Cela a changé avec la découverte de la rétro-propagation (également connue sous le nom de règle du delta généralisée), permettant l'entraînement dans les MLP. La question de savoir qui a "découvert" la rétro-propagation est quelque peu controversée, puisqu'il s'agit essentiellement de l'application de la règle dérivation des fonctions composées aux réseaux de neurones, mais il est généralement admis qu'elle a d'abord été démontrée expérimentalement par [60]. Malgré qu'il s'agisse "uniquement de la règle de dérivation des fonctions composées", écarter cette première démonstration de la rétro-propagation dans les réseaux de neurones revient à sous-estimer l'importance de cette découverte pour le domaine, et à écarter les difficultés pratiques de la première mise en œuvre de l'algorithme — un fait qui sera attesté par tous ceux qui ont essayé depuis.

Ce qui suit est une dérivation de la rétro-propagation basée sur les références [56, 121], mais avec une notation différente. Cette dérivation s'appuie sur la dérivation de la règle delta dans la section A.1, bien qu'il soit important de noter que, comme indiqué dans la figure A.2, l'indexation que nous utiliserons pour faire référence aux neurones des différentes couches diffère de celle de la figure A.1 pour le cas d'un réseau monocouche.

Nous cherchons à déterminer la sensibilité de l'erreur E à un poids donné w_{ij} dans le réseau. Il existe deux classes de poids pour lesquels nous devons dériver des règles

différentes, **(i)** ceux appartenant aux *neurones de la couche de sortie*, c'est-à-dire les neurones situés directement avant la sortie, comme w_{kj} dans la figure A.1, et **(ii)** les poids appartenant aux neurones de la couche cachée, comme w_{ji} dans la figure A.2.

(i) Couche de sortie: les poids de sortie sont relativement faciles à trouver, car ils correspondent aux mêmes types de poids que ceux des réseaux monocouches et ont un accès direct au signal d'erreur, c'est-à-dire e_j^n. En effet, la dérivation de la section A.1 décrit également la sensibilité des poids dans la couche de sortie d'un MLP. Avec un certain changement de notation (maintenant l'indexation par k plutôt que j pour correspondre à la figure A.2), nous pouvons utiliser la sensibilité trouvée dans Eq. A.10.

$$\frac{\partial E^n}{\partial w_{kj}} = \frac{\partial E^n}{\partial a_k^n}\frac{\partial a_k^n}{\partial w_{kj}} = \delta_j^n x_{kj}^n = \delta_k^n y_j^n \tag{A.15}$$

(ii) Couche cachée: nous allons d'abord dériver la dérivée partielle $\partial E^n/\delta w_{ji}$, pour un réseau à couche cachée unique, comme celui illustré dans la figure A.2. Contrairement au premier cas, les poids appartenant aux neurones cachés n'ont pas d'accès direct au signal d'erreur, mais nous devons calculer le signal d'erreur à partir de tous les neurones qui relient indirectement le neurone à l'erreur (c'est-à-dire chaque neurone de sortie y_k).

En suivant la règle de dérivation des fonctions composées, nous pouvons écrire la dérivée partielle d'un poids caché w_{ji} par rapport à l'erreur E^n,

$$\frac{\partial E^n}{\partial w_{ji}} = \underbrace{\frac{\partial E^n}{\partial e_k^n}\frac{\partial e_k^n}{\partial y_k^n}\frac{\partial y_k^n}{\partial a_k^n}\frac{\partial a_k^n}{\partial y_j^n}}_{neurones\,de\,sortie}\underbrace{\frac{\partial y_j^n}{\partial a_j^n}\frac{\partial a_j^n}{\partial w_{ji}}}_{neurones\,cachés} \tag{A.16}$$

où la somme provient du fait que, contrairement à Eq. A.9 où le poids w_{kj} n'affecte qu'une seule sortie, le poids caché w_{ji} affecte tous les neurones de la couche suivante (voir figure A.2).

Nous savons déjà comment calculer les partiels pour la couche de sortie à partir de la dérivation de la règle du delta pour les réseaux monocouche, et nous pouvons les substituer à partir de Eq. A.15 pour les neurones de sortie et les dérivés partiels d'erreur,

$$\frac{\partial E^n}{\partial w_{ji}} = (\sum_k \delta_k^n y_j^n \frac{\partial a_k^n}{\partial y_j^n})\frac{\partial y_j^n}{\partial a_j^n}\frac{\partial a_j^n}{\partial w_{ji}} \tag{A.17}$$

Rappelons que dans Eq. A.2, l'activation du réseau a est une somme de tous les poids des couches précédentes. Ainsi,

$$\frac{\partial a_k^n}{\partial y_j^n} = \frac{\partial}{\partial y_j^n}(\sum_j w_{kj} y_j^n) = w_{kj} \tag{A.18}$$

et en substituant de Eq. A.8 et Eq. A.9 à Eq. A.17,

$$\frac{\partial E^n}{\partial w_{ji}} = (\sum_k \delta_k^n y_j^n w_{kj}) f'(a_j^n) x_i. \tag{A.19}$$

Cela ressemble un peu à l'expression dérivée pour une couche unique, et tout comme dans Eq. A.11, nous pouvons utiliser notre définition du delta pour la simplifier. Pour les couches cachées, cela s'évalue comme,

$$\delta_j^n = \frac{\partial E^n}{\partial a_k^n} = (\sum_k \frac{\partial E^n}{\partial e_k^n} \frac{\partial e_k^n}{\partial y_k^n}) \frac{\delta y_k^n}{\delta a_k^n} = (\sum_k \delta_k^n y_j^n w_{kj}) f'(a_j^n). \tag{A.20}$$

Cela nous laisse avec l'expression plus commode,

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j^n x_i. \tag{A.21}$$

La dérivation ci-dessus était basée sur le cas spécifique d'un réseau à une seule couche cachée, mais il est trivial d'étendre ce résultat à plusieurs couches cachées. Il y a une récursion dans le calcul des dérivées partielles dans Eq. A.20 qui vaut pour un réseau avec un nombre quelconque de couches cachées.

Le delta est défini,

$$\delta_i^n = \begin{cases} f'(a_j^n) e_j^n & \text{lorsque } j \text{ est un neurone de la couche de sortie} \\ f'(a_j^n)(\sum_j \delta_j^n y_i^n w_{ji}) & \text{lorsque } j \text{ est un neurone de la couche cachée,} \end{cases} \tag{A.22}$$

pour toute couche de réseau de neurones adjacente i, j, y compris la couche de sortie où les sorties sont considérées comme ayant un indice j. La sensibilité est alors,

$$\frac{\partial E^n}{\partial w_{ji}} = \delta_j^n y_i. \tag{A.23}$$

A.2 Entraînement avec la rétro-propagation

L'entraînement avec rétro-propagation ressemble beaucoup à la règle du delta ; les sensibilités sont utilisées pour corriger les poids proportionnels à un taux d'apprentissage constant ou à un paramètre de taille de pas γ.

Bien que la correction soit proportionnelle à la sensibilité, nous souhaitons réduire l'erreur E^n, et nous déplaçons donc le poids dans la direction opposée du gradient. Notons que, plutôt que d'optimiser directement la fonction d'erreur, on utilise généralement une perte de substitution qui est plus facile à optimiser. Formellement, la règle de changement de poids est donnée par,

$$\Delta w_{ij}^n = -\gamma \frac{\partial E^n}{\partial w_{ji}} = -\gamma \delta_j^n y_i, \tag{A.24}$$

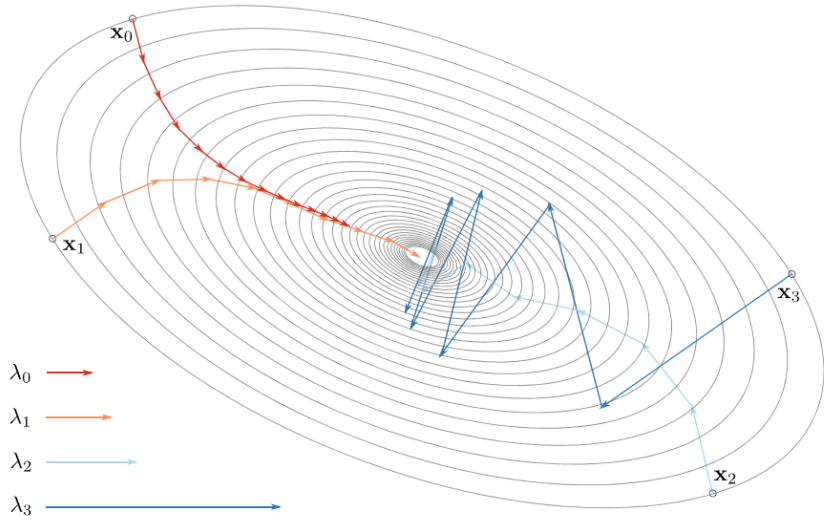


Figure A.3: Une illustration de l'effet du taux d'apprentissage et de la politique d'entraînement sur la convergence avec la rétro-propagation. Cet exemple est celui d'une surface d'erreur 2D symétrique, où les paramètres sont initialisés à l'un des points de la surface symétriquement identiques x_i, où $i = 0 \dots 4$. Pour chacun des différents taux d'apprentissage initiaux γ_i, le taux d'apprentissage est diminué de 10% à chaque itération.

où δ_j^n est tel que défini dans Eq. A.22, et y_i est la sortie du neurone i. La rétro-propagation est une méthode de descente la plus rapide. Ceci est illustré dans la figure A.3, où la règle d'entraînement de la rétro-propagation, Eq. A.24, spécifie une taille de pas sous la forme de taux d'apprentissage. Le paramètre de taux d'apprentissage met à l'échelle la taille de pas, ou l'ampleur du vecteur de changement de poids. La figure A.3 illustre également l'effet de la vitesse d'apprentissage sur la descente du gradient. Un taux d'apprentissage trop faible peut entraîner un apprentissage très lent, comme pour γ_0, tandis qu'une taille de pas trop importante peut entraîner un rebondissement autour des minima (γ_2, γ_3), ou le fait de les manquer complètement.

Afin de s'installer dans un minima local, le taux d'apprentissage doit également être réduit au fur et à mesure de la progression de l'entraînement. Cependant, un taux de diminution trop rapide peut ne jamais atteindre le bassin d'attraction des minima locaux, comme avec γ_0, tandis que si le taux de diminution est trop lent, il faudra beaucoup de temps pour entrer dans le bassin d'attraction, comme avec γ_3.

L'équilibre entre la recherche d'un taux d'apprentissage approprié et une politique d'entraînement fait malheureusement partie de la "magie noire" de l'entraînement des DNN qui découle de l'expérience, mais [63, 122] sont d'excellentes références sur certaines des approches communes adoptées pour simplifier cette tâche.

A.2.1 Problème de l'optimisation du premier ordre

La raison pour laquelle le taux d'apprentissage et la politique d'entraînement ont un effet aussi important repose sur le fait que la descente de gradient est une méthode d'optimisation du premier ordre, et ne prend en compte que les dérivées partielles du premier ordre, c'est-à-dire que pour une surface d'erreur 2D $E(x, y)$, la descente de gradient se déplace dans la direction opposée au gradient,

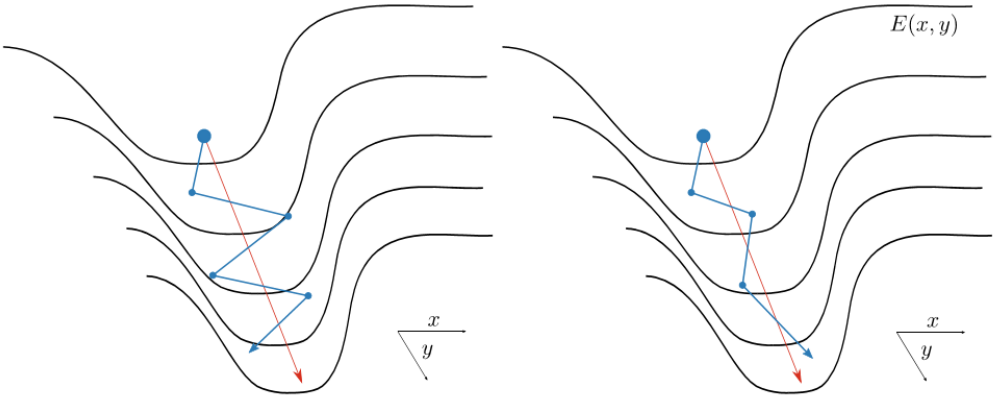
$$\nabla E(x, y) = (\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}). \tag{A.25}$$

Ce gradient nous indique la direction de la croissance maximale en un point donné de la surface d'erreur, mais il ne nous donne aucune information sur la courbure de la surface à ce point. La courbure de la surface est décrite par des dérivées d'ordre supérieur telles que les dérivées partielles du second ordre, par exemple $\frac{\partial^2 E}{\partial x^2}$, et les dérivées partielles mixtes, par exemple $\frac{\partial^2 E}{\partial x^2 \partial y^2}$. Ces partielles du second ordre donnent des informations importantes sur la courbure de la surface d'erreur E. Par exemple, dans la figure A.3, la surface d'erreur prend une forme elliptique, ce qui pose des problèmes lorsque l'on considère uniquement la direction de la décroissance maximale $-\nabla E$. L'exemple classique d'une telle surface d'erreur pathologique pour les méthodes du premier ordre est une surface d'erreur qui ressemble à une vallée étroite, comme le montre la figure A.4 (a). Avec une initialisation à l'extérieur du fond de la vallée, la descente de la pente rebondit le long des parois de la vallée, ce qui entraîne une convergence d'apprentissage très lente.

Pour les surfaces bien comportées où l'échelle des paramètres est similaire, les bassins d'attraction autour d'un minimum sont à peu près circulaires, et évitent donc ce problème, puisque les gradients de premier ordre pointeront presque directement sur les minima pour tout endroit de la surface d'erreur.

Il existe des méthodes d'optimisation de second ordre basées sur la méthode de Newton, mais le problème est qu'elles ne s'adaptent pas à la taille d'un DNN pratique. La matrice des dérivés partiels du second ordre pour une fonction à valeurs scalaires, le Hessien $\mathbf{H}$, est nécessaire pour toute méthode d'optimisation complète du second ordre, cependant le Hessien est carré dans le nombre de paramètres du réseau. Pour les réseaux de millions de paramètres, cela signifie que le stockage du Hessien est impossible.

Il existe toute une série d'astuces d'optimisation pour la descente de gradient, en essayant souvent de compenser les défauts de l'optimisation de premier ordre sans utiliser le Hessien, ou en utilisant une approximation de celui-ci. Un historique complet des problèmes d'optimisation dans les DNN est hors de la portée de cette



(a) Gradient descent (b) Momentum

Figure A.4: Courbure pathologique. Une surface d'erreur $E(x, y)$ présentant une vallée étroite, et le chemin optimal du point de départ jusqu'aux minima indiqués par la flèche rouge. Dans une surface d'erreur pathologique comme celle-ci, les méthodes de premier ordre ne peuvent pas utiliser les informations fournies par le Hessian sur la courbure de la surface pour éviter de rebondir le long des parois de la vallée, ce qui ralentirait la descente. La méthode des moments atténue quelque peu ce problème en amortissant le changement de direction, en préservant les informations sur les gradients précédents, ce qui permet une descente plus rapide.

thèse, cependant les lecteurs intéressés devraient se référer à [63] pour en savoir plus sur ces méthodes, et à [123] pour une excellente introduction aux problèmes de l'optimisation du premier et du second ordre dans les DNN.

A.3 Moment

Une amélioration courante de la descente de gradient est le *moment* [60], une astuce pour minimiser l'effet de la courbure pathologique sur la descente de gradient, qui aide également avec la variance des gradients. Le nom vient de l'analogie de la mise à jour du moment physique ρ pour une particule en mouvement, $\rho = mv$, où nous supposons la masse unitaire, $m = 1$.

Dans le moment, les gradients sur plusieurs itérations sont accumulés en un gradient de vitesse,

$$\begin{aligned} \mathbf{v}_{t+1} &= \alpha \mathbf{v}_t - \gamma \nabla E(w) \\ \Delta \mathbf{w} &= \mathbf{w}_t + \mathbf{v}_{t+1}, \end{aligned} \tag{A.26}$$

où γ est le taux d'apprentissage, t est l'itération, ∇E est le gradient de la surface d'erreur $E(\mathbf{w})$ qui est minimisée, et $\mathbf{w}$ est le vecteur de poids optimisé. Le Moment stocke en effet certaines informations sur les gradients trouvés dans les itérations passées, et les utilise pour amortir l'effet d'un nouveau gradient sur la direction de

la recherche, comme l'illustre la figure A.4 (b). Pour les surfaces d'erreur présentant des courbures pathologiques, cela peut accélérer considérablement l'apprentissage.

A.4 Descente par lots et gradient stochastique

Bien que la règle de changement de poids par rétro-propagation, Eq. A.24, nous indique comment changer les poids à partir d'un seul échantillon d'entraînement x^n, dans la pratique, cette méthode est rarement utilisée. La raison est simplement que les gradients d'un seul échantillon sont trop biaisés, ou bruyants, et qu'ils ne sont pas représentatifs de l'ensemble de données en général ; Δw_{ij}^n n'est qu'une approximation du véritable gradient que nous voulons — il provient d'un seul échantillon, x^n, de l'ensemble de données d'entraînement X.

A.4.1 Gradient de descente par lots

À l'autre extrémité du spectre, il y a l'entraînement par lots où le gradient est calculé sur tous les échantillons de données dans l'ensemble d'entraînement,

$$\Delta w_{ij} = -\gamma \frac{1}{N} \sum_{n=0}^{N} \frac{\partial E^n}{\partial w_{ji}}, \tag{A.27}$$

où N est le nombre d'échantillons d'entraînement en X . L'entraînement par lots nous donne le véritable gradient, mais il est également très coûteux, car il nous oblige à effectuer le passage en avant du réseau sur tous les échantillons d'entraînement pour chaque mise à jour.

A.4.2 Gradient stochastique de descente (SGD)

Au lieu de calculer le gradient sur un seul échantillon d'entraînement, ou sur l'ensemble de données d'entraînement, nous pourrions plutôt utiliser un sous-ensemble (*mini-batch*) important de l'ensemble d'entraînement — un mini lot. Cette approche est appelée *Stochastic Gradient Descent (SGD).*

Lorsque nous utilisons le SGD, nous échantillonnons au hasard (sans remplacement) un sous-ensemble de l'ensemble d'entraînement $X_{mb} \subset X$, de telle sorte que,

$$\Delta w_{ij} = -\gamma \frac{1}{|X_{mb}|} \sum_{\{n|X^n \in X_{mb}\}}^{N} \frac{\partial E^n}{\partial w_{ji}}, \tag{A.28}$$

où la taille du mini-batch $|X_{mb}|$ doit être suffisamment significatif pour représenter les statistiques de la distribution de l'ensemble d'entraînement, c'est-à-dire que pour un problème de classification, le mini-batch doit capturer un nombre significatif de

classes de l'ensemble d'entraînement. L'utilisation d'un mini-batch composé d'un seul échantillon comme indiqué dans Eq. A.24, est un cas particulier de SGD.

Il a été observé dans la pratique que l'ajout de bruit au gradient en utilisant la descente stochastique du gradient aide souvent à la généralisation par rapport à la descente en lot du gradient, peut-être en évitant le sur-apprentissage. Notons que même si nous utilisons le vrai gradient pour l'ensemble des données d'entraînement, l'ensemble d'entraînement X n'est qu'un échantillon de la distribution de la population à laquelle nous voulons que notre réseau se généralise.

Bibliographie

[1] Ayoub Bouziane et al. “An open and free speech corpus for speaker recognition: The fscsr speech corpus”. In: *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*. IEEE. 2016, pp. 1–5.

[2] Soufiane Hourri and Jamal Kharroubi. “A novel scoring method based on distance calculation for similarity measurement in text-independent speaker verification”. In: *Procedia computer science* 148 (2019), pp. 256–265.

[3] Soufiane Hourri and Jamal Kharroubi. “A deep learning approach for speaker recognition”. In: *International Journal of Speech Technology* 23.1 (2019), pp. 123–131.

[4] Soufiane Hourri, Nikola S Nikolov, and Jamal Kharroubi. “A deep learning approach to integrate convolutional neural networks in speaker recognition”. In: *International Journal of Speech Technology* 23 (2020), pp. 615–623.

[5] Soufiane Hourri, Nikola S Nikolov, and Jamal Kharroubi. “Convolutional neural network vectors for speaker recognition”. In: *International Journal of Speech Technology* (2021), pp. 1–12.

[6] Yuji Kijima et al. “Speaker adaptation in large-vocabulary voice recognition”. In: *ICASSP’84. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 9. IEEE. 1984, pp. 405–408.

[7] M Morito et al. “A single-chip speaker independent voice recognition system”. In: *ICASSP’86. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 11. IEEE. 1986, pp. 377–380.

[8] Patricia A Nava and Javin M Taylor. “Speaker independent voice recognition with a fuzzy neural network”. In: *Proceedings of IEEE 5th International Fuzzy Systems*. Vol. 3. IEEE. 1996, pp. 2049–2052.

[9] Rebecca Heyer. “Biometrics Technology Review 2008.” In: (2008).

[10] Homayoon SM Beigi et al. “IBM model-based and frame-by-frame speaker recognition”. In: *Speaker Recognition and its Commercial and Forensic Applications, Avignon, France* 4 (1998).

[11] Stéphane H Maes and Homayoon SM Beigi. “Open sesame! Speech, password or key to secure your door?” In: *Asian Conference on Computer Vision*. Springer. 1998, pp. 531–541.

[12] H Ertan Cetingul et al. “Robust lip-motion features for speaker identification”. In: *Proceedings.(ICASSP’05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.* Vol. 1. IEEE. 2005, pp. I–509.

[13] JT Foote and Harvey F Silverman. "A model distance measure for talker clustering and identification". In: *Proceedings of ICASSP'94. IEEE International Conference on Acoustics, Speech and Signal Processing*. Vol. 1. IEEE. 1994, pp. I–317.

[14] AR Abu-El-Quran et al. "Talker identification using reverberation sensing system". In: *SENSORS, 2007 IEEE*. IEEE. 2007, pp. 970–973.

[15] Younès Bennani and Patrick Gallinari. *A modular connectionist architecture for text-independent talker identification*. Université de Paris-Sud, Centre d'Orsay, Laboratoire de Recherche en . . ., 1991.

[16] Oscar Tosi and O Tosi. *Voice identification: theory and legal applications*. University Park Press Baltimore, 1979.

[17] Lawrence George Kersta. "Voiceprint identification". In: *Nature* 196.4861 (1962), pp. 1253–1257.

[18] James L Flanagan. *Speech analysis synthesis and perception*. Vol. 3. Springer Science & Business Media, 2013.

[19] Richard L Miller. "Nature of the vocal cord wave". In: *The Journal of the Acoustical Society of America* 31.6 (1959), pp. 667–677.

[20] Gordon E Peterson and Harold L Barney. "Control methods used in a study of the vowels". In: *The Journal of the acoustical society of America* 24.2 (1952), pp. 175–184.

[21] A Sadeghi Naini and MM Homayounpour. "Speaker age interval and sex identification based on jitters, shimmers and mean mfcc using supervised and unsupervised discriminative classification methods". In: *2006 8th international Conference on Signal Processing*. Vol. 1. IEEE. 2006.

[22] Steven Davis and Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), pp. 357–366.

[23] Matthieu Hébert and Larry P Heck. "Phonetic class-based speaker verification". In: *Eighth European Conference on Speech Communication and Technology*. 2003.

[24] Sachin S Kajarekar and Hynek Hermansky. "Speaker verification based on broad phonetic categories". In: *2001: A Speaker Odyssey-The Speaker Recognition Workshop*. 2001.

[25] Robert Faltlhauser and Günther Ruske. "Improving speaker recognition using phonetically structured gaussian mixture models". In: *Seventh European Conference on Speech Communication and Technology*. 2001.

[26] Asmaa El Hannani, Dijana Petrovska-Delacrétaz, and Gérard Chollet. "Linear and non-linear fusion of ALISP-based and GMM systems for text-independent speaker verification". In: *ODYSSEY04-The Speaker and Language Recognition Workshop*. 2004.

[27] Joseph P Campbell. "Speaker recognition: A tutorial". In: *Proceedings of the IEEE* 85.9 (1997), pp. 1437–1462.

[28] Frank K Soong et al. "Report: A vector quantization approach to speaker recognition". In: *AT&T technical journal* 66.2 (1987), pp. 14–26.

[29] Sadaoki Furui. “Comparison of speaker recognition methods using statistical features and dynamic features”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.3 (1981), pp. 342–350.

[30] Douglas A Reynolds and Richard C Rose. “Robust text-independent speaker identification using Gaussian mixture speaker models”. In: *IEEE transactions on speech and audio processing* 3.1 (1995), pp. 72–83.

[31] Douglas A Reynolds, Thomas F Quatieri, and Robert B Dunn. “Speaker verification using adapted Gaussian mixture models”. In: *Digital signal processing* 10.1-3 (2000), pp. 19–41.

[32] Todd K Moon. “The expectation-maximization algorithm”. In: *IEEE Signal processing magazine* 13.6 (1996), pp. 47–60.

[33] J-L Gauvain and Chin-Hui Lee. “Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains”. In: *IEEE transactions on speech and audio processing* 2.2 (1994), pp. 291–298.

[34] Kevin R Farrell, Richard J Mammone, and Khaled T Assaleh. “Speaker recognition using neural networks and conventional classifiers”. In: *IEEE Transactions on speech and audio processing* 2.1 (1994), pp. 194–205.

[35] B Yegnanarayana and S Prahallad Kishore. “AANN: an alternative to GMM for pattern recognition”. In: *Neural Networks* 15.3 (2002), pp. 459–469.

[36] William M Campbell et al. “SVM based speaker verification using a GMM supervector kernel and NAP variability compensation”. In: *2006 IEEE International conference on acoustics speech and signal processing proceedings*. Vol. 1. IEEE. 2006, pp. I–I.

[37] William M Campbell, Douglas E Sturim, and Douglas A Reynolds. “Support vector machines using GMM supervectors for speaker verification”. In: *IEEE signal processing letters* 13.5 (2006), pp. 308–311.

[38] Patrick Kenny et al. “Joint factor analysis versus eigenchannels in speaker recognition”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 15.4 (2007), pp. 1435–1447.

[39] Najim Dehak et al. “Front-end factor analysis for speaker verification”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.4 (2010), pp. 788–798.

[40] Simon JD Prince and James H Elder. “Probabilistic linear discriminant analysis for inferences about identity”. In: *2007 IEEE 11th International Conference on Computer Vision*. IEEE. 2007, pp. 1–8.

[41] Ondřej Glembek et al. “Simplification and optimization of i-vector extraction”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2011, pp. 4516–4519.

[42] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. “Deep blue”. In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[43] Douglas B Lenat and Ramanathan V Guha. *Building large knowledge-based systems; representation and inference in the Cyc project*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[44] Shlomo Mor-Yosef et al. “Vaginal delivery following one previous cesarean birth: nation wide survey”. In: *Asia-Oceania Journal of Obstetrics and Gynaecology* 16.1 (1990), pp. 33–37.

[45] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[46] Nan Hou et al. “Non-fragile state estimation for discrete Markovian jumping neural networks”. In: *Neurocomputing* 179 (2016), pp. 238–245.

[47] Fan Yang et al. “A new approach to non-fragile state estimation for continuous neural networks with time-delays”. In: *Neurocomputing* 197 (2016), pp. 205–211.

[48] Yajing Yu et al. “Design of non-fragile state estimators for discrete time-delayed neural networks with parameter uncertainties”. In: *Neurocomputing* 182 (2016), pp. 18–24.

[49] Yuan Yuan and Fuchun Sun. “Delay-dependent stability criteria for time-varying delay neural networks in the delta domain”. In: *Neurocomputing* 125 (2014), pp. 17–21.

[50] Jie Zhang, Lifeng Ma, and Yurong Liu. “Passivity analysis for discrete-time neural networks with mixed time-delays and randomly occurring quantization effects”. In: *Neurocomputing* 216 (2016), pp. 657–665.

[51] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[53] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[54] LB Cohen. “Changes in neuron structure during action potential propagation and synaptic transmission.” In: *Physiological reviews* 53.2 (1973), pp. 373–418.

[55] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.

[56] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[57] Marvin L Minsky and Seymour A Papert. “Perceptrons: expanded edition”. In: (1988).

[58] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[59] Bernard Widrow and Marcian E Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[60] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[61] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[62] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. “Multilayer feedforward networks are universal approximators.” In: *Neural networks* 2.5 (1989), pp. 359–366.

[63] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[64] Li Deng. “A tutorial survey of architectures, algorithms, and applications for deep learning”. In: *APSIPA Transactions on Signal and Information Processing* 3 (2014).

[65] Dong Yu and Li Deng. “Deep learning and its applications to signal and information processing [exploratory dsp]”. In: *IEEE Signal Processing Magazine* 28.1 (2010), pp. 145–154.

[66] Geoffrey E Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[67] Yoshua Bengio and Olivier Delalleau. “Justifying and generalizing contrastive divergence”. In: *Neural computation* 21.6 (2009), pp. 1601–1621.

[68] Ilya Sutskever and Tijmen Tieleman. “On the convergence properties of contrastive divergence”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 789–795.

[69] Asja Fischer and Christian Igel. “Training RBMs based on the signs of the CD approximation of the log-likelihood derivatives.” In: *ESANN*. 2011.

[70] Volodymyr Mnih, Hugo Larochelle, and Geoffrey E Hinton. “Conditional restricted boltzmann machines for structured output prediction”. In: *arXiv preprint arXiv:1202.3748* (2012).

[71] Li Deng. “Three classes of deep learning architectures and their applications: a tutorial survey”. In: *APSIPA transactions on signal and information processing* (2012).

[72] Geoffrey E Hinton et al. “The" wake-sleep" algorithm for unsupervised neural networks”. In: *Science* 268.5214 (1995), pp. 1158–1161.

[73] Ao Tang et al. “A real-time hand posture recognition system using deep neural networks”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 6.2 (2015), pp. 1–23.

[74] Itamar Arel, Derek C Rose, and Thomas P Karnowski. “Deep machine learning-a new frontier in artificial intelligence research [research frontier]”. In: *IEEE computational intelligence magazine* 5.4 (2010), pp. 13–18.

[75] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), p. 106.

[76] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[77] Y Bengio. *Learning deep architectures for AI. Found Trends Mach Learn 2 (1): 1–127*. 2009.

[78] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[79] Y-Lan Boureau, Jean Ponce, and Yann LeCun. “A theoretical analysis of feature pooling in visual recognition”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 111–118.

[80] Y-Lan Boureau et al. “Ask the locals: multi-way local pooling for image recognition”. In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2651–2658.

[81] Yangqing Jia, Chang Huang, and Trevor Darrell. “Beyond spatial pyramids: Receptive field learning for pooled image features”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 3370–3377.

[82] Askar Rozi et al. “An open/free database and Benchmark for Uyghur speaker recognition”. In: *2015 International Conference Oriental COCOSDA held jointly with 2015 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*. IEEE. 2015, pp. 81–85.

[83] Joaquin Gonzalez-Rodriguez. “Evaluating automatic speaker recognition systems: An overview of the nist speaker recognition evaluations (1996-2014)”. In: *Loquens* (2014).

[84] Håkan Melin. “Databases for speaker recognition: Activities in COST250 working group 2”. In: *COST 250-Speaker Recognition in Telephony, Final Report 1999* (1999).

[85] Joseph P Campbell and Douglas A Reynolds. “Corpora for the evaluation of speaker recognition systems”. In: *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*. Vol. 2. IEEE. 1999, pp. 829–832.

[86] John S Garofolo et al. “DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1”. In: *STIN* 93 (1993), p. 27403.

[87] D Petrovska et al. “Polycost: a telephone-speech database for speaker recognition”. In: *RLA2C, Avignon, France* (1998), pp. 211–214.

[88] Alfredo Maesa et al. “Text independent automatic speaker recognition system using mel-frequency cepstrum coefficient and gaussian mixture models”. In: *Journal of Information Security* 3.04 (2012), p. 335.

[89] Saurabh Bhardwaj et al. “GFM-based methods for speaker identification”. In: *IEEE transactions on cybernetics* 43.3 (2013), pp. 1047–1058.

[90] N Vishnu Prasad and Srinivasan Umesh. “Improved cepstral mean and variance normalization using Bayesian framework”. In: *2013 IEEE workshop on automatic speech recognition and understanding*. IEEE. 2013, pp. 156–161.

[91] Seyed Omid Sadjadi et al. “The 2016 NIST Speaker Recognition Evaluation.” In: *Interspeech*. 2017, pp. 1353–1357.

[92] Christopher Cieri, David Miller, and Kevin Walker. “The Fisher Corpus: a Resource for the Next Generations of Speech-to-Text.” In: *LREC*. Vol. 4. 2004, pp. 69–71.

[93] Arnav Gupta and Harshit Gupta. "Applications of MFCC and vector quantization in speaker recognition". In: *Intelligent Systems and Signal Processing (ISSP), 2013 International Conference on*. IEEE. 2013, pp. 170–173.

[94] Satyanand Singh and E G Rajan. "Vector Quantization Approach for Speaker Recognition using MFCC and Inverted MFCC". In: *International Journal of Computer Applications* 17.1 (2011), pp. 975–8887.

[95] D. Burton. "Text-dependent speaker verification using vector quantization source coding". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.2 (Feb. 1987), pp. 133–143.

[96] John A Hartigan and Manchek A Wong. "Algorithm AS 136: A k-means clustering algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108.

[97] Dzmitry Bahdanau et al. "End-to-end attention-based large vocabulary speech recognition". In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016, pp. 4945–4949.

[98] Abdel-rahman Mohamed, George E Dahl, Geoffrey Hinton, et al. "Acoustic modeling using deep belief networks". In: *IEEE Trans. Audio, Speech & Language Processing* 20.1 (2012), pp. 14–22.

[99] Geoffrey Hinton et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal processing magazine* 29.6 (2012), pp. 82–97.

[100] Andrej Karpathy and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137.

[101] Chao Dong et al. "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2016), pp. 295–307.

[102] Mohammed Senoussaoui et al. "First attempt of boltzmann machines for speaker verification". In: *Odyssey 2012-The Speaker and Language Recognition Workshop*. 2012.

[103] Vasileios Vasilakakis et al. "Speaker recognition by means of deep belief networks". In: *Proc. Biometric Technologies in Forensic Science* (2013).

[104] Sreenivas Sremath Tirumala and Seyed Reza Shahamiri. "A review on Deep Learning approaches in Speaker Identification". In: *Proceedings of the 8th international conference on signal processing systems*. ACM. 2016, pp. 142–147.

[105] Yuan Liu et al. "Deep feature for text-dependent speaker verification". In: *Speech Communication* 73 (2015), pp. 1–13.

[106] Yun Lei et al. "A novel scheme for speaker recognition using a phonetically-aware deep neural network". In: *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1695–1699.

[107] Patrick Kenny et al. "Deep neural networks for extracting baum-welch statistics for speaker recognition". In: *Proc. Odyssey*. 2014, pp. 293–298.

[108] Fred Richardson, Douglas Reynolds, and Najim Dehak. "Deep neural network approaches to speaker and language recognition". In: *IEEE Signal Processing Letters* 22.10 (2015), pp. 1671–1675.

[109] Mitchell McLaren, Yun Lei, and Luciana Ferrer. "Advances in deep neural network approaches to speaker recognition". In: *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE. 2015, pp. 4814–4818.

[110] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[111] Hossein Salehghaffari. "Speaker Verification using Convolutional Neural Networks". In: *arXiv preprint arXiv:1803.05427* (2018).

[112] Yanick Lukic et al. "Speaker identification and clustering using convolutional neural networks". In: *2016 IEEE 26th international workshop on machine learning for signal processing (MLSP)*. IEEE. 2016, pp. 1–6.

[113] László Tóth. "Combining time-and frequency-domain convolution in convolutional neural network-based phone recognition". In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 190–194.

[114] Yu-hsin Chen et al. "Locally-connected and convolutional neural networks for small footprint speaker recognition". In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[115] Chao Li et al. "Deep speaker: an end-to-end neural speaker embedding system". In: *arXiv preprint arXiv:1705.02304* (2017).

[116] Tijmen Tieleman and Geoffrey Hinton. "Using fast weights to improve persistent contrastive divergence". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM. 2009, pp. 1033–1040.

[117] Chunlei Zhang, Kazuhito Koishida, and John HL Hansen. "Text-independent speaker verification based on triplet convolutional neural network embeddings". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.9 (2018), pp. 1633–1644.

[118] Mirco Ravanelli and Yoshua Bengio. "Speaker recognition from raw waveform with sincnet". In: *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2018, pp. 1021–1028.

[119] Dimitri Palaz, Ronan Collobert, et al. *Analysis of cnn-based speech recognition system using raw speech as input*. Tech. rep. Idiap, 2015.

[120] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. "A survey of binary similarity and distance measures". In: *Journal of Systemics, Cybernetics and Informatics* (2010), pp. 43–48.

[121] Simon Haykin. "Neural Networks: A Comprehensive Foundation, Mac". In: *Milan, New York* (1994).

[122] Léon Bottou. "Stochastic gradient descent tricks". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.

[123] James Martens. "Deep learning via hessian-free optimization." In: *ICML*. Vol. 27. 2010, pp. 735–742.

Centre d'Etudes Doctorales : Sciences et Techniques de l'Ingénieur

Titre de la thèse : Étude des Techniques de Deep Learning en Reconnaissance Automatique du Locuteur

Nom et prénom du candidat : HOURRI Soufiane
Spécialité : Informatique

Résumé de la thèse

La reconnaissance automatique du locuteur (RAL), parfois appelée biométrie du locuteur, implique l'identification, la vérification (authentification), la classification et, par extension, la segmentation, le suivi et la détection des locuteurs. Il s'agit d'un terme générique utilisé pour toute procédure impliquant la reconnaissance de l'identité d'une personne sur la base de sa voix. Dans ce contexte, le deep learning a suscité beaucoup plus d'intérêt de la part des chercheurs en traitement de la parole, et il a été introduit récemment dans la RAL. Dans la plupart des cas, les modèles de deep learning sont adaptés des applications de reconnaissance automatique de la parole (RAP) et appliqués à la RAL, et ils ont montré leur capacité à concurrencer les approches de l'état de l'art. Néanmoins, l'utilisation de deep learning dans la RAL est toujours liée à la RAP. D'autre part, les modèles de deep learning sont maintenant considérés comme état de l'art dans nombreux domaines de la reconnaissance des formes. En RAL, plusieurs architectures ont été étudiées, telles que les réseaux de neurones profonds (DNN), les réseaux de croyances profonds (DBN), les machines de Boltzmann restreintes (RBM), etc. tandis que les réseaux de neurones convolutifs (CNN) sont les modèles les plus utilisés en traitement d'images.

L'objectif de cette thèse est d'étudier les modèles de deep learning pour le domaine de la RAL. Pour cette raison, nous avons proposé une nouvelle façon d'utiliser les DBN et les DNN dans la RAL, dans le but d'extraire les caractéristiques profondes du locuteur (DeepSF). Par la suite, nous avons proposé une nouvelle utilisation des CNN pour le problème de la RAL. Bien qu'ils soient particulièrement conçus pour les problèmes de traitement d'images, les CNN ont récemment été appliqués à la RAL en utilisant des spectrogrammes comme images d'entrée. Nous pensons que cette approche n'est pas optimale car elle peut entraîner deux erreurs cumulatives dans la résolution d'un problème de traitement d'images et de RAL. C'est pourquoi nous avons développé une nouvelle méthode qui permet d'utiliser les CNN sans utiliser d'images.

Les résultats de la thèse représentent une découverte importante pour comprendre comment les modèles de deep learning peuvent être adaptés au problème de la RAL.

Mots clés : *Reconnaissance Automatique du Locuteur, Vérification Automatique du Locuteur, Apprentissage Profond, Réseau de Neurones Profond, Réseau de Croyance profond, Réseau de Neurones Convotutif, Machine de Boltzmann Restreinte.*