Réseaux de neurones récurrents pour le traitement automatique de la parole

Grégory Gelly

NNT : 2017SACLS295

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université Paris-Sud

Ecole doctorale n°580
Sciences et technologies de l'information et de la communication
Spécialité de doctorat : Informatique

par

M. Gregory Gelly

Réseaux de neurones récurrents pour le traitement automatique de la parole

Thèse présentée et soutenue à Orsay, le 22 septembre 2017.

Composition du Jury :

M.	Gaël Richard	Professeur Telecom ParisTech	Président du jury
M.	Jerome Bellegarda	Scientifique distingué Apple Inc.	Rapporteur
M.	Denis Jouvet	Directeur de Recherche INRIA-LORIA	Rapporteur
M	Hermann Ney	Professeur RWTH Aachen	Examinateur
M.	Sylvain Surcin	Ingénieur Ministère de la Défense	Examinateur
M.	Jean-Luc Gauvain	Directeur de Recherche LIMSI-CNRS	Directeur de thèse

Remerciements

Je tiens tout d'abord à remercier MM. Jerôme Bellegarda et Denis Jouvet pour avoir accepté de rapporter sur mes travaux. Je remercie par la même occasion MM. Gaël Richard, Hermann Ney et Sylvain Surcin d'avoir accepté de participer à mon jury de thèse.

Je remercie chaleureusement mon directeur de thèse, M. Jean-Luc Gauvain, qui m'a fait confiance tout au long de ces trois années de thèse. Il a su créer des conditions de travail idéales malgré ma présence réduite dans les locaux du LIMSI, alliant une très grande liberté et la participation à plusieurs évaluations internationales. Pour tout cela, je le remercie vivement.

Je tiens à remercier également Hervé, Claude et Camille pour les discussions techniques et moins techniques, à la cantine ou autour d'un café qui m'ont permis de toujours avancer même quand c'était difficile.

Merci beaucoup à Aude pour sa relecture exhaustive du manuscrit. Ça n'a pas dû être qu'un plaisir mais tu fais partie des quelques personnes qui l'ont lu en entier et ton aide m'a été précieuse.

Je remercie également tous mes collègues du [redacted] et tout particulièrement [redacted], [redacted], [redacted], [redacted] et [redacted] qui m'ont patiemment écouté parler de mes travaux et de mon état de fatigue pendant ces trois années.

Un énorme merci à Laura qui m'a soutenu tout au long de ce projet de thèse, depuis l'idée de départ jusqu'à l'organisation du pot de soutenance en passant par les soirées à me regarder bosser.

Je finirai par un petit mot à l'attention de Delilah et Louise qui ont su me supporter quand il fallait que je travaille constamment sur ma thèse au lieu de jouer, danser et raconter des histoires. Delilah, tu vas bientôt pouvoir avoir mon ordinateur de thèse ;)

Table des matières

Introduction

Le domaine du traitement automatique de la parole regroupe un très grand nombre de tâches parmi lesquelles on trouve la tâche principale de reconnaissance de la parole mais également des tâches plus simples comme la détection de la parole, l'identification de la langue ou encore l'identification du locuteur. Ce domaine de recherche est étudié depuis le milieu du vingtième siècle, en effet l'un des premiers systèmes de reconnaissance de la parole a été conçu par des chercheurs de Bell Labs en 1952 pour identifier des chiffres isolés prononcés par un unique locuteur. Les systèmes de reconnaissance des années 50 ne dépassèrent pas des tailles de vocabulaire de plus de dix mots et les technologies de reconnaissance de la parole évoluèrent peu pendant cette période. C'est à la fin des années 60 qu'on voit apparaître les premiers systèmes permettant de traiter de la parole continue avec les travaux de Raj Reddy sur un système de commande vocale pour jouer aux échecs et la conception de l'algorithme de DTW [1] par des chercheurs soviétiques pour un système ayant un vocabulaire d'environ 200 mots. Dans les années 70 les efforts de recherche dans le domaine furent relancés grâce au programme "Speech Understanding Research" de la DARPA et on voit alors apparaître l'utilisation de l'analyse cepstrale, de la LPC [2] et des HMM [3]. Ces derniers sont encore très largement utilisés dans les systèmes d'aujourd'hui. Dans les années 80, l'équipe de Fred Jelinek à IBM mis au point le système Tangora fondé sur des HMM et ayant un vocabulaire de vingt mille mots. C'est également à cette période que le modèle de langage n-grammes, lui aussi encore largement utilisé aujourd'hui, fait son apparition. Il est à noter que toutes ces avancées n'ont été rendues possibles que grâce aux progrès exponentiels réalisés sur la capacité de calcul des moyens informatiques utilisables par les chercheurs.

Avec l'évolution des systèmes de reconnaissance et des moyens informatiques (naissance du PC), on voit apparaître dans les années 90 les premiers systèmes commerciaux grand public pour la dictée de texte en continu ayant des vocabulaires dont la taille dépasse le vocabulaire moyen d'un être humain. Durant ces années là et le début des années 2000, les systèmes de reconnaissance de la parole continuent de progresser en particulier grâce aux grands programmes financés par la DARPA et des campagnes d'évaluations internationales organisées par le NIST qui permettent d'évaluer objectivement les performances des technologies proposées par les différents participants. Mais

1. Un certain nombre d'acronymes anglais sont utilisés dans cette thèse lorsqu'ils sont très usités en France où lorsqu'il n'y a pas d'acronyme équivalent existant en français.

2. *Linear Predictive Coding*. Permet de modéliser l'enveloppe spectrale d'un signal audio sous une forme compressée à l'aide d'un modèle prédictif linéaire.

3. Le modèle de Markov caché (*Hidden Markov Model*) est un modèle statistique dans lequel le système modélisé est supposé être un processus markovien de paramètres inconnus.

il faut attendre 2009 pour la prochaine rupture technologique et une amélioration marquée de l'état de l'art avec l'utilisation de réseaux de neurones profonds (*Deep Neural Network*) comme modèle acoustique par les groupes de recherche de Geoffrey Hinton et Li Deng. L'utilisation de systèmes hybrides HMM-DNN pour le traitement de la parole n'est pas récente et les premiers travaux remontent aux années quatre-vingt. Mais jusque là les résultats obtenus n'était pas compétitifs avec ceux des systèmes fondés sur des modèles HMM-DNN. Avec les travaux de Hinton et Deng on assiste au début des années 2010 à une déferlante de travaux sur l'utilisation des DNN en reconnaissance de la parole surtout pour les modèles acoustiques mais aussi pour les modèles de langage. En 2013, lorsque nous avons initié les travaux présentés ici les efforts de recherche ne portaient quasi-exclusivement que sur les réseaux de neurones profonds de type perceptron multi-couches (*MultiLayer Perceptron* - MLP). Inspirés par les travaux d'Alex Graves [1, 2], nous souhaitions explorer les capacités d'un autre type de réseaux de neurones : les réseaux de neurones récurrents (RNN). En effet, ces derniers nous semblaient plus adaptés pour traiter au mieux les séquences temporelles du signal de parole.

Les RNN ont plusieurs caractéristiques qui en font un modèle de choix pour le traitement automatique de la parole. Ils peuvent en particulier conserver et prendre en compte dans leurs décisions l'information contextuelle passée (et future dans le cas de RNN bidirectionnels). Le but de cette thèse est d'approfondir les travaux déjà réalisés sur l'utilisation des RNN pour le traitement automatique de la parole et de confronter ce modèle neuronal à deux tâches : la détection de la parole, et l'identification de la langue ou du dialecte.

Nous nous intéressons tout particulièrement aux RNN à mémoire court-terme persistante (*Long Short-Term Memory*) qui permettent de s'affranchir d'un certain nombre de difficultés rencontrées avec des RNN standards. Nous augmentons ce modèle et nous proposons des processus d'optimisation permettant d'améliorer les performances obtenues en détection de la parole et en identification de la langue.

Contributions

L'objectif principal de cette thèse est d'étudier le comportement des LSTM sur des tâches spécifiques de traitement automatique de la parole telles que la détection de la parole et l'identification de la langue ; et de proposer de nouvelles méthodes d'apprentissage de manière à améliorer les performances tout en essayant de diminuer le temps nécessaire à cet apprentissage.

Premièrement, nous introduisons une modification du modèle LSTM en ajoutant des liens entre les portes logiques des cellules pour améliorer leur comportement. Ce nouveau modèle que nous appelons Long Short-Term Memory with Coordinated Gates (CG-LSTM) est comparé au modèle LSTM standard ainsi qu'à d'autres types de réseaux de neurones.

Pour la détection de la parole, nous proposons un processus d'apprentissage regroupant des algorithmes d'optimisation et des fonctions de coût permettant d'optimiser le comportement d'un système utilisant des CG-LSTM pour différentes tâches. On peut par exemple vouloir minimiser directement le Frame Error Rate (FER) ou bien

s'attacher à minimiser indirectement le Word Error Rate (WER) d'un système de reconnaissance de parole en aval du système de détection de la parole.

Pour l'identification de la langue, nous présentons une stratégie d'apprentissage s'inspirant de la maxime *diviser pour mieux régner*. Cet apprentissage que nous désignons par D&C (Divide and Conquer) divise la tâche principale de classification multi-classes (identifier la langue parlée parmi les n langues cibles) en sous-tâches plus simples que l'on résout avec de petits réseaux de neurones. Ces petits réseaux sont ensuite recombinés pour initialiser le réseau que l'on souhaite utiliser comme classifieur multilingue. Cette méthode permet à la fois d'obtenir une meilleure identification de langue mais également de diminuer le temps d'apprentissage.

Ensuite, toujours pour l'identification de la langue, nous introduisons une architecture neuronale permettant de projeter les séquences temporelles d'entrées sur un unique vecteur unitaire dans un espace vectoriel de dimension n, dont la direction est représentative de la langue. Conjointement, nous proposons la fonction de coût dite de *proximité angulaire* qui permet de réaliser l'apprentissage d'un tel modèle. Ces deux contributions permettent d'améliorer significativement les performances en identification de la langue et de diminuer très fortement la durée d'apprentissage (50% de la durée d'apprentissage avec D&C).

Structure du document

Ce manuscrit est organisé en six chapitres.

Le premier chapitre détaille le contexte de nos travaux au travers d'une présentation générale des réseaux de neurones ainsi que des deux tâches de traitement automatique de la parole sur lesquelles nous avons travaillé : la détection de la parole et l'identification de la langue parlée. Pour ces deux tâches nous présentons les données que nous avons utilisées ainsi que les protocoles expérimentaux que nous avons suivis.

Le Chapitre 2 est consacré à la première contribution de cette thèse c'est-à-dire le modèle CG-LSTM dont nous donnons la description mathématique ainsi que le mécanisme d'apprentissage par rétro-propagation du gradient. Ce nouveau modèle est comparé à différents types de réseaux de neurones plus classiques sur les deux tâches de détection de la parole et d'identification de la langue parlée.

Le Chapitre 3 détaille le processus d'optimisation (algorithmes + fonctions de coût) que nous proposons pour pouvoir optimiser les paramètres d'un algorithme de détection de la parole utilisant des CG-LSTM de manière à minimiser le WER d'un système de reconnaissance de parole en aval du système de détection.

Le Chapitre 4 décrit la méthode d'apprentissage inspirée de la devise *diviser pour mieux régner* que nous proposons dans le but d'améliorer les performances d'un modèle CG-LSTM d'identification de la langue.

Le Chapitre 5 présente une autre approche pour résoudre les problèmes multi-classe à l'aide d'un modèle CG-LSTM. On présente ainsi une nouvelle architecture neuronale de *projection vectorielle* avec une fonction de coût de *proximité angulaire* qui permettent d'améliorer notablement les performances d'identification de la langue tout en accélérant significativement l'apprentissage.

Finalement, le Chapitre 6 détaille les algorithmes d'optimisation des paramètres des réseaux de neurones que nous avons sélectionnés pour nos diverses expérimenta-

tions, ainsi que les bonnes pratiques que nous avons recueillies durant cette thèse pour l'apprentissage des réseaux de neurones.

Chapitre 1

Contexte et conditions expérimentales

L'utilisation des réseaux de neurones artificiels pour le traitement automatique de la parole n'est pas récente. Dès les années 1980, on voit apparaître des systèmes utilisant les réseaux de neurones pour reconnaître des voyelles [3, 4] puis pour la reconnaissance de phonèmes [5–7]. Mais les résultats obtenus à cette époque ne permettent pas d'améliorer l'état de l'art. Pendant les deux décennies suivantes quelques progrès sont réalisés [8–11] mais il faut attendre le début des années 2010 et l'émergence des réseaux de neurones profonds (DNN) avec de nouvelles méthodes d'apprentissage et des ressources informatiques spécifiques (GPU), pour que ces derniers s'imposent comme solution à l'état de l'art dans les systèmes de reconnaissance de la parole à large vocabulaire.

Dans ce chapitre, nous commençons par décrire les plus classiques des réseaux de neurones artificiels : le perceptron multi-couches (*MultiLayer Perceptron*), et le réseau de neurones récurrent (*Recurrent Neural Network*) proposé par Elman [4]. Nous détaillons leurs modèles mathématiques ainsi que les mécanismes de rétro-propagation du gradient des erreurs permettant leur apprentissage. Ces éléments seront repris lorsque nous décrirons le modèle récurrent CG-LSTM proposé dans le chapitre suivant.

Ensuite, nous détaillons les deux tâches de traitement automatique de la parole sur lesquelles nous avons travaillé durant cette thèse : la détection de la parole et l'identification de la langue parlée. Pour ces deux tâches, nous décrivons les algorithmes classiques (sans réseaux de neurones), les travaux sur des algorithmes utilisant des réseaux de neurones précédant les travaux de cette thèse, ainsi que les conditions expérimentales dans lesquelles nous nous sommes placés pour valider nos développements.

Les algorithmes utilisant des réseaux de neurones que nous avons conçus spécifiquement pour répondre à nos besoins sont quant à eux présentés dans les chapitres 3, 4 et 5.

1.1 Réseaux de neurones

Avec l'augmentation rapide des capacités de calcul des ordinateurs et l'évolution des techniques d'apprentissage, l'utilisation des réseaux de neurones connaît un vif essor en particulier au sein des communautés du traitement d'image, de la traduction automatique et du traitement de la parole. Au cours des dernières années, deux types de réseaux de neurones ont marqué une rupture technologique dans le domaine du traitement de la parole : les réseaux dits "profonds" et les réseaux récurrents. Nous détaillons dans cette partie le fonctionnement de ces réseaux de neurones et les travaux que nous avons réalisés sur les modèles eux-mêmes et sur les méthodes d'apprentissage.

Le réseau de neurones artificiels (*Artificial Neural Network* - ANN) fut introduit comme un modèle rudimentaire du traitement de l'information dans le cerveau humain. Ainsi, la structure élémentaire d'un ANN est un réseau de petits nœuds de calcul reliés entre eux par des liens dirigés et pondérés (cf. Fig. 1.1). Les nœuds représentent les neurones et les liens pondérés représentent la force des connections synaptiques reliant les neurones entre eux. Dans cette représentation, le neurone peut alors être un sommateur des potentiels des signaux synaptiques qui lui parviennent, et qui transmet à son tour une information basée sur cette somme via une fonction de transfert de préférence non linéaire (cf. Fig. 1.2). Un ANN est activé en injectant des données au niveau de tout ou partie des nœuds puis en propageant l'information en suivant les liens pondérés. Une fois l'information propagée, on peut collecter les niveaux d'activation de tout ou partie des nœuds et les utiliser comme commande d'un système, comme prédiction ou encore comme classification.

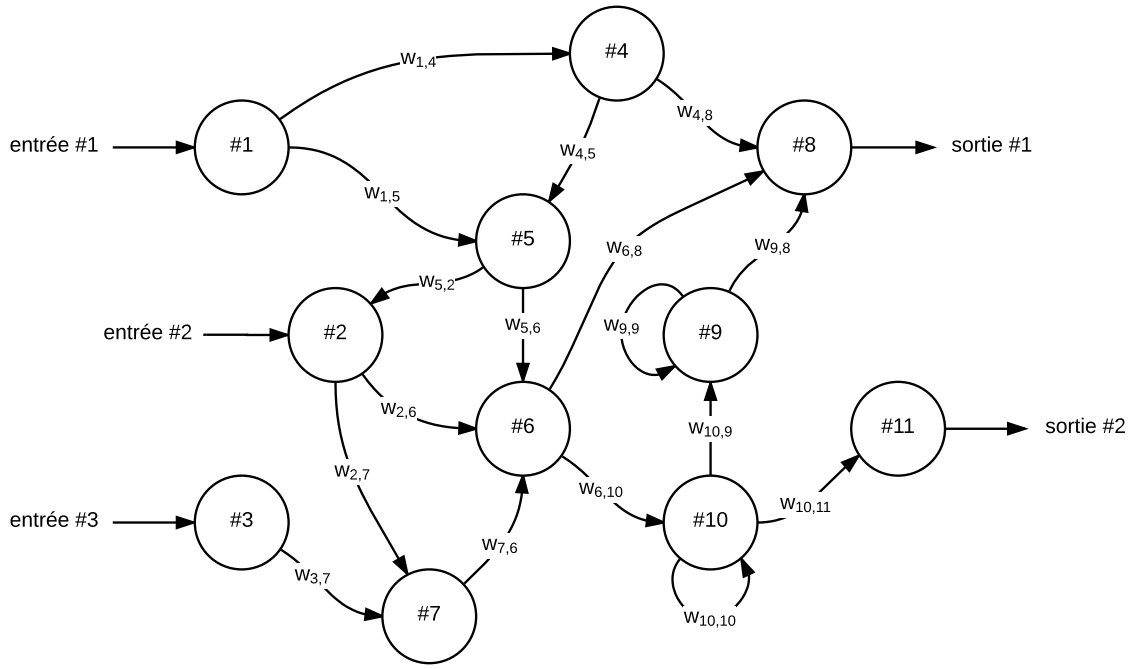


FIGURE 1.1 – Représentations d'un réseau de neurones artificiels (ANN) sous la forme d'un réseau de nœuds de calculs reliés par des liens dirigés et pondérés par les coefficients $w_{i,j}$. L'information se propage en suivant ces liens entre les entrées et les sorties.

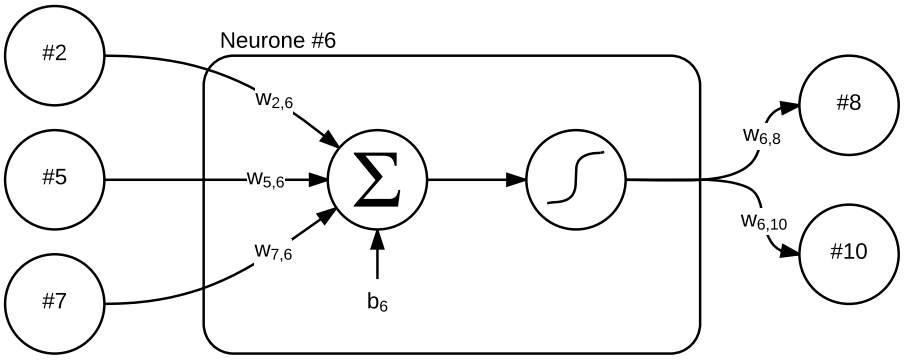


FIGURE 1.2 – Structure interne d'un nœud de calcul dans le cas d'un neurone artificiel sommateur. Tout d'abord, la somme pondérée des entrées du nœud est calculée puis on applique à cette somme une fonction de transfert non-linéaire. La sortie est alors communiquée aux autres nœuds de calculs.

De nombreux types de réseaux de neurones ayant des propriétés très diverses ont été développés depuis l'apparition des neurones formels dans les années 1940 [12]. Plus spécifiquement, on distingue deux types de réseaux : ceux dont le graphe de connexion présente au moins un cycle et ceux pour lesquels ce n'est pas le cas. Les premiers sont dits récurrents et les seconds sont dits acycliques. Parmi les réseaux acycliques on trouve des réseaux de type perceptron [13], convolutifs [14], Radial Basis Function [15] ou les cartes de Kohonen [16]. Mais la variante la plus fréquemment utilisée est le MLP.

1.1.1 Perceptron multi-couches

Nous commençons donc par présenter le plus classique des réseaux de neurones : le perceptron multi-couches qui est un réseau de neurones acyclique (*Feed-Forward Neural Network* - FFNN) structuré en couches. Un perceptron multi-couches est ainsi composé d'une couche d'entrée, d'une ou plusieurs couches intermédiaires dites cachées et d'une couche de sortie. Pour chacune des couches, l'ensemble de ses nœuds est relié à l'ensemble des nœuds de la couche précédente. La figure Fig. 1.3 donne une représentation d'un perceptron multi-couches avec deux couches cachées.

L'origine du perceptron multi-couches remonte à F. Rosenblatt qui en 1957 s'inspire des travaux sur les neurones formels [12] et sur la règle de Hebb introduite dans [17] pour développer le modèle du perceptron mais également la méthode qui permet à ce modèle d'apprendre [13]. Ce modèle n'a qu'une seule couche mais permet déjà de résoudre des tâches simples de classification de symboles géométriques. Il est cependant impossible avec la méthode formulée par F. Rosenblatt d'entraîner un système ayant plusieurs couches ce qui se révèle quelques années plus tard très restrictif. En effet, en 1969 M. Minsky et S. Papert démontrent par une analyse rigoureuse que le perceptron est incapable d'apprendre des fonctions si celles-ci ne sont pas séparables linéairement (comme par exemple, la fonction booléene XOR). Ils vont même un peu plus loin en démontrant qu'il est nécessaire d'avoir au moins une couche de neurones supplémentaire

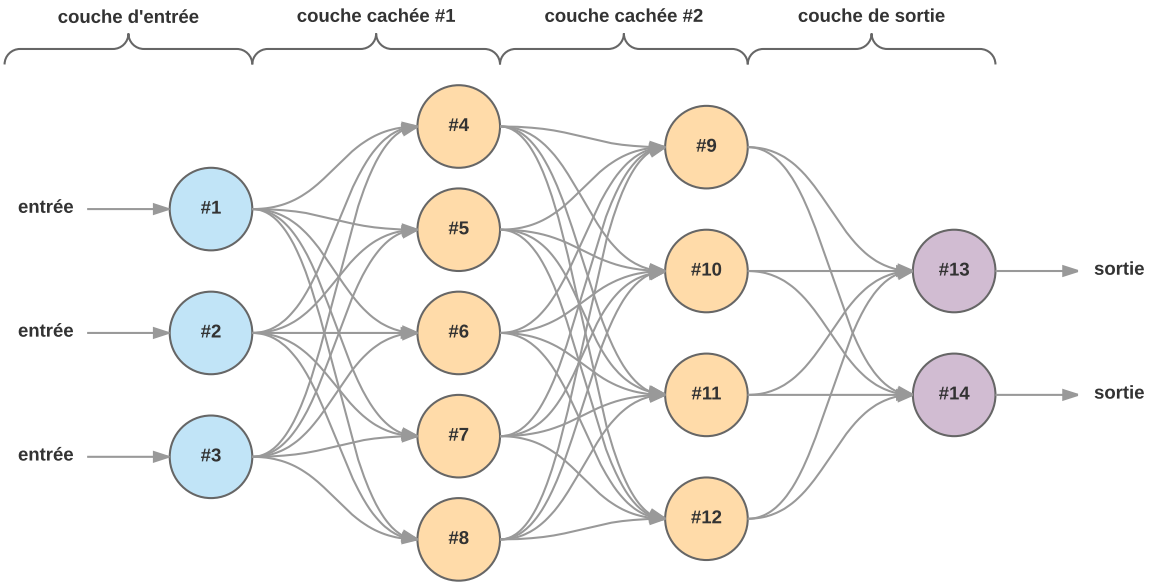


FIGURE 1.3 – Réseau de neurones dense, acyclique et structuré en couches communément appelé perceptron multi-couches (MLP).

(soit un perceptron multi-couches) pour pouvoir régler le problème. Mais il n'existe alors aucune méthode permettant d'entraîner un perceptron multi-couches.

Il faut attendre plus de dix ans pour que des chercheurs mettent au point la technique d'apprentissage qui permet de régler les paramètres d'un perceptron multi-couches. En effet, c'est P. Werbos qui le premier propose l'idée d'appliquer aux ANN la technique de rétro-propagation du gradient des erreurs développée dans les années 1960. Durant sa thèse qu'il soutient en 1974 [18], il analyse la pertinence de cette méthode mais, compte tenu du désintérêt de la communauté scientifique pour les ANN suite à la publication de M. Minsky et S. Papert, il ne publie aucun résultat sur le sujet avant 1982 [19]. C'est finalement au milieu des années 1980 que cette méthode est redécouverte par plusieurs équipes de recherche [20–22] et qu'elle finit par être popularisée. En 1986, D. Rumelhart, G. Hinton et R. Williams montrent dans [23] qu'à l'aide de la rétro-propagation du gradient des erreurs appliquée à un perceptron multi-couches on peut enfin dépasser les limites du perceptron qui avaient été soulevées par M. Minsky et S. Papert en 1969. En particulier, le perceptron multi-couches permet de traiter des problèmes non linéaires complexes et peut approximer, avec une seule couche cachée et un nombre de neurones suffisant, n'importe quelle fonction non linéaire et continue sur un espace compact avec une précision arbitraire [24]. Le perceptron multi-couches est ainsi appelé un approximateur universel de fonctions.

Description mathématique du modèle

Un MLP est défini par les n couches qui le composent et qui se succèdent. La couche $j \in [\![1, N]\!]$ d'un MLP est défini par un triplet : $L_j = (n_j, \sigma_j, a_j)$ où

— $n_j \in \mathbb{N}$ est le nombre de neurones dans la couche j,

— $a_j : \mathbb{R}^{n_{j-1}} \mapsto \mathbb{R}^{n_j}$ est la transformation affine définie par la matrice $\boldsymbol{W}_j \in \mathbb{R}^{n_{j-1} \times n_j}$ et le vecteur $\boldsymbol{b}_j \in \mathbb{R}^{n_j}$,

— $\sigma_j : \mathbb{R}^{n_j} \mapsto \mathbb{R}^{n_j}$ est la fonction de transfert de la couche j.

Il est intéressant de noter que :

— la matrice $\boldsymbol{W}_j$ est appelée matrice d'interconnexion (ou matrice de poids) entre la couche $j-1$ et la couche j,

— le vecteur $\boldsymbol{b}_j$ est appelé vecteur de biais de la couche j,

— la fonction σ_j est traditionnellement choisie parmi les fonctions de $\mathbb{R} \mapsto \mathbb{R}$, non-linéaires, bornées et dérivables telles que la tangente hyperbolique ou la sigmoïde. Dans ce cas, la fonction de transfert est appliquée à chacun des éléments des vecteurs de dimension n_j.

Dans le cas d'un MLP prenant en entrée des vecteurs de dimension n_e et dont les vecteurs de sortie sont de dimension n_s, on a $n_0 = n_e$ et $n_N = n_s$. Alors, pour un vecteur d'entrée donné $\boldsymbol{x} \in \mathbb{R}^{n_e}$, on obtient le vecteur de sortie $\boldsymbol{z} \in \mathbb{R}^{n_s}$ en initialisant le vecteur $\boldsymbol{h}_0 = \boldsymbol{x}$ puis en appliquant itérativement chacune des couches L_j, $j \in [\![1, N]\!]$ au vecteur de sortie de la couche précédente :

— on applique la transformation affine a_j au vecteur de sortie de la couche précédente :

$$\boldsymbol{g}_j = a_j(\boldsymbol{h}_{j-1}) = \boldsymbol{W}_j \times \boldsymbol{h}_{j-1} + \boldsymbol{b}_j \tag{1.1}$$

— puis on applique la fonction de transfert au résultat :

$$\boldsymbol{h}_j = \sigma_j\left(\boldsymbol{g}_j\right) \tag{1.2}$$

avec $\boldsymbol{h}_j, \boldsymbol{g}_j \in \mathbb{R}^{n_j}$ et $\boldsymbol{z} = \boldsymbol{h}_N$.

Les figures Fig.1.4a et Fig.1.4b représentent la couche L_j sous la forme d'un graphe computationnel.

Cette propagation de l'information de l'entrée de la première couche à la sortie de la dernière couche du MLP est communément appelée la "passe avant" du MLP. La figure Fig. 1.5 permet de visualiser ce processus.

Un MLP est donc une fonction $f_{mlp} : \mathbb{R}^{n_e} \mapsto \mathbb{R}^{n_s}$ définie par la composition successive d'une transformation affine et d'une fonction de transfert non-linéaire et ce pour chacune de ses couches :

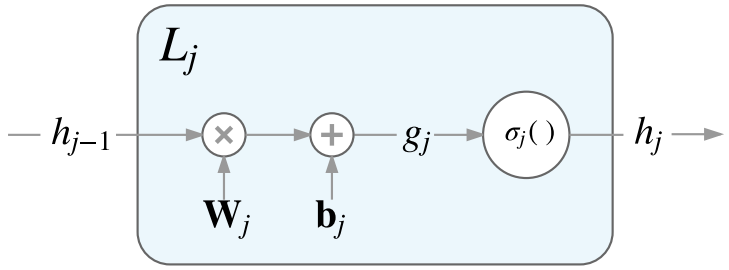
$$\boldsymbol{z} = f_{mlp}(\boldsymbol{x}) = \sigma_N \circ a_N \circ \sigma_{N-1} \circ a_{N-1} \circ ... \circ \sigma_1 \circ a_1(\boldsymbol{x}) \tag{1.3}$$

Ainsi, si l'ensemble des fonctions de transfert du MLP sont différentiables la fonction f_{mlp} l'est également.

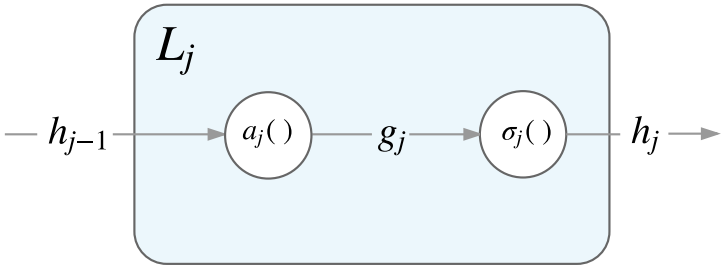
Utilisation pour la classification

La dernière couche L_N du MLP est aussi appelée la couche de sortie du MLP. Suivant l'application à laquelle on destine le MLP, la fonction de transfert σ_N sera différente. Nous nous intéressons ici à des tâches de classification et nous distinguons deux cas d'utilisation : la classification binaire et la classification multi-classes où le nombre de classes est supérieur à deux.

Dans le cas de la classification binaire, la sortie du MLP est de dimension 1 ($n_s = 1$) et la fonction de transfert de la couche de sortie est une sigmoïde dont l'intervalle de



(a) Graphe computationnel détaillé de la couche j d'un MLP.



(b) Graphe computationnel synthétique de la couche j d'un MLP.

FIGURE 1.4 – Représentations de la couche j d'un MLP sous forme de graphe computationnel.

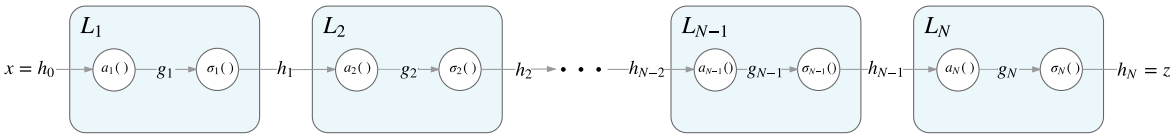


FIGURE 1.5 – Visualisation de la propagation de l'information lors de la passe avant dans un MLP.

sortie est $[0, 1]$. Pour un vecteur d'entrée $\boldsymbol{x}$ on peut ainsi interpréter la sortie $f_{mlp}(\boldsymbol{x}) = z \in \mathbb{R}$ comme la probabilité que le vecteur d'entrée appartienne à la première classe plutôt qu'à la seconde classe. On a alors :

$$p(C = \bar{1}|\boldsymbol{x}) = z \tag{1.4}$$

$$p(C = \bar{0}|\boldsymbol{x}) = (1 - z) \tag{1.5}$$

Dans le cadre d'un apprentissage supervisé, nous faisons l'hypothèse que nous disposons d'exemples de vecteurs d'entrée $\boldsymbol{x}$ pour lesquels nous connaissons la classe d'appartenance. Nous pouvons représenter l'appartenance à l'une des deux classes par un réel cible c qui prend la valeur 1 lorsque $\boldsymbol{x} \in \bar{1}$ et la valeur 0 lorsque $\boldsymbol{x} \in \bar{0}$. On peut alors réécrire les équations Eq. 1.4 et Eq. 1.5 sous la forme d'une unique équation :

$$p(c|\boldsymbol{x}) = z^c \times (1 - z)^{(1-c)} \tag{1.6}$$

Pour les problèmes de classification avec n_C classes et $n_C > 2$, la convention est d'utiliser un MLP avec $n_N = n_C$ et d'utiliser la fonction de transfert appelée *softmax* pour obtenir une estimation des probabilités d'appartenance aux différentes classes. L'application de la fonction *softmax* à chacun des élément $g_{N(k)}$ du vecteur $\boldsymbol{g}_N$ obtenu après la transformation affine a_N de la couche de sortie est régie par l'équation suivante :

$$z_{(k)} = h_{N(k)} = \frac{e^{g_{N(k)}}}{\sum_{m=1}^{n_{Cl}} e^{g_{N(m)}}} \tag{1.7}$$

Dans ce cas, nous pouvons représenter l'appartenance d'un vecteur $\boldsymbol{x}$ d'entrée à la $m_{\text{ième}}$ classe sous la forme d'un vecteur cible $\boldsymbol{c} \in \mathbb{R}^{n_C}$ dont les composantes sont toutes nulles à l'exception de la $m^{\text{ième}}$ composante qui prend la valeur $c_{(m)} = 1$. On peut alors écrire :

$$p(\boldsymbol{c}|\boldsymbol{x}) = \prod_{m=1}^{n_C} z_{(m)}^{c_{(m)}} \tag{1.8}$$

Dans les deux cas, on peut ensuite définir une fonction de coût $\mathcal{C}(\boldsymbol{z} = f_{mlp}(\boldsymbol{x}), \boldsymbol{c})$ qui permet de quantifier l'impact sur la tâche de classification de l'erreur en sortie du MLP par rapport à la cible. La fonction de coût la plus répandue dans ce type de tâche est une fonction de coût dite d'entropie croisée :

$$\mathcal{C}(\boldsymbol{z} = f(\boldsymbol{x}), \boldsymbol{c}) = -\ln p(\boldsymbol{c}|\boldsymbol{x}) \tag{1.9}$$

Dans le cas d'une classification binaire, on a :

$$\mathcal{C}(z, c) = -\ln p(c|\boldsymbol{x}) = (c - 1)\ln(1 - z) - c \ln z \tag{1.10}$$

Pour n_C classes avec $n_C > 2$, on a :

$$\mathcal{C}(\boldsymbol{z}, \boldsymbol{c}) = -\ln p(\boldsymbol{c}|\boldsymbol{x}) = -\sum_{m=1}^{n_C} c_{(m)} \ln z_{(m)} \tag{1.11}$$

Ces fonctions étant monotones décroissantes en fonction de l'erreur, pour améliorer la classification on cherchera à minimiser le coût $\mathcal{C}$.

Rétro-propagation du gradient

Comme nous l'avons vu au §1.1.1, un MLP dont les fonctions de transfert sont différentiables est lui-même un opérateur différentiable. Ainsi, son comportement peut être amélioré dans le but de minimiser n'importe quelle fonction de coût $\mathcal{C}$ également différentiable. Pour ce faire, il est nécessaire de calculer les dérivées partielles de cette fonction de coût par rapport à chacun des paramètres du MLP : c'est à dire $\boldsymbol{W}_j$ et $\boldsymbol{b}_j$ pour $j \in [\![1, N]\!]$. Une fois les dérivées partielles connues, il est possible d'utiliser un algorithme de descente de gradient pour déterminer de nouveaux poids permettant de diminuer le coût actuel. L'exemple le plus simple d'algorithme de descente de gradient est de modifier chacun des poids w dans la direction opposée au gradient du coût par rapport à ce poids :

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} - \lambda \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_{n-1}) \tag{1.12}$$

où λ est le taux d'apprentissage permettant d'ajuster la modification des poids. Cet algorithme est donné pour faciliter la compréhension de la démarche mais on lui préférera des méthodes plus performantes telles que celles décrites dans la partie §6.1.1.

Pour déterminer les dérivées partielles du coût en fonction des paramètres, il est nécessaire de calculer les dérivées partielles du coût en fonction des sorties $\boldsymbol{g}_j$ de chacune des transformations affines du MLP. Ensuite en dérivant Eq. 1.1 nous obtenons les équations suivantes qui nous permettent de déterminer les gradients qui nous intéressent pour $j \in [\![1, N]\!]$:

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} = \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} \times \frac{\partial \boldsymbol{g}_j}{\partial \boldsymbol{b}_j} = \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} \tag{1.13}$$

et

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{W}_j} = \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} \times \frac{\partial \boldsymbol{g}_j}{\partial \boldsymbol{W}_j} = \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} \times \boldsymbol{h}_{j-1}^T \tag{1.14}$$

où ^{T} est l'opérateur de transposition.

Remarque : on voit au passage qu'il est nécessaire de conserver en mémoire tous les vecteurs $\boldsymbol{h}_j$ pour $j \in [\![1, N-1]\!]$ de manière à calculer le gradient du coût par rapport aux matrices de poids $\boldsymbol{W}_j$.

Le calcul des dérivées partielles en sortie de chacune des transformations affines est réalisé grâce à la technique dite de rétro-propagation du gradient qui fut popularisée par la publication [23]. Cette technique exploite les règles de dérivation des fonctions composées de façon itérative en parcourant les couches du MLP de la dernière à la première. C'est ce qu'on appelle communément la "passe arrière" du MLP : les gradients constatés en sortie du réseau de neurones sont propagées à rebours dans les couches du MLP.

Pour effectuer la "passe arrière" du MLP, il faut au préalable réaliser la "passe avant" du MLP telle que décrite au §1.1.1, c'est à dire appliquer à un vecteur d'entrée $\boldsymbol{x}$ les transformations successives correspondant aux N couches du MLP de manière à déterminer le vecteur de sortie $\boldsymbol{z}$ correspondant. On peut alors calculer le coût $\mathcal{C}(\boldsymbol{z}, \boldsymbol{c})$ où $\boldsymbol{c}$ est le vecteur cible associé à $\boldsymbol{x}$. Et dans le cas où $\mathcal{C}$ est différentiable, on peut déterminer son gradient par rapport à la sortie du MLP et donc à la sortie $\boldsymbol{h}_N = \boldsymbol{z}$ de sa dernière couche L_N. Le processus complet est représenté sur la figure Fig. 1.6.

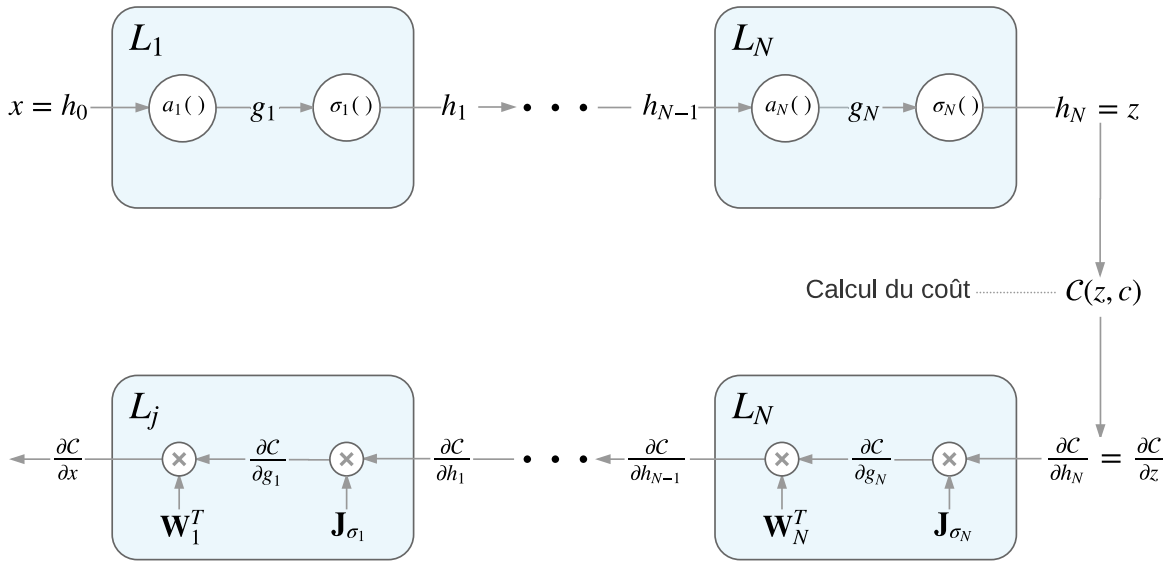


FIGURE 1.6 – Visualisation de la propagation de l'information lors de la passe avant dans un MLP et de la rétro-propagation du gradient de la fonction de coût $\mathcal{C}$ lors de la passe arrière.

En utilisant les règles de dérivation des fonctions composées on obtient alors pour j décroissant de N à 1 :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} = \boldsymbol{J}_{\sigma_j}(\boldsymbol{g}_j) \times \frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j} \tag{1.15}$$

et

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_{j-1}} = \boldsymbol{J}_{a_j}(\boldsymbol{h}_{j-1}) \times \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} \tag{1.16}$$

où on définit $\boldsymbol{J}_\gamma(\boldsymbol{y}) \in \mathbb{R}^{m \times n}$ comme étant la matrice Jacobienne de la fonction $\gamma : \mathbb{R}^m \mapsto \mathbb{R}^n$ au point $\boldsymbol{y} \in \mathbb{R}^m$ soit :

$$\boldsymbol{J}_\gamma(\boldsymbol{y}) = \frac{d\gamma}{d\boldsymbol{x}}(\boldsymbol{y}) = \begin{bmatrix} \frac{\partial \gamma_{(1)}}{\partial x_{(1)}}(\boldsymbol{y}) & \cdots & \frac{\partial \gamma_{(n)}}{\partial x_{(1)}}(\boldsymbol{y}) \\ \vdots & \ddots & \vdots \\ \frac{\partial \gamma_{(1)}}{\partial x_{(m)}}(\boldsymbol{y}) & \cdots & \frac{\partial \gamma_{(n)}}{\partial x_{(m)}}(\boldsymbol{y}) \end{bmatrix} \tag{1.17}$$

On voit au passage que dans le cas général, il est nécessaire de conserver en mémoire tous les vecteurs $\boldsymbol{g}_j$ pour $j \in [\![1, N]\!]$ afin de calculer les matrices Jacobiennes des σ_j lors de la passe arrière.

D'autre part, comme les fonctions a_j sont des transformations affines leurs matrices jacobiennes ont une forme très simple et l'équation Eq. 1.16 devient :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_{j-1}} = \boldsymbol{W}_j^T \times \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j} \tag{1.18}$$

L'algorithme complet de calcul du gradient du réseau de neurones est décrit dans Alg. 1.

Algorithm 1 Calcul du gradient dans un réseau de neurones

Choix d'une paire d'apprentissage $(\boldsymbol{x}, \boldsymbol{c})$
Initialisation de $\boldsymbol{h}_0 = \boldsymbol{x}$
Passe avant :
for $j = 1..N$ **do**
$\boldsymbol{g}_j = a_j(\boldsymbol{h}_{j-1}) = \boldsymbol{W}_j \times \boldsymbol{h}_{j-1} + \boldsymbol{b}_j$
$\boldsymbol{h}_j = \sigma_j\left(\boldsymbol{g}_j\right)$
end for
Calcul du coût et de son gradient en fonction de la sortie $\boldsymbol{h}_N$ du réseau de neurones et de la cible $\boldsymbol{c}$:

- $\mathcal{C}(\boldsymbol{h}_N, \boldsymbol{c})$, $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{h}_N}$

Passe arrière :
for $j = N..1$ **do**
A l'aide des vecteurs $\boldsymbol{g}_j$ et $\boldsymbol{h}_{j-1}$ calculés lors de la "passe avant", on détermine les dérivées partielles de la fonction de coût par rapport aux paramètres de la couche L_j :

- $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} = \boldsymbol{J}_{\sigma_j}(\boldsymbol{g}_j) \times \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j}$
- $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{W}_j} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} \times \boldsymbol{h}_{j-1}^T$

Puis on calcule les dérivées partielles de la fonction de coût par rapport à la sortie de la couche précédente $j - 1$:

- $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{h}_{j-1}} = \boldsymbol{W}_j^T \times \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j}$

end for

Application à un cas simple

Pour illustrer ce que nous venons de décrire, prenons le cas simple d'un MLP à 2 couches ($N = 2$) destiné à la classification binaire $n_N = 1$. Nous utilisons la fonction logistique pour la fonction de transfert σ_2 de la couche de sortie du MLP et nous choisissons la tangente hyperbolique pour la fonction de transfert σ_1 de la couche cachée. Ainsi, $\forall x \in \mathbb{R}$:

$$\sigma_1(x) = \frac{\mathrm{e}(x) - \mathrm{e}(-x)}{\mathrm{e}(x) + \mathrm{e}(-x)} \tag{1.19}$$

$$\sigma_1'(x) = (1 - \sigma_1(x)^2) \tag{1.20}$$

et

$$\sigma_2(x) = \frac{1}{1 + \mathrm{e}(-x)} \tag{1.21}$$

$$\sigma_2'(x) = \sigma_2(x)(1 - \sigma_2(x)) \tag{1.22}$$

Ces 2 fonctions de transfert étant appliquées indépendamment à chacun des éléments de leurs vecteurs d'entrée, leurs matrices Jacobiennes sont diagonales et l'équation Eq. 1.15 prend une forme très simple et plus efficace en termes de calcul :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_1} = (\mathbf{1} - \boldsymbol{h}_1 \odot \boldsymbol{h}_1) \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_1} \tag{1.23}$$

et

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_2} = \boldsymbol{h}_2 \odot (\mathbf{1} - \boldsymbol{h}_2) \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_2} \tag{1.24}$$

où $\odot$ représente le produit matriciel de Hadamard (multiplication terme à terme).

Dans cet exemple, nous utilisons la fonction de coût présentée dans le §1.1.1 et décrite par l'équation Eq. 1.10. Par conséquent, pour une paire $(\boldsymbol{x}, c)$ composée d'un vecteur d'entrée et de la cible associée (ici $c = 0$ ou $c = 1$ pour une classification binaire), nous pouvons associer à $\boldsymbol{x}$ un réel h_2 en sortie du MLP et le coût correspondant :

$$h_2 = \sigma_2(\boldsymbol{W}_2 \times \sigma_1(\boldsymbol{W}_1 \times \boldsymbol{x} + \boldsymbol{b}_1) + b_2) \tag{1.25}$$

et

$$\mathcal{C}(h_2, c) = (c - 1)\ln(1 - h_2) - c \ln h_2 \tag{1.26}$$

Le calcul de la dérivée partielle de la fonction de coût donne alors :

$$\frac{\partial \mathcal{C}}{\partial h_2} = \frac{h_2 - c}{h_2(1 - h_2)} \tag{1.27}$$

et Eq. 1.24 se simplifie en :

$$\frac{\partial \mathcal{C}}{\partial g_2} = h_2 - c \tag{1.28}$$

Finalement, en combinant tous ces éléments nous pouvons écrire les dérivées partielles de la fonction de coût par rapport aux paramètres du réseau de neurones comme suit :

$$\frac{\partial \mathcal{C}}{\partial b_2} = h_2 - c \tag{1.29}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{W}_2} = (h_2 - c) \times \boldsymbol{h}_1^T \tag{1.30}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{b}_1} = (h_2 - c) \times (\mathbf{1} - \boldsymbol{h}_1 \odot \boldsymbol{h}_1) \odot \boldsymbol{W}_2^T \tag{1.31}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{W}_1} = \frac{\partial \mathcal{C}}{\partial \boldsymbol{b}_1} \times \boldsymbol{h}_0^T = \frac{\partial \mathcal{C}}{\partial \boldsymbol{b}_1} \times \boldsymbol{x}^T \tag{1.32}$$

Une fois ces calculs faits pour un certain nombre de paires d'exemples $(\boldsymbol{x}, c)$, nous pouvons utiliser ces gradients pour déterminer de nouveaux paramètres qui permettront de diminuer le coût et donc de mieux réaliser la tâche de classification.

Il est intéressant de noter que dans le cas d'une tâche de classification avec un nombre de classes supérieur à 2, la matrice Jacobienne de la fonction de transfert softmax et le vecteur des dérivées partielles de la fonction de coût par rapport au vecteur de sortie (Eq. 1.11) se combinent pour simplifier l'équation Eq. 1.15 et obtenir une équation de forme similaire à l'équation Eq. 1.28 :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_2} = \boldsymbol{h}_2 - \boldsymbol{c} \tag{1.33}$$

La seule différence est que $\boldsymbol{h}_2$ et $\boldsymbol{c}$ sont des vecteurs de dimension n_{Cl} le nombre de classes et non plus de simples réels.

1.1.2 Réseaux de neurones profonds

Les réseaux de neurones dits "profonds" (Deep Neural Networks en anglais) sont des MLP avec un nombre de couches supérieur à trois. Pendant longtemps les méthodes d'apprentissage pour ce type de réseaux de neurones acycliques ne permettaient pas de converger vers un réseau de neurones performant. Des avancées majeures sur les méthodes d'entraînement et le choix de la fonction de transfert Rectified Linear Unit (ReLU) qui minimise l'impact de la dilution du gradient dans les couches basses du réseaux ont permis d'utiliser des réseaux de neurones de plus en plus gros.

1.1.3 Réseaux de neurones récurrents

Contrairement à un réseau de neurones acyclique (*Feed-Forward Neural Network* - FFNN), un réseau de neurones récurrent (*Recurrent Neural Network* - RNN) est un réseau de neurones dont le graphe de connexion contient au moins un cycle. Comme pour les FFNN, de nombreux types de RNN ont été développés au cours des 30 dernières années tels que les réseaux d'Elman [25], les réseaux de Jordan [26] et les Echo State Networks [27].

Au cours des dernières années, un type de RNN est devenu la norme grâce à ses excellentes performances sur des tâches aussi nombreuses que variées : les réseaux de neurones à base de cellules Long Short-Term Memory (LSTM). Les LSTM étant centraux dans cette thèse le chapitre 2 leur est consacré.

Dans cette partie, nous nous intéressons à la version la plus simple des RNN proposée par Jeff Elman en 1990 et dans laquelle on rajoute des liens à un MLP pour donner en entrée d'une couche du réseau sa propre sortie au pas de temps précédent en plus de

la sortie courante de la couche précédente. Dans le cas où les vecteurs d'entrées $\boldsymbol{x}$ sont indépendants les uns des autres cela n'a pas beaucoup d'intérêt mais dans le cas où l'on a des entrées sous la forme de séquences temporelles ou spatiales cette modification a un très grand impact. En effet, la structure d'un RNN introduit un mécanisme de mémoire des entrées précédentes qui persiste dans les états internes du réseau et peut ainsi impacter toutes ses sorties futures. Avec cette simple modification il est en théorie possible d'approximer n'importe quelle fonction qui transforme une séquence d'entrée en une séquence de sortie donnée avec une précision arbitraire [28]. Ce que ne permet pas le MLP. Le revers de la médaille est que ce type de RNN peut être particulièrement difficile à entraîner (bien que des développements récents améliorent cela, voir [29–31]) et que, sans précautions particulières, le contexte réellement utilisable reste très local.

Description du modèle

Nous détaillons ici la version la plus simple des RNN. Pour définir un tel RNN, il suffit d'ajouter une matrice d'interconnexion $\boldsymbol{V}_j \in \mathbb{R}^{n_j \times n_j}$ au triplet définissant une couche L_j dans le cas d'un MLP pour obtenir une couche L'_j du réseau récurrent.

Pour une séquence de vecteurs d'entrée $\boldsymbol{x}(t)$ avec $t \in [\![1, t_f]\!]$, on obtient une séquence de vecteurs de sortie $\boldsymbol{z}(t)$ avec $t \in [\![1, t_f]\!]$ en initialisant les vecteurs des états internes :

— $\boldsymbol{h}_0(t) = \boldsymbol{x}(t)$, $\forall t \in [\![1, t_f]\!]$,

— et $\boldsymbol{h}_j(0) = \boldsymbol{0}$, $\forall j \in [\![1, N]\!]$.

Ensuite, on applique récursivement pour chaque pas de temps l'ensemble des couches L'_j, $j \in [\![1, N]\!]$ du réseau au vecteur de sortie de la couche précédente de la façon suivante :

— on applique la transformation affine a_j au vecteur de sortie de la couche précédente et on y ajoute la transformation linéaire définie par $\boldsymbol{V}_j$ et appliquée à son propre vecteur de sortie au pas de temps précédent :

$$\boldsymbol{g_j}(t) = \boldsymbol{W}_j \times \boldsymbol{h}_{j-1}(t) + \boldsymbol{V}_j \times \boldsymbol{h}_j(t-1) + \boldsymbol{b}_j \tag{1.34}$$

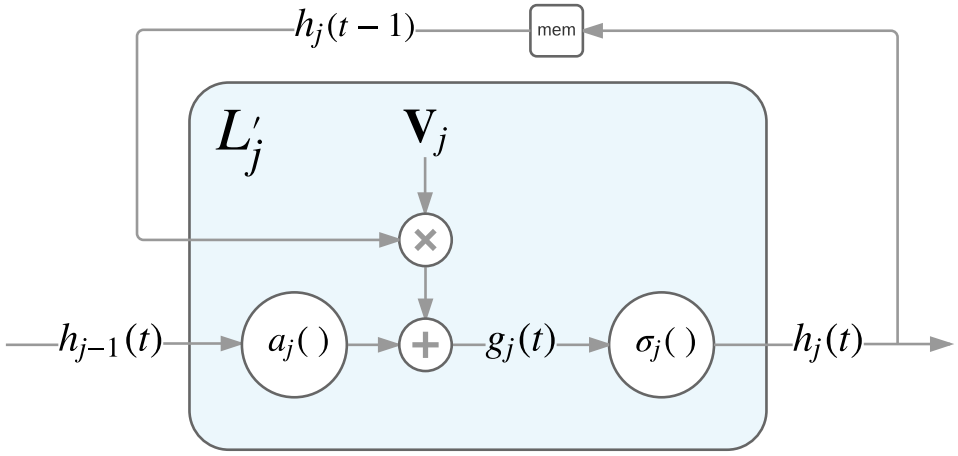
— puis on applique la fonction de transfert au résultat :

$$\boldsymbol{h}_j(t) = \sigma_j\left(\boldsymbol{g}_j(t)\right) \tag{1.35}$$

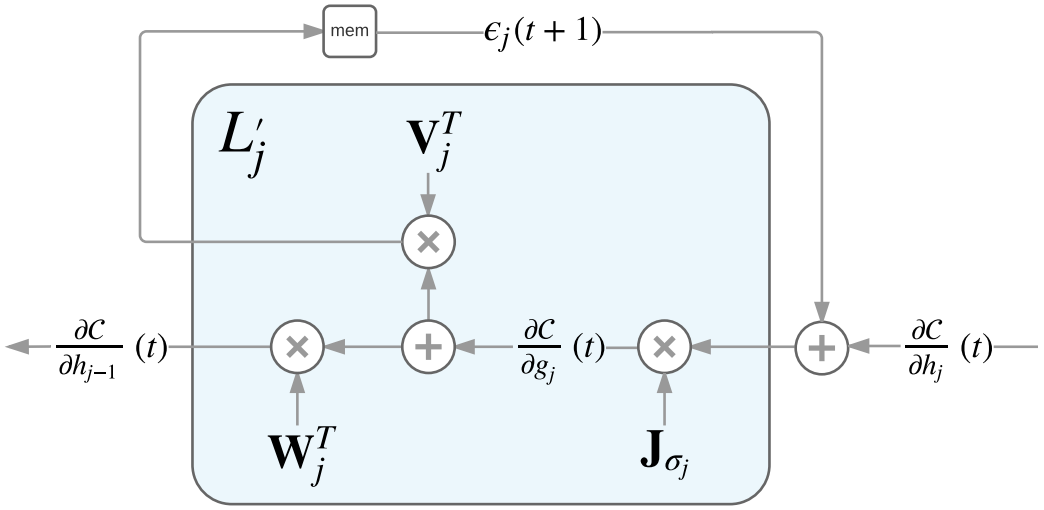
avec $\forall t \in [\![1, t_f]\!]$ et $\forall j \in [\![1, N]\!]$ $\boldsymbol{h}_j(t)$, $\boldsymbol{g}_j(t) \in \mathbb{R}^{n_j}$, $\boldsymbol{z}(t) = \boldsymbol{h}_N(t)$

La figure Fig. 1.7a permet de visualiser l'impact de cette modification sur la propagation de l'information lors de la passe avant dans la couche L'_j.

Pour les tâches de classification de séquences (par exemple, une classe pour toute la séquence), les fonctions de coût et les fonctions de transfert décrites dans le cas du MLP (cf. §1.1.1) peuvent être utilisées de façon similaire. Pour le calcul des erreurs, on peut le faire soit à chaque pas de temps lorsqu'on veut combiner l'ensemble des décisions prises sur la séquence (par exemple avec les moyennes arithmétique ou géométrique), soit à la fin de la séquence uniquement si l'on souhaite n'utiliser que la décision finale.



(a) Graphe computationnel de la passe avant de la couche j d'un RNN



(b) Graphe computationnel de la passe arrière de la couche j d'un RNN

FIGURE 1.7 – Visualisation de la propagation de l'information lors de la passe avant dans la couche j d'un RNN et de la rétro-propagation du gradient de la fonction de coût $\mathcal{C}$ lors de la passe arrière dans cette même couche.

Rétro-propagation des erreurs

Pour un RNN, nous utilisons une version légèrement différente de l'algorithme de rétro-propagation du gradient : la rétro-propagation dans le temps (*Back-Propagation Through Time* - BPTT) proposé par P. Werbos dans [32]. L'algorithme de BPTT consiste à appliquer récursivement les règles de dérivation des fonctions composées pour tous les pas de temps en parcourant le temps à rebours ($t : t_f \rightarrow 1$). Les gradients se rétro-propagent de façon similaire au cas du MLP mais il faut également rétro-propager la contribution aux dérivées partielles des liens récurrents $\boldsymbol{\epsilon}_j(t+1)$ (cf. Fig. 1.7b). Ainsi, la forme de l'équation Eq. 1.15 ne change pas, seuls la dépendance au temps et $\boldsymbol{\epsilon}_j(t+1)$ apparaissent :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) = \boldsymbol{J}_{\sigma_j}(\boldsymbol{g}_j(t)) \times \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j}(t) + \boldsymbol{\epsilon}_j(t+1) \right) \tag{1.36}$$

avec

$$\boldsymbol{\epsilon}_j(t+1) = \boldsymbol{V}_j^T \times \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t+1) \tag{1.37}$$

et $\boldsymbol{\epsilon}_j(t_f+1) = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t_f+1) = \mathbf{0}$ car aucune contribution au gradient ne peut venir après la fin de la séquence.
L'équation Eq. 1.18 quant à elle reste identique à la dépendance au temps près :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_{j-1}}(t) = \boldsymbol{W}_j^T \times \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \tag{1.38}$$

Une fois l'ensemble des pas de temps et des couches du RNN parcourus, nous pouvons déterminer les dérivées partielles qui nous intéressent pour chacune des couches $L'_j, j \in [\![1, N]\!]$ en additionnant les contributions de chacun des pas de temps :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \tag{1.39}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{W}_j} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \times \frac{\partial \boldsymbol{g}_j}{\partial \boldsymbol{W}_j}(t) = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \times (\boldsymbol{h}_{j-1}(t))^T \tag{1.40}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{V}_j} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \times \frac{\partial \boldsymbol{g}_j}{\partial \boldsymbol{V}_j}(t) = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \times (\boldsymbol{h}_j(t-1))^T \tag{1.41}$$

L'algorithme complet de BPTT (Alg. 2) est décrit ci dessous.

Algorithm 2 Calcul du gradient dans un RNN

Choix d'une paire de séquences d'apprentissage $(\boldsymbol{x}(t), \boldsymbol{c}(t))$
Initialisation de $\boldsymbol{h}_0(t) = \boldsymbol{x}(t), \forall t \in [\![1, t_f]\!]$
Passe avant :
for $j = 1..N$ **do**
  for $t = 1..t_f$ **do**
    $\boldsymbol{g_j}(t) = \boldsymbol{W}_j \times \boldsymbol{h}_{j-1}(t) + \boldsymbol{V}_j \times \boldsymbol{h}_j(t-1) + \boldsymbol{b}_j$
    $\boldsymbol{h}_j(t) = \sigma_j\left(\boldsymbol{g}_j(t)\right)$
  end for
end for
Pour chaque pas de temps, on calcule le coût et sa dérivée en fonction de la sortie $\boldsymbol{h}_N(t)$ du réseau de neurone et de la cible correspondante $\boldsymbol{c}(t) : \mathcal{C}(\boldsymbol{h}_N(t), \boldsymbol{c}(t)), \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{h}_N}(t)$

Passe arrière :
for $j = N..1$ **do**
  Initialisation de $\boldsymbol{\epsilon}_j(t_f + 1) = \boldsymbol{0}$ et initialisation des gradients par rapport aux paramètres de la couche j :
  $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{W}_j} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{V}_j} = \boldsymbol{0}$
  for $t = t_f..1$ **do**
    On détermine les dérivées partielles de la fonction de coût par rapport aux paramètres de la couche j au pas de temps t :
    $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) = \boldsymbol{J}_{\sigma_j}(\boldsymbol{g}_j(t)) \times \left(\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j}(t) + \boldsymbol{\epsilon}_j(t+1)\right)$
    $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{b}_j} + \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t)$
    $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{W}_j} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{W}_j} + \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \times (\boldsymbol{h}_{j-1}(t))^T$
    $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{V}_j} = \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{V}_j} + \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t) \times (\boldsymbol{h}_j(t-1))^T$

    Puis on calcule les dérivées partielles de la fonction de coût par rapport à la sortie de la couche précédente $j-1$ au pas de temps t :
    $\dfrac{\partial \mathcal{C}}{\partial \boldsymbol{h}_{j-1}}(t) = \boldsymbol{W}_j^T \times \dfrac{\partial \mathcal{C}}{\partial \boldsymbol{g}_j}(t)$
  end for
end for

1.1.4 RNN bidirectionnel

Dans le cas de nombreuses tâches de classification de séquences, pour prendre une décision à un instant donné il est intéressant de connaître le passé et tout ou partie du futur de la séquence. Par exemple, pour être capable de déterminer quel phonème est prononcé à l'instant t, il est très utile de connaître les phonèmes qui le suivent aussi bien que ceux qui le précèdent. Les RNN que nous avons décrits dans la section précédente traitent les séquences dans l'ordre temporel et n'ont donc pas accès à l'information future.

Pour s'attaquer à ce problème, on peut rajouter de l'information future dans les données d'entrée (par exemple en ajoutant au vecteur courant les 10 prochains vecteurs d'entrée) ou introduire un délai entre un vecteur d'entrée et sa cible pour que le RNN ait le temps de traiter un peu d'information future avant d'avoir à prendre sa décision. Cependant, ces différentes approches rajoutent des contraintes sur l'apprentissage soit en augmentant le nombre de paramètres dans la couche d'entrée, soit en forçant le RNN à apprendre le délai choisi pour donner sa réponse au bon moment. Et dans les deux cas on ne résout pas pour autant le problème puisqu'on introduit des hyper-paramètres (nombre de vecteurs d'entrées à agréger ou délai de réponse) qui peuvent s'avérer difficiles à régler. De plus, on ne supprime en aucun cas la dissymétrie entre le passé et le futur pour ce qui est du contexte exploitable.

En 1997, Schuster et Paliwal ont introduit dans [33] une solution élégante appelée réseau de neurones récurrent bidirectionnel (*bidirectional Recurrent Neural Network* - BiRNN) qui consiste à présenter chaque séquence à traiter à deux RNN de même type mais avec des paramètres différents :

— le premier traite la séquence dans l'ordre naturel $t : 1 \rightarrow t_f$,

— le second traite la séquence dans l'ordre inverse $t : t_f \rightarrow 1$.

Les deux séquences obtenues en sortie des deux RNN sont ensuite concaténées avant d'être mises en entrée d'un MLP qui pour chaque vecteur d'entrée initial produit alors une sortie (par exemple une classification) qui se base sur tout le contexte passé via la sortie du premier RNN et tout le contexte futur via la sortie du second. Il est ainsi possible d'exploiter pleinement les capacités des RNN sur l'ensemble de la séquence pour chaque pas de temps. Depuis les premières expériences probantes sur la structure secondaire des protéines [34], de nombreuses expérimentations (y compris sur la parole) ont démontré que l'utilisation d'un BiRNN est toujours préférable à l'utilisation d'un RNN unidirectionnel ayant le même nombre de paramètres (voir par exemple [1] et [2]). C'est pour cette raison que dans toutes les expérimentations réalisées durant cette thèse nous avons toujours utilisé des BiRNN que ce soit avec des couches récurrentes standard (cf. §1.1.3) ou des couches LSTM (cf. §2.1). Il est à noter que dans le cas où les données sont à traiter en temps réel, l'utilisation d'un RNN bidirectionnel est impossible puisque la fin de la séquence n'est pas accessible. La figure Fig. 1.8 représente l'architecture des BiRNN utilisés dans la quasi-totalité des expériences menées durant cette thèse (nous expliciterons l'architecture alternative lorsque ce n'est pas le cas).

Pour ce type d'architecture, la rétro-propagation du gradient s'effectue de la façon suivante :

— on commence par rétro-propager le gradient à travers le MLP qu'on appelle ici le réseau de sortie pour tous les éléments de la séquence ;

— on récupère alors la moitié supérieure des gradients au niveau de l'entrée du MLP pour les rétro-propager à travers le premier RNN ;

— on fait de même avec la moitié inférieure des gradients au niveau de l'entrée du MLP et le second RNN en pensant bien à rétro-propager les gradients en partant du début de la séquence et non de la fin.

Il est à noter que pour pouvoir réaliser la rétro-propagation du gradient dans les RNN d'une architecture bidirectionnelle, il est nécessaire de conserver en mémoire toutes les activations internes des RNN durant les passes avant et arrière du MLP de sortie. Suivant la taille de la séquence et/ou des RNN cela peut avoir un impact non négligeable sur la quantité de mémoire utilisée lors de l'apprentissage.

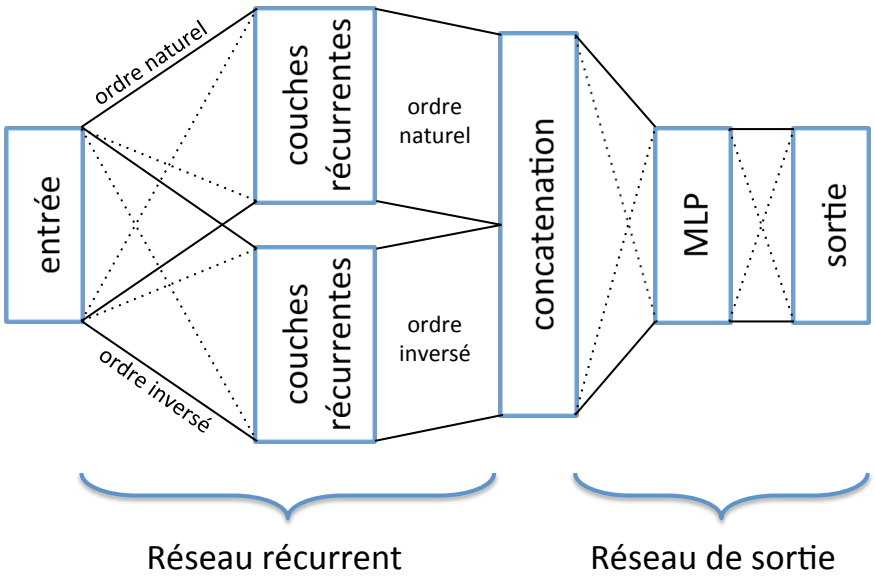


FIGURE 1.8 – Schéma d'un BiRNN. Dans un RNN bidirectionnel la séquence d'entrée est parcourue dans les deux sens par deux RNN distincts dont les séquences de sorties sont combinées avant d'être mises en entrée d'un MLP. Ce dernier peut ainsi prendre, à chaque pas de temps, une décision qui se base sur l'information contenue dans l'ensemble de la séquence d'entrée.

1.2 Détection de la parole

Sous le terme de détection de la parole (ou Speech Activity Detection soit SAD) on fait référence à la séparation des zones d'une séquence audio qui contiennent de la parole des zones qui n'en contiennent pas. Ces dernières peuvent ne contenir que du silence mais elles peuvent également contenir du bruit, de la musique et toutes sortes d'autres signaux acoustiques. Cette tâche de SAD peut sembler simple mais elle peut s'avérer particulièrement difficile dans le cas d'un faible rapport signal sur bruit (ou Signal to Noise Ratio soit SNR) avec un bruit instationnaire. De plus, le champ des applications est très étendu avec des besoins spécifiques pour chacune d'entre elles : réduction de bruit pour les aides auditives [35, 36], reconnaissance automatique de la parole [37, 38], encodage de la parole à taux variable [39–41]. D'autre part, la détection de la parole est également utilisée comme pré-traitement de tous les traitements de plus haut niveau tels que la reconnaissance de la parole, l'identification de la langue ou encore l'identification du locuteur.

Nous nous intéressons dans cette thèse à des tâches liées à la RAP, pour lesquelles la détection de la parole est cruciale. Et bien que le sujet de la détection de parole ait été amplement étudié par le passé, il a fait l'objet d'un regain d'intérêt au cours des dernières années pour des conditions particulièrement difficiles et dans des environnements acoustiques très variés. Il est d'ailleurs au cœur de diverses campagnes d'évaluation internationales telles que le programme RATS de la DARPA, la campagne d'évaluation OpenSAD'15 du NIST et le challenge CHIME organisé par l'université de Sheffield.

Nous présentons dans les paragraphes qui suivent l'architecture d'un système de SAD et les différents algorithmes classiques de SAD que nous avons étudiés. Puis nous détaillons les différentes conditions expérimentales dans lesquelles nous nous sommes placés pour tester nos développements basés sur des ANN et comparer leurs performances à celles des algorithmes plus classiques.

1.2.1 Algorithmes de détection de parole

Un algorithme de SAD peut généralement se scinder en trois traitements distincts (cf. Fig. 1.9). Le premier consiste à calculer, pour chaque fenêtre de signal à classifier, un critère brut de SAD donnant une indication sur la probabilité que cette fenêtre contienne de la parole. Le second traitement consiste à considérer la séquence des critères de SAD calculés lors de la première phase pour déterminer les frontières des segments de parole. Le troisième et dernier traitement consiste à lisser les décisions lorsque cela est nécessaire (suppression des segments trop courts, fusion de segments proches). Dans la suite nous regroupons le lissage et la prise de décisions au sein d'un module unique.

Dans cette partie, nous commençons par présenter des techniques dites "classiques" pour la tâche de SAD. Ensuite, nous présentons la façon dont nous utilisons des ANN pour réaliser cette tâche. Puis nous détaillons l'algorithme de décision et de lissage que nous utilisons quelle que soit la technique de SAD choisie.

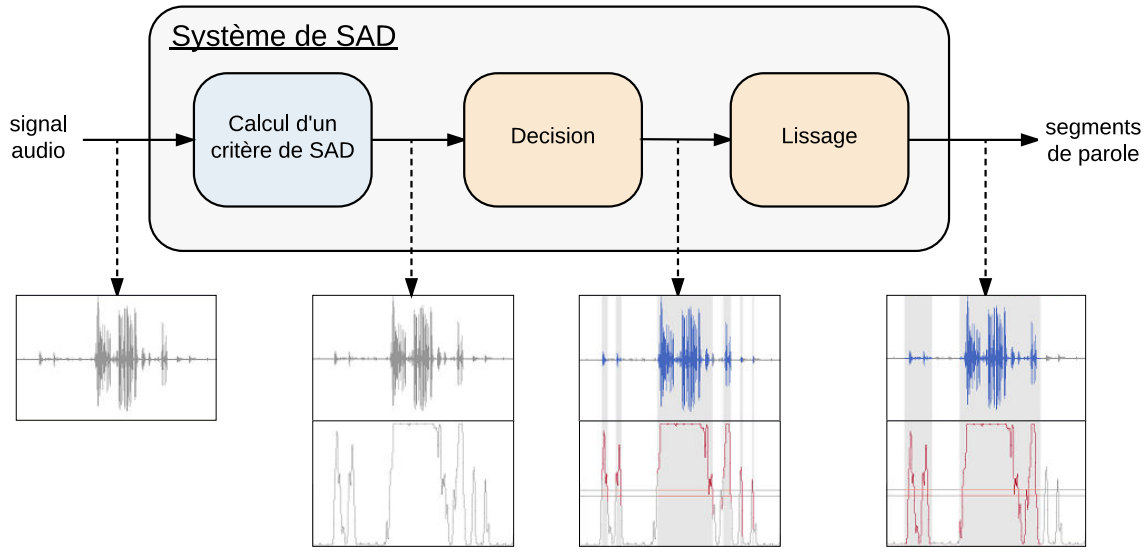


FIGURE 1.9 – Décomposition d'un système de SAD : 1) calcul d'un critère temporel de SAD ; 2) création de segments de parole basés sur ce critère (parties de la figure avec le signal en bleu et le fond grisé) ; 3) lissage des segments de parole (suppression des segments trop courts, fusion de segments proches...).

Techniques classiques

Déterminer un critère robuste de détection de la parole a fait l'objet d'un grand nombre de recherches au cours des dernières décennies. On a pu ainsi voir apparaître diverses méthodes exploitant les caractéristiques spectro-temporelles de la parole mais aussi celles du bruit pour déterminer la présence ou non de parole dans un signal bruité [38, 42, 43]. Plusieurs algorithmes de SAD utilisent des critères basés sur l'énergie du signal [44, 45], d'autres utilisent des coefficients d'auto-corrélation du signal [46–48], la transformation en ondelettes [49], l'entropie [50, 51] ou encore des critères basés sur le degré d'instationnarité du signal à une échelle plus grande [52, 53].

Parmi toutes ces méthodes nous avons sélectionné deux d'entre elles pour servir de point de comparaison lors de nos développements d'un système de détection de la parole fondé sur des ANN : CrossCor et le critère de variabilité du signal sur le long-terme (ou Long-Term Signal Variability soit LTSV).

CrossCor est un critère de SAD qui ajoute à la valeur maximale de la fonction d'auto-corrélation normalisée du signal telle que décrite dans [47], une estimation de la périodicité de la fonction d'auto-corrélation elle-même. Ce faisant, on obtient un critère permettant de mieux distinguer la parole du bruit en particulier à faibles SNR (voir [48]).

Le critère de SAD LTSV a été introduit dans [53] et se base sur une estimation de la variabilité spectrale du signal à partir de l'information fréquentielle sur des plages de temps bien plus grandes que l'usuel fenêtrage de 20 à 30 ms. On obtient ainsi une mesure bien plus réaliste de la variabilité des caractéristiques spectrales du signal qu'avec des fenêtres de 20 ms. Cette mesure peut être efficacement utilisée pour déterminer la présence ou non de parole.

Nous utilisons également un algorithme de SAD développé à la fin des années 1990

[54] comme point de comparaison indépendant puisque nous ne l'optimisons pas pour les différentes tâches que nous traitons. Cela nous permet en effet de quantifier l'intérêt d'une méthode d'optimisation adaptée à la tâche à traiter. Ce SAD que nous appelons *gmmSAD* utilise des GMM construits sur les MFCC pour distinguer le signal contenant de la parole du bruit. Ce *gmmSAD* que nous utilisons à été entraîné sur des centaines d'heures d'audio multilingue.

Techniques neuronales

Les ANN ont déjà été proposés comme méthode pour calculer un critère de SAD [55–57] ou pour servir de mécanisme de fusion de plusieurs critères de SAD plus classiques [58, 59].

Parmi ces techniques basées sur des ANN, les RNN et plus spécifiquement les LSTM présentent des qualités qui les rendent attrayants [60–63]. Ainsi, contrairement au MLP, un RNN ne travaille pas sur un contexte temporel fixé par un hyper-paramètre à ajuster mais sur l'ensemble de la séquence ce qui se traduit par un gain de performance comme on le verra au Chapitre 3.

Algorithme de décision et de lissage

Pour toutes les techniques d'extraction de critère de SAD détaillées dans ce chapitre, nous utilisons le même algorithme pour déterminer les frontières des segments de parole qui constitueront la sortie finale de notre système de SAD. Dans un premier temps, la séquence $z(t)$, $t \in [\![1, t_f]\!]$ obtenue en sortie de l'algorithme de détection de la parole est parcourue pour déterminer le début et la fin des segments de parole. Ainsi, un segment de parole débute lorsque $z(t)$ passe au-dessus du seuil de début de segment θ_d et que l'aire au-dessus du seuil de début est supérieure à S_d. Et, le segment courant se termine lorsque $z(t)$ passe sous le seuil de fin de segment θ_f et que l'aire sous le seuil de fin est supérieure à S_f.

Une fois la première passe terminée, des tampons avant et après les segments sont rajoutés (respectivement de taille δ_- et δ_+). Il peut alors s'avérer nécessaire de fusionner certains d'entre eux si, suite à l'ajout des tampons, le temps de fin d'un des segments devient supérieure au temps de début du segment suivant. Puis, les segments qui ne sont séparés que par un silence dont la durée est inférieure au paramètre δ_0 sont fusionnés. Pour finir, les segments dont la durée est inférieure au paramètre δ_1 sont supprimés. La figure Fig. 1.10 permet de visualiser l'ensemble du processus.

Lors de l'optimisation des systèmes, les huit paramètres (θ_d, S_d, θ_f, S_f, δ_1, δ_-, δ_+ et δ_0) sont adaptés au comportement spécifique de chacune des techniques de SAD.

1.2.2 Conditions expérimentales (SAD)

Durant cette thèse, nous nous sommes intéressés à des tâches de détection de la parole variées autant en termes de difficultés, de langues que d'environnements acoustiques. Nous avons ainsi travaillé sur des tâches de détection pure (c'est-à-dire dont le but est de minimiser le nombre de fenêtres de signal mal classifiées parole/non-parole) et des tâches de détection de la parole pour de la reconnaissance de parole c'est-à-dire de manière à minimiser le WER d'un système de RAP utilisé en aval du système de

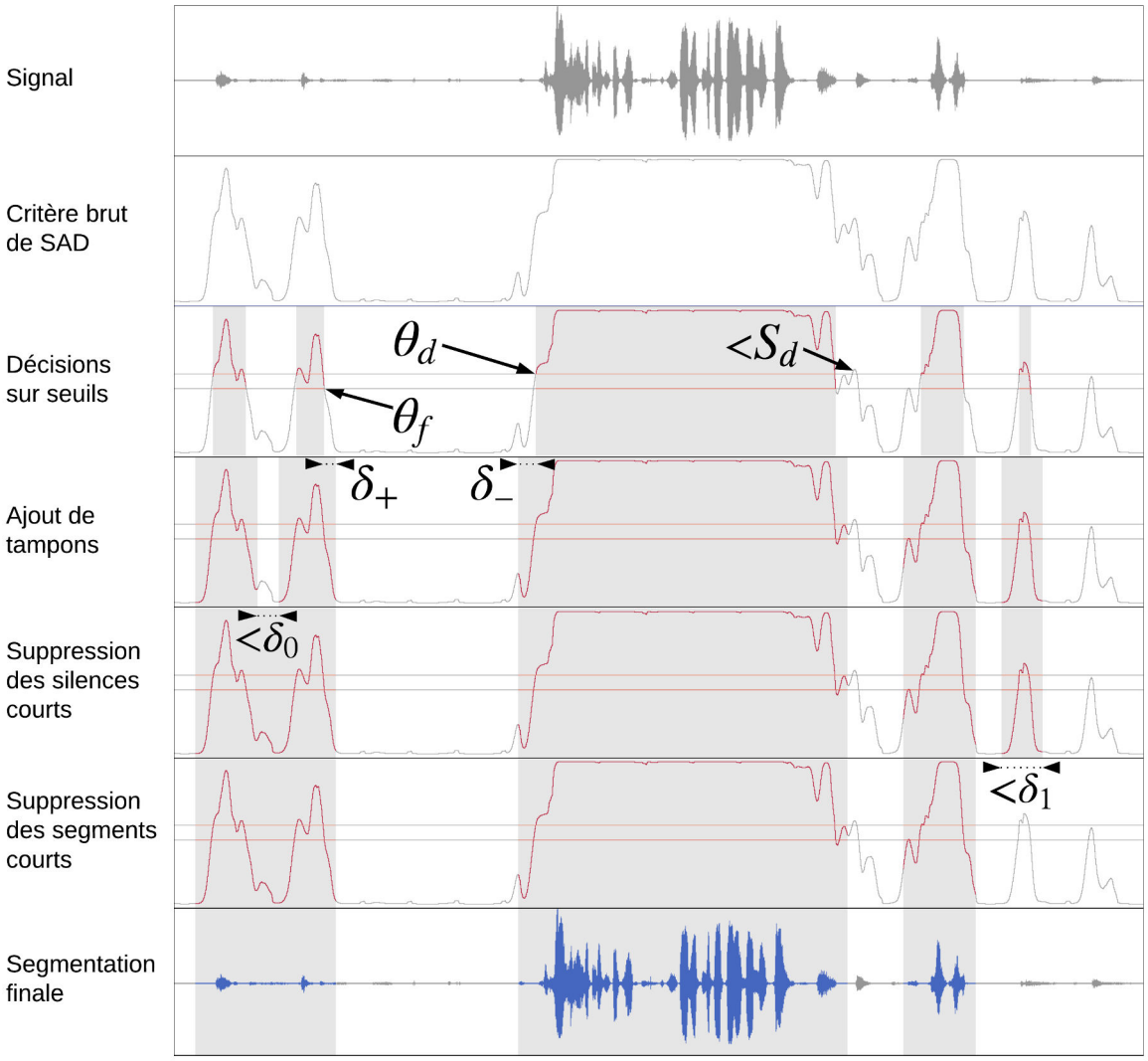


FIGURE 1.10 – Visualisation du processus de décision et de lissage pour déterminer les segments de parole à partir du critère brut de SAD. De haut en bas, nous avons le signal audio à partir duquel un algorithme de détection de la parole détermine un critère brut de détection. Ensuite, ce critère est comparé aux seuils de décision pour déterminer les frontières des segments de parole. Des tampons sont ajoutés avant et après chaque segment. Les segments encadrant un silence trop court sont fusionnés au sein d'un seul grand segment. Pour finir, les segments trop courts sont supprimés pour obtenir la segmentation finale.

détection de la parole. Nous avons travaillé sur des données de types très différents telles que des conversations téléphoniques, des enregistrements de réunions de travail ou des bandes audio de séries télévisées.

Données du programme Babel de l'IARPA

L'IARPA a lancé en 2012 le programme Babel dans le but de développer des technologies de RAP qui puissent être rapidement appliquées à n'importe quelle langue parlée. L'IARPA est partie du constat que la grande majorité des systèmes de RAP ont été d'abord développés pour l'anglais et qu'on observe une dégradation de la performance sur toute autre langue pour laquelle la quantité de ressources disponibles est plus limitée. De plus, le temps de développement d'un système de RAP pour une nouvelle langue est souvent très long (de l'ordre de l'année) et l'ensemble des langues couvertes est relativement restreint. Dans ce contexte, le programme Babel entendait permettre l'émergence de nouvelles méthodes avec pour but ultime de pouvoir développer un nouveau système de RAP rudimentaire pour n'importe quelle langue en l'espace d'une semaine. Les données sur lesquelles les travaux ont été réalisés ont été collectées sur des canaux réalistes, bruités et avec de la parole spontanée. Les développeurs de systèmes de RAP étaient ainsi placés dans des conditions particulièrement intéressantes et difficiles.

Nous avons ainsi pu travailler sur de nombreuses langues telles que le cebuano, le kazakh, le kurde, le lituanien, le telugu et le tok pisin (voir par exemple [64] et [65]). La liste de l'ensemble des langues traitées dans le programme Babel est donnée dans le tableau Tab. 1.1.

Nous présentons dans cette thèse les résultats obtenus sur quatre langues de la première année particulièrement difficiles pour les systèmes de RAP : le vietnamien, le pachto, le turc et le tagalog. Pour ces quatre langues, les jeux de données sont à peu près équilibrés en genre et contiennent de nombreux locuteurs et environnements acoustiques. Pour chaque langue, le jeu d'apprentissage contient 160 heures de signal pour 90 heures de parole environ. Les jeux de validation et de test contiennent quant à eux 30 heures de signal audio pour environ 17 heures de parole. Les versions des jeux de données que nous avons utilisées sont les suivantes : babel107b-v0.7 (vietnamien), babel104b-v0.4 (pachto), babel105b-v0.4 (turc) et babel106b-v0.2 (tagalog).

Pour toutes nos expérimentations, nous avons utilisé des systèmes de RAP à l'état-de-l'art. Ces systèmes sont similaires à ceux décrits dans [66]. Ils utilisent des MLP comme modèles acoustiques et des modèles de langage basés sur des modèles 4-grammes. Ils respectent également la contrainte du NIST pour l'évaluation des systèmes dans le programme Babel qui stipule que l'on ne peut utiliser aucune autre donnée que celles fournies pour la langue visée dans le cadre du programme Babel.

La métrique officielle pour les évaluations que nous avons effectuées est le WER mais nous avons également utilisé le Frame Error Rate (FER) lors d'expériences préliminaires.

Évaluation OpenSAD'15 du NIST

Le NIST organise régulièrement des évaluations ouvertes et internationales sur les différentes tâches de traitement automatique de la parole. En 2015, le NIST a ainsi or-

TABLEAU 1.1 – Liste des langues traitées dans le cadre du programme Babel. Les langues sont regroupées par année de mise à disposition des données.

Année	Langue	Abbréviation
2013	Cantonais	can
	Assamese	ass
	Bengali	ben
	Pachto	pas
	Turc	tur
	Tagalog	tag
	Vietnamien	vie
	Créole haïtien	cre
	Zulu	zul
	Lao	lao
2014	Kurde	kur
	Tok Pisin	tpi
	Cebuano	ceb
	Kazakh	kaz
	Telugu	tel
	Lituanien	lit
	Swahili	swa
2015	Tamil	tam
	Guarani	gua
	Igbo	igb
	Amharique	amh
	Mongol	mon
	Javanais	jav
	Dholuo	dho
	Géorgien	geo

ganisé l'évaluation OpenSAD'15 dont le but était de fournir un cadre aux développeurs de systèmes de SAD permettant de mesurer, par un organisme indépendant, la performance de leurs systèmes sur des données audio particulièrement difficiles. En effet, les signaux audio rassemblés pour cette évaluation proviennent du programme RATS de la DARPA qui s'intéressait essentiellement à des canaux fortement distordus et/ou bruités. Ces différents canaux sont de type HF, VHF et UHF et sont au nombre de 7 (appelés A, B, C, E, F, G et H) avec la particularité que deux de ces canaux (A et C) sont volontairement absents des jeux d'apprentissage et de validation mais présent dans le jeu de test pour évaluer la capacité de généralisation des systèmes de SAD. Tous les jeux de données (apprentissage, validation et test) contiennent des conversations enregistrées dans cinq langues différentes : anglais, arabe syro-libanais, persan, pashto et urdu.

La métrique officielle de l'évaluation était le DCF défini comme suit :

$$DCF = 0.75 \cdot P_{miss} + 0.25 \cdot P_{fa} \tag{1.42}$$

où l'on définit :

$$P_{miss} = \frac{\text{durée de parole étiquetée comme non-parole}}{\text{durée totale de parole}} \tag{1.43}$$

et

$$P_{fa} = \frac{\text{durée de non-parole étiquetée comme parole}}{\text{durée totale de non-parole}} \tag{1.44}$$

Le choix d'une pondération de 75% de la probabilité de manqués par rapport au 25% alloués à la probabilité de fausses alarmes reflète les besoins des utilisateurs finaux du programme RATS (renseignement militaire) pour qui il est important de ne manquer aucun segment de parole quitte à devoir écouter quelques segments ne contenant pas de parole.

Expérimentations sur d'autres données

Nous avons également mené trois expériences sur des données collectées dans des environnements acoustiques très différents de ceux des conversations téléphoniques de l'évaluation OpenSAD'15 et du programme Babel. Pour ces trois expériences le but final était de segmenter en locuteurs. La segmentation en locuteurs consiste à segmenter un signal audio en tours de paroles homogènes c'est-à-dire ne contenant qu'un seul locuteur. La métrique de choix pour cette tâche est le Diarization Error Rate (DER) qui se décompose en deux parties : le FER auquel on ajoute un terme d'erreur correspondant aux confusions entre locuteurs. Par conséquent, pour optimiser un système de segmentation en locuteurs il est préférable de minimiser le FER du système de détection de la parole.

Nous avons travaillé sur une tâche de segmentation en locuteurs dans les flux audio d'émissions de télévision collectés pour la campagne d'évaluation REconnaissance de PERsonnes des Emissions audiovisuelles (REPERE) organisée par le LNE. Comme expliqué dans [67] et [68], la tâche principale de cette campagne était la reconnaissance multimédia de personnes dans des documents télévisuels. Le corpus REPERE est composé de journaux télévisés extraits des chaînes LCP et BFMTV. Les vidéos contiennent

différents types de situations incluant des débats, des reportages et des interviews en plateau. Elles présentent à cet égard une grande variété d'environnements acoustiques (intérieurs/extérieurs, interviews en plateau ou en duplex, reportages avec des voix off, jingles, extraits de films et de concerts...) et donc une réelle difficulté pour la tâche de SAD. Les annotations des corpus de l'évaluation REPERE incluent la transcription manuelle horodatée que nous utilisons pour définir la référence de la tâche de SAD. Dans l'expérience qui est présentée ici nous avons utilisé les corpus d'apprentissage, de développement et de test de la phase 2 de la campagne d'évaluation.

Nous avons également travaillé sur les données du projet européen AMI qui était un consortium multi-disciplinaire de 15 membres dont la mission était la recherche et le développement de technologies permettant d'améliorer les interactions au sein d'un groupe de travail. En particulier, AMI visait à développer un moyen de recherche dans le contenu de réunions qui permettrait d'augmenter l'efficacité du groupe de travail en ayant un meilleur accès à son historique de réunions. Pour permettre de développer cette technologie, AMI a collecté et fait annoter un corpus de réunions de travail qui ont été enregistrées (AMI Meeting Corpus). Ce corpus est composé de 100 heures d'enregistrements avec les annotations en tours de parole. Le corpus est divisé en trois jeux de données : apprentissage, validation et test.

Pour finir, nous avons travaillé sur des données audio issues de séries télévisées. Nous nous sommes appuyés sur le corpus TVD [69] et nous nous sommes focalisés sur la première saison de la série télévisée Game of Thrones (GoT) qui offre des environnements acoustiques particulièrement variés (scènes en intérieur, extérieur, batailles). Ce corpus est ainsi composé de dix épisodes de 55 minutes environ avec les annotations de tours de parole par locuteur. Ces annotations ont été obtenues grâce à l'alignement forcé des transcriptions manuelles avec les pistes audio et peuvent donc être incomplètes voire erronées par moment. Nous utilisons ces annotations comme référence pour la tâche de détection de la parole. Nous utilisons les cinq premiers épisodes comme corpus d'apprentissage, le sixième épisode est utilisé comme corpus de validation et les quatre derniers épisodes de la saison sont utilisés comme corpus de test.

1.3 Identification de la langue

On appelle identification de la langue (IdL) la tâche qui consiste à déterminer la langue parlée dans un segment audio. Cette tâche est cruciale pour de nombreuses applications telles que le routage d'appels, la traduction de parole [70], la reconnaissance de parole multilingue [71] ou encore la recherche multilingue d'information parlée [72].

On estime le nombre de langues parlées dans le monde à un peu moins de 7000 [73], la tâche d'IdL peut donc s'avérer difficile en particulier lorsque l'on s'attache à distinguer des langues proches ou des dialectes d'une même langue. Cela est d'ailleurs le thème principal des dernières campagnes d'évaluation internationales organisées par le NIST.

Comme cela est décrit en détails par Haizhou Li dans [74] des expériences sur l'audition humaine suggèrent qu'il existe deux grandes classes d'indicateurs de la langue : l'information lexicale et les informations sous-lexicales telles que le répertoire phonétique, les contraintes phonotactiques, le rythme et l'intonation [75]. Il a été démontré dans [76] que les deux types d'informations sont utilisés lors du processus cognitif de la reconnaissance de la langue. Cependant la même étude montre que les enfants en bas âge comme les adultes maîtrisant mal une langue utilisent presque exclusivement les informations sous-lexicales pour correctement identifier la langue parlée. Par ailleurs, des études sur la reconnaissance automatique de la langue [77, 78] ont confirmé que les caractéristiques sous-lexicales sont les indicateurs les plus efficaces pour cette tâche. Nous comparons dans cette thèse différentes façons d'exploiter les informations sous-lexicales en particulier à l'aide de réseaux de neurones.

L'un des objectif de cette thèse était de mettre au point système d'IdL basé sur un ANN qui obtienne des performances comparables ou supérieures à celles des systèmes à l'état de l'art. Pour ce faire, nous nous sommes intéressés à deux campagnes d'évaluation organisées en 2007 et 2015 par le NIST. Pour chacune de ces évaluations nous avons comparé et/ou combiné les systèmes basés sur des ANN avec deux systèmes d'IdL plus classiques : un système acoustique basé sur la technique des *i-vectors* et un système phonotactique.

Nous présentons d'abord les algorithmes de référence que nous avons testés, puis nous détaillons les conditions expérimentales dans lesquelles nous nous sommes placés pour tester nos développements basés sur des ANN.

1.3.1 Algorithmes d'identification de la langue

De façon générale, les modèles d'identification de la langue sont construits pour représenter les caractéristiques dépendantes de la langue observées dans le jeu de données d'apprentissage. Ces caractéristiques peuvent être diverses et variées mais nous nous intéressons dans cette thèse aux caractéristiques sous-lexicales que l'on peut extraire à partir de l'analyse spectrale du signal.

Plus spécifiquement, nous nous intéressons à deux approches distinctes. La première que l'on appelle acoustique se base sur une modélisation implicite des caractéristiques dépendantes de la langue directement à partir du spectre. La seconde que l'on appelle phonotactique se base sur une modélisation explicite des contraintes régissant l'enchaînement des phonèmes au sein de chacune des langues cibles.

Dans la suite de cette section, nous présentons deux approches acoustiques et une approche phonotactique. La première approche acoustique appelée *i-vectors* était l'approche acoustique la plus populaire au début des travaux pour cette thèse. La seconde approche acoustique utilise un réseau de neurones récurrent pour modéliser implicitement les caractéristiques discriminantes des différentes langues cibles. L'approche phonotactique utilise quant à elle des modèles n-gramme [1] de phones pour distinguer les langues entre elles.

Pour toutes les approches présentées dans cette thèse, lors de la phase d'identification le modèle utilisé permet, pour chaque séquence audio de test, d'estimer un rapport de vraisemblance par langue cible. La langue ayant le rapport de vraisemblance le plus élevé donne la classification finale.

Approche acoustique par *i-vectors*

Dans les approches acoustiques, on essaye de modéliser la distribution acoustico-phonétique d'une langue sans modélisation explicite des contraintes phonotactiques ou lexicales. Pour ce faire, il existe de nombreuses méthodes mises au point depuis plusieurs décennies comme le montrent les nombreuses publications sur le sujet [79–87].

Nous avons choisi l'approche acoustique la plus répandue au moment de nos travaux comme point de comparaison : la technique des *i-vectors*.

L'approche *i-vectors* a d'abord été mise au point pour des tâches de vérification du locuteur [88–90] puis a très vite été appliquée avec succès à des tâches d'IdL [91, 92]. Cette approche permet de caractériser une langue et un segment audio par un vecteur unique de taille fixe en projetant les données audio sur un *espace de variabilité totale* T sur lequel les informations liées au canal et à la langue sont denses. On l'exprime en général ainsi :

$$S = m + Tw \tag{1.45}$$

où w est appelé *i-vector* et m et S correspondent respectivement aux super-vecteurs GMM pour le modèle du monde indépendant de la langue et pour le modèle de la langue.

Lors d'un test on détermine le *i-vector* correspondant au segment audio dont on souhaite identifier la langue et on le compare aux *i-vectors* obtenus lors de l'apprentissage pour chacune des langues.

Pour améliorer les performances d'un tel système il est recommandé d'utiliser une PLDA [2]. Cette technique appliquée aux *i-vectors* (cf. [94]) permet de décomposer un *i-vector* w en plusieurs composantes :

$$w = \mu_w + \Phi y_s + \Gamma z + \epsilon \tag{1.46}$$

où μ_w est la moyenne des *i-vectors* obtenus sur l'ensemble du jeu d'apprentissage et Φ et Γ sont des matrices rectangulaires représentant respectivement les sous-espaces propres à la voix et au canal. y_s et z sont respectivement appelés le facteur de langue et le facteur de canal, et suivent une loi normale. ϵ représente le bruit résiduel.

1. Un modèle n-gramme de phones est un modèle stochastique à entrées discrètes qui décrit la probabilité d'apparition d'une séquence donnée de n phones dans la langue modélisée.

2. Probabilistic Linear Discriminant Analysis introduite dans [93]

Lors de la phase de test, le score obtenu en comparant le *i-vector* d'une langue w_l et celui du segment audio de test w_{tst} est déterminé par :

$$score(w_l, w_{tst}) = \log \frac{p(w_l, w_{tst}|\theta_{cib})}{p(w_l, w_{tst}|\theta_{non})} \tag{1.47}$$

où θ_{cib} correspond à l'hypothèse que w_l et w_{tst} sont de la même langue et θ_{non} correspond à l'hypothèse qu'ils ne le sont pas.

Pour les expérimentations effectuées dans cette thèse, le système d'IdL basé sur des *i-vectors* prend en entrée 7 MFCC [3] dont C_0. Pareillement à ce qui est présenté dans [91], on effectue d'abord une normalisation de la longueur du conduit vocal puis une normalisation de la moyenne et de la variance. Puis comme dans [96], on calcule les coefficients SDC [4] que l'on concatène aux MFCC normalisés pour ajouter de l'information contextuelle. Finalement, le vecteur d'entrée est de dimension 56.

Pour chacune de nos expérimentations, l'extracteur de i-vector est entraîné sur le jeu d'apprentissage et produit des i-vectors de dimension 600. Ensuite le modèle PLDA est optimisé sur l'ensemble du jeu d'apprentissage avec dix itérations de l'algorithme EM [5].

Ce système d'identification de la langue basé sur des *i-vectors* a été construit à l'aide du toolkit Kaldi [98].

Utilisation des RNN

Au cours des quinze dernières années, les ANN ont été utilisés avec succès au sein de systèmes d'IdL divers et variés. Ils ont ainsi été proposés comme extracteurs de vecteurs caractéristiques à partir du signal [99, 100], comme système d'IdL à part entière [101–103] ou encore comme algorithme de fusion pour combiner les décisions de plusieurs systèmes d'IdL entre eux [104, 105].

Concernant plus particulièrement les RNN, [102] a montré dans une expérience qui se limitait à huit langues que ces modèles surpassaient les autres techniques classiques d'IdL sur les segments courts. Pour cette thèse, nous nous sommes inspirés de ces travaux pour développer un système d'IdL basé uniquement sur des RNN. L'intérêt de cette approche est détaillé dans les chapitres 2, 4 et 5.

Approche phonotactique

Nous présentons ici le système phonotactique d'IdL que nous avons utilisé dans cette thèse comme référence de l'état de l'art.

Les premiers systèmes utilisant les motifs phonotactiques pour distinguer les langues entre elles comparaient la fréquence d'occurrence de certains sons dans les langues cibles [106]. Pour toutes les méthodes de ce type il est donc nécessaire de segmenter le signal de parole en petites unités acoustiques. Pour ce faire on peut utiliser de nombreuses méthodes de reconnaissance de la parole qui permettent de faire correspondre au signal audio une séquence d'unités acoustiques comme par exemple les classes phonétiques ou les phones [106–111].

3. Mel-Frequency Cepstral Coefficients introduits dans [95]
4. Shifted Delta Cepstral
5. Expectation-Maximization proposé par Dempster [97].

Pour cette thèse, le système phonotactique utilisé est celui développé par le LIMSI depuis le début des années 1990 pour identifier la langue parlée mais également les informations non-linguistiques contenues dans la parole [109, 112, 113]. Ce système utilise les phones comme unité de base et des modèles n-gramme pour modéliser les contraintes phonotactiques. Le système actuel utilise l'approche Parallel Phone Recognizer followed by Language Modeling (PPRLM) représentée sur la figure Fig. 1.11 et introduite dans [78] où il est démontré qu'il n'est pas nécessaire d'utiliser un décodeur de phones spécifique pour chaque langue pour correctement estimer les contraintes phonotactiques. Le système du LIMSI utilise trois décodeurs de phones différents qui ont été construits sur des jeux de données dédiés dans les trois langues suivantes : anglais, italien et russe. Ces décodeurs utilisent des modèles acoustiques basés sur des HMM-DNN pour traiter les séquences d'entrées calculées toutes les 10 ms [114, 115]).

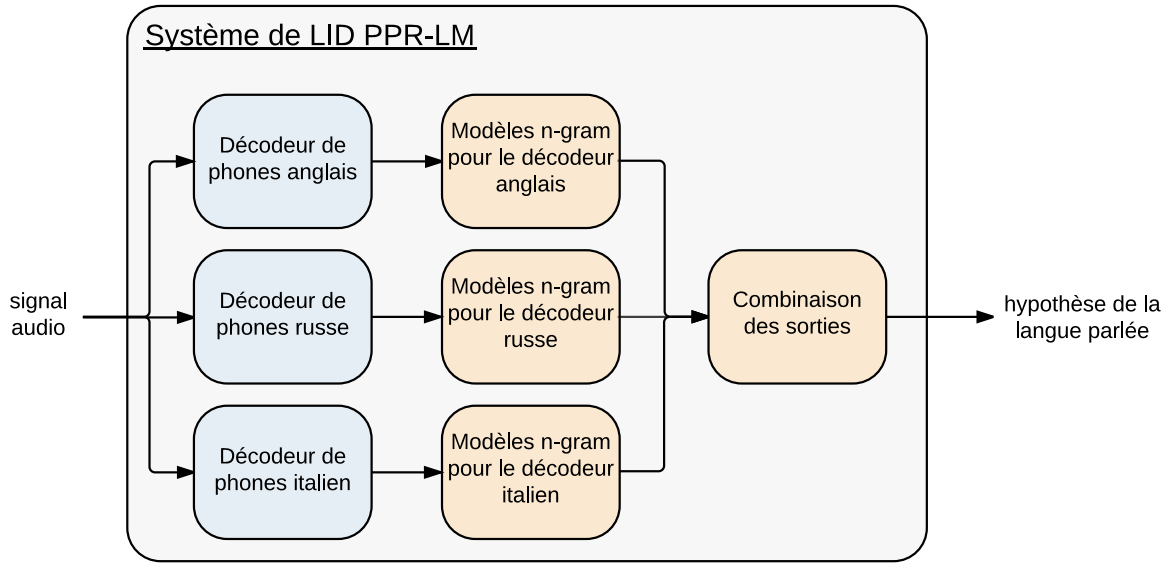


FIGURE 1.11 – Composition du système PPR-LM utilisé dans cette thèse. Ce système utilise trois décodeurs de phones construits sur des langues différentes : anglais, italien et russe. Et pour chacun de ces décodeurs, un jeu de modèles n-grammes modélisant chacune des langues cibles du système de LID.

Ces entrées sont initialement composées de 12 coefficients PLP, des logarithmes de l'énergie et de la fréquence fondamentale. On applique une normalisation de la moyenne et de la variance par segment de parole et on introduit de l'information contextuelle en transformant la concaténation de 9 vecteurs d'entrées centrés sur le pas de temps courant en un vecteur de sortie de dimension 40 grâce à une analyse discriminante linéaire. Ensuite pour réduire l'impact de certaines variabilités comme le locuteur et le canal de communication on utilise une transformation CMLLR[6] non-supervisée. La séquence de vecteurs de dimension 40 qui en résulte est fournie en entrée d'un MLP ayant 4 couches cachées de 2000 neurones chacune et une couche de sortie de dimension

6. Constrained Maximum Likelihood Linear Regression comme introduite dans [116]

allant de 88 à 142 suivant le nombre de phones modélisés dans les HMM à trois états. Le décodage produit ainsi un treillis de phones pour chaque segment de parole et pour chacun des trois décodeurs (anglais, italien et russe).

Ensuite, comme cela est décrit dans [104] et [117], on utilise ces treillis de phones pour calculer les statistiques des n-grammes de phones et ainsi modéliser les contraintes phonotactiques observées dans le signal audio. Le décodage des phones est réalisé sans modèle de langage. Les modèles n-gramme de phones sont alors utilisés pour calculer l'espérance de la vraisemblance pour chaque langue possible (voir [104]). Le résultat final correspond à la moyenne obtenue sur l'ensemble des trois décodeurs.

Fusion de systèmes

La plupart des systèmes d'IdL à l'état de l'art sont en fait composés de multiples sous-systèmes incluant souvent différentes formes d'approches acoustiques et phonotactiques. Il est donc primordial de se doter d'un moyen de combiner convenablement les sorties des différents systèmes pour obtenir une meilleure performance que le meilleur sous-système pris individuellement. C'est ce qu'on appelle la fusion de systèmes.

Il existe de nombreuses manières de réaliser cette fusion (voir [118], [119] et [120]) et durant nos développements, nous avons exploré plusieurs façons de combiner les sorties des différents systèmes d'IdL à notre disposition.

Nous avons tout d'abord utilisé la moyenne géométrique des vecteurs de probabilités a posteriori des différents systèmes [7]. Afin de tirer le meilleur parti de cette combinaison, les sorties de chacun des systèmes sont au préalable calibrées pour maximiser l'entropie croisée normalisée.

Nous avons également essayé de fusionner les sorties des différents systèmes à l'aide d'un MLP pour produire un vecteur de probabilités a posteriori unique. Pour éviter tout problème de sur-apprentissage, ce MLP n'est composé que d'une seule couche de faible dimension (64). La couche de sortie est une couche *softmax* dont la dimension est égale au nombre de langues à identifier. Pour finir, ce MLP est entraîné à l'aide de la fonction de coût d'entropie croisée comme explicité dans la section §1.1.1.

Lors de notre préparation à l'évaluation LRE-15 organisée par le NIST nous avons observé un manque de robustesse de la fusion par MLP pour un gain potentiel faible. Nous ne présentons donc dans ce manuscrit que des résultats de fusion par moyenne géométrique des vecteurs de probabilités a posteriori des différents systèmes.

1.3.2 Conditions expérimentales (IdL)

Cette section présente les conditions expérimentales de deux évaluations internationales organisées par le NIST pour lesquelles il fallait respectivement distinguer 14 langues (LRE-07) et 20 langues et dialectes (LRE-15).

La mise à disposition via des évaluations internationales de corpus de données suffisamment fournis a été l'un des principaux moteurs des développements de systèmes de traitement de la parole et en particulier des systèmes de reconnaissance de la langue parlée. En effet, pour correctement modéliser une langue il est nécessaire d'avoir des

7. La moyenne arithmétique a également été testée mais très vite rejetée puisqu'elle donnait de bien moins bons résultats

données en quantité suffisante pour couvrir la plupart des variabilités intrinsèques à la langue telles que le locuteur, le contenu de la discussion, les canaux d'enregistrement et tous les bruits pouvant parasiter le signal de parole.

Un autre aspect important des évaluations internationales est qu'elles ont permis de standardiser les protocoles d'évaluation des systèmes et ont ainsi créé un espace de discussion dans lequel les différentes technologies peuvent être comparées objectivement. Le NIST, en regroupant des données annotées avec une tâche bien définie, une métrique d'évaluation et un atelier pour discuter des résultats suite à l'évaluation, a ainsi pu créer une émulation au sein de la communauté de recherche en traitement automatique de la parole. En ce qui concerne plus spécifiquement la tâche d'IdL, le NIST a organisé sept évaluations appelées LRE (pour Language Recognition Evaluation) en 1996, 2003, 2005, 2007, 2009, 2011 et 2015. Les évaluations LRE organisées par le NIST avaient pour buts affichés de favoriser la recherche pour :

— explorer de nouvelles et prometteuses idées en reconnaissance de la langue,
— développer des technologies de pointe basées sur ces idées,
— mesurer objectivement la performance de ces technologies pour pouvoir les comparer.

Avec chaque évaluation, le panel de langues couvert a augmenté passant de 12 langues en 1996 à une vingtaine de langues/dialectes dans les deux dernières évaluations.

Dans cette thèse, nous nous sommes d'abord intéressés à l'évaluation de 2007 puisqu'elle avait été étudiée en profondeur au LIMSI avant ces travaux et qu'on disposait ainsi d'un très bon système phonotactique pour nous servir de référence. Puis nous avons eu l'opportunité de participer à l'évaluation de 2015.

Pour toutes nos expérimentations sur l'identification de la langue nous avons utilisé trois métriques différentes : la métrique Cavg tel que définie dans [121, 122] et qui est la référence du NIST pour ses évaluations, le taux d'égale erreur (ou Equal Error Rate) et le LER moyen défini comme suit :

$$\mathrm{LER} = \frac{1}{N} \sum_{c \in C} \left(\frac{1}{N_C} \sum_{d \in D_c} \mathrm{Perr}(d) \right) \tag{1.48}$$

où C est l'ensemble des groupes de langues, N est le nombre de groupes, D_c est l'ensemble des langues/dialectes du groupe c et N_C est le nombre de langues/dialectes contenus dans le groupe c (pour LRE-07, il n'y a qu'un seul groupe de langues contenant toutes les langues de l'évaluation).

Évaluation LRE-07

L'évaluation LRE-07 s'attachait à mesurer la difficulté de distinguer automatiquement les 14 langues listées dans le tableau Tab. 1.2. Les données proviennent exclusivement de conversations téléphoniques. Dans toutes les expériences que nous présentons dans cette thèse, nous nous plaçons dans les conditions imposées aux participants de l'évaluation qui stipulaient que les systèmes d'IdL ne pouvaient être entraînés que sur les données fournies par le NIST. La seule exception à cette règle concerne les modèles acoustiques HMM-DNN du système phonotactique qui avaient été entraînés au préalable sur d'autres données en anglais, russe et italien. En revanche, les modèles n-gramme de ce système ont eux été construits uniquement sur les données de LRE-07.

TABLEAU 1.2 – Répartition des données d'apprentissage et de test par langue pour l'évaluation LRE-07 organisée par le NIST. Pour chacun des fichiers de test, le NIST a isolé trois extraits contenant respectivement 3 s, 10 s et 30 s de parole et c'est sur ces trois extraits que sont mesurées les performances des systèmes de LID.

Code	Langue	Nbre de fichiers		Durée de parole (h)
		appr.	test	appr.
ara	Arabe	80	80	15.1
ben	Bengali	40	80	4.1
chi	Chinois	524	398	73.9
eng	Anglais	342	240	54.2
far	Persan	40	80	11.1
ger	Allemand	82	80	15.7
hin	Hindi	152	240	23.3
jap	Japonais	224	80	34.1
kor	Coréen	202	80	29.5
rus	Russe	40	160	3.9
spa	Espagnol	210	240	39.7
tam	Tamil	132	160	20.2
tha	Thaï	40	80	4.2
vie	Vietnamien	40	160	13.2

Évaluation LRE-15

L'évaluation LRE-15 est très similaire aux évaluations de LRE-09 et LRE-11 et s'attache à mesurer la difficulté de distinguer automatiquement des langues proches ou les dialectes d'une même langue. Les données proviennent essentiellement de conversations téléphoniques et quelques interventions téléphoniques retransmises lors d'émissions télévisées. Cette seconde source de données permet surtout d'assurer une plus grande variabilité au sein des données puisque les segments audio correspondants sont souvent plus courts et avec des locuteurs toujours différents. Pour cette évaluation les données sont regroupées en six groupes de langues (arabe, chinois, anglais, français, ibérique et slave) qui contiennent au total 20 langues ou dialectes.

Comme pour LRE-07, dans toutes les expériences que nous présentons dans cette thèse, nous nous plaçons dans les conditions imposées aux participants de l'évaluation qui stipulaient que les systèmes d'IdL ne pouvaient être entraînés que sur les données fournies par le NIST. Pour le système phonotactique, les modèles acoustiques HMM-DNN ont toutefois été entraînés au préalable sur d'autres données (en anglais, russe et italien) mais les modèles n-gramme ont eux été construits uniquement sur les données de LRE-15.

La table Tab. 1.3 donne le détail des langues de l'évaluation LRE15 et des quantités de données en nombre de fichiers et en temps de parole disponibles pour chacune des langues et dialectes. Comme on peut le voir dans la table certaines langues (comme par exemple l'anglais britannique, le brésilien et le cantonais) sont très peu dotées en données d'apprentissage. Nous avons donc décidé d'utiliser la technique d'augmentation de données décrite dans la section §6.2.6 (p. 117) pour améliorer la robustesse des modèles.

Pour l'évaluation officielle, aucun jeu de développement n'ayant été fourni aux participants et nous avons sélectionné 10% des données d'apprentissage pour composer notre propre jeu de données de développement. A partir de ces fichiers et en utilisant la technique d'augmentation de données, nous avons créé deux jeux de développement : un premier contenant 1180 segments audio contenant entre 10 s et 50 s de parole avec une moyenne de 27.6 s, et un second contenant 7500 segments audio contenant entre 3 s et 10 s de parole avec une moyenne de 4.5 s. Le but de ces deux jeux de développement était de s'assurer du bon comportement de nos systèmes sur des segments longs et sur des segments courts puisque nous ne savions pas, pendant nos développements, quelle serait la distribution des temps de parole par fichier dans le jeu de test.

Il est important de remarquer que certaines des langues sont très faiblement représentées dans les jeux d'apprentissage et de développement. On peut alors regretter que la répartition des données entre les jeux d'apprentissage et de test n'ait pas été plus judicieusement réalisée compte tenu de la quantité globale de données disponibles y compris pour ces langues (cf. Tab.1.3).

Concernant le jeu de test officiel, les segments audio contiennent des durées de parole comprises entre 1 s et 82 s avec plus du tiers des fichiers contenant moins de 5s de parole comme on peut le voir sur la figure Fig. 1.12. Compte tenu du nombre de fichiers et de leurs durées de parole, le jeu de test est beaucoup plus varié que notre jeu de développement voire même que le jeu d'apprentissage avec en plus une proportion de segments très courts bien plus importante.

Tout ces éléments rendent difficile l'analyse des résultats sur le jeu de test officiel,

c'est pourquoi nous avons décidé, après l'évaluation officielle, de constituer un nouveau jeu d'apprentissage "post-évaluation" contenant 10% du jeu de test officiel et de conserver les 90% restants pour le test.

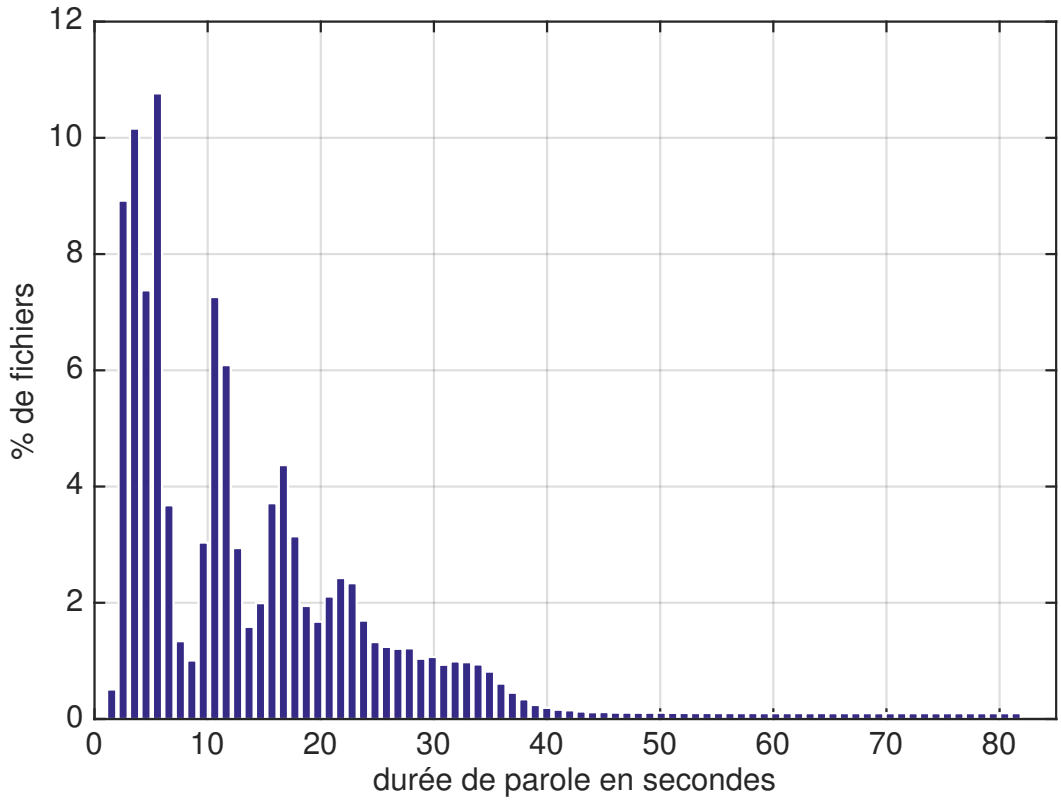


FIGURE 1.12 – Distribution des segments audio dans le jeu de test officiel de LRE15 en fonction de la durée de parole.

Tableau 1.3 – Répartition des données d'apprentissage et de test par groupe de langues et par langue pour l'évaluation LRE-15. On y retrouve les 6 groupes de langues : arabe, chinois, anglais, français, slave et ibérique. Il est à noter que si les groupes chinois et arabe regroupent bien des dialectes, les groupes anglais, français et ibérique regroupe plutôt des accents. On notera également que le groupe slave contient deux langues tout à fait distinctes et que le portugais n'est pas un dialecte de l'espagnol mais bien une autre langue. Par ailleurs, certains des dialectes sont très faiblement représentés dans les jeux d'apprentissage (par exemple, anglais britannique et portugais). On peut alors regretter que la répartition des données entre les jeux d'apprentissage et de test n'ait pas été réalisée plus judicieusement compte tenu de la quantité globale de données disponibles.

Code	Groupe - Langue	Nbre de fichiers		Durée de parole (h)	
		appr.	test	appr.	test
ara-arz	Arabe - Égyptien	220	8023	95.4	41.8
ara-acm	Arabe - Irakien	210	8994	37.2	47.7
ara-apc	Arabe - Levantin	225	6802	41.1	38.4
ara-ary	Arabe - Maghrébin	207	8264	38.6	46.3
ara-arb	Arabe - Standard	406	2447	3.7	8.2
zho-yue	Chinois - Cantonais	17	22532	3.4	151.9
zho-cmn	Chinois - Mandarin	219	6026	71.8	30.4
zho-cdo	Chinois - Min Dong	37	8542	8.1	47.1
zho-wuu	Chinois - Wu	36	7496	7.7	41.2
eng-gbr	Anglais - Britannique	47	7998	0.5	27.1
eng-usg	Anglais - Américain	214	6980	100.0	33.8
eng-sas	Anglais - Indien	392	6932	8.1	35.4
fre-waf	Français - Afrique de l'ouest	34	6935	7.7	38.3
fre-hat	Français - Haïtien	323	28741	2.7	168.6
qsl-pol	Slave - Polonais	363	4818	30.0	17.0
qsl-rus	Slave - Russe	386	3051	18.0	11.4
spa-car	Ibérique - Esp. des Caraïbes	60	2332	26.9	12.2
spa-eur	Ibérique - Esp. Européen	38	5803	8.1	25.3
spa-lac	Ibérique - Esp. d'Am. Latine	30	6973	6.9	30.3
por-brz	Ibérique - Portugais Brésilien	47	4645	0.8	15.3

Chapitre 2

LSTM augmenté (CG-LSTM)

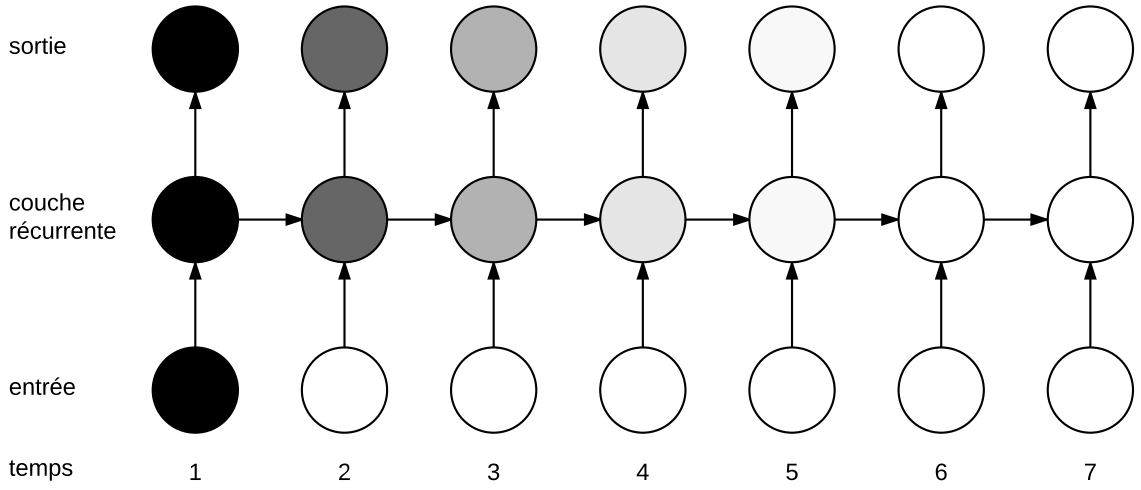
L'intérêt des RNN réside dans leur capacité à exploiter l'information contextuelle pour passer d'une séquence d'entrée à une séquence de sortie qui soit le plus proche possible de la séquence cible. Malheureusement, pour les RNN standards l'apprentissage peut se révéler difficile et le contexte réellement exploité très local. Le problème vient du fait qu'un vecteur d'entrée ne peut avoir une influence sur les décisions futures qu'au travers des liens récurrents (et donc via la multiplication répétée par la matrice $\boldsymbol{V}_j$ (cf. §1.1.3 p. 16) et l'application répétée de la fonction d'activation) et que par conséquent cette influence décroit ou augmente exponentiellement au fur et à mesure qu'on avance dans la séquence (voir Fig. 2.1a). Ce phénomène est souvent appelé *vanishing gradient problem* dans la littérature [123, 124] parce qu'il impacte la rétro-propagation du gradient.

Dans les années 1990, de nombreuses architectures neuronales et méthodes d'apprentissage ont été testées pour essayer de contrer ce phénomène. Parmi ces méthodes on trouve des méthodes d'optimisation heuristiques qui n'utilisent pas le gradient telles que les méthodes de recuit simulé ou de propagation d'erreurs discrètes [123], l'introduction de retards explicites dans l'architecture récurrente [7, 125] ou encore la compression hiérarchique de séquences [126]. Mais l'approche qui s'est montrée la plus efficace et qui est maintenant devenue la norme pour traiter des séquences est le modèle LSTM proposé par Hochreiter et Schmidhuber en 1997 [127]. En effet, ce modèle introduit des portes logiques multiplicatives qui permettent de conserver et d'accéder à l'information pertinente sur de longs intervalles permettant ainsi de diminuer l'impact du problème de gradient évanescent (voir Fig. 2.1b).

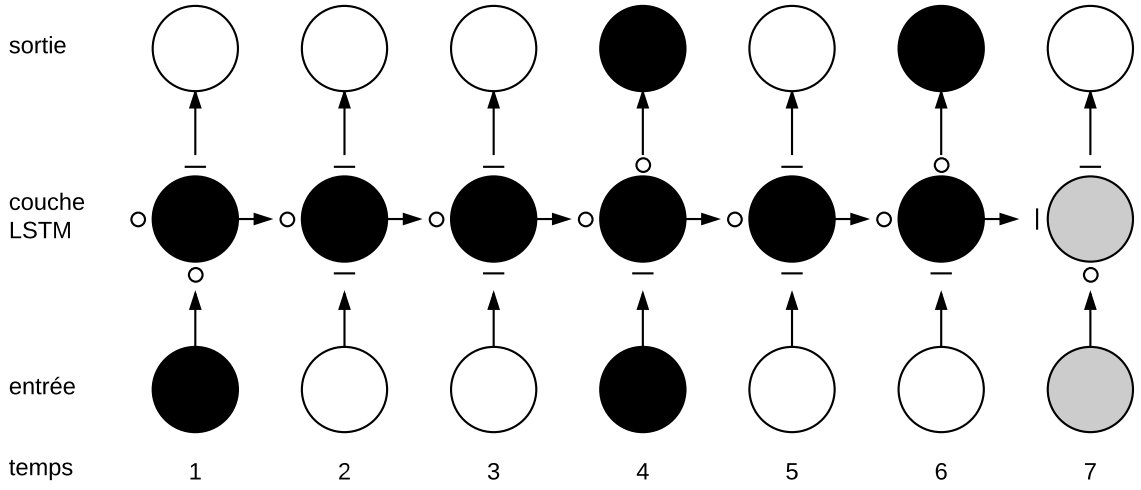
C'est à ce type de modèle neuronal que nous nous intéressons principalement dans cette thèse et nous décrivons dans cette section le modèle LSTM standard avec l'ajout de "peepholes" tels qu'ils furent introduits par F. Gers en 2002 [128].

Ensuite nous introduisons la première contribution de cette thèse qui est une augmentation du modèle LSTM en ajoutant des liens directs entre les 3 types de portes (entrée, oubli et sortie) pour qu'elles puissent mieux coordonner leurs comportements. On appelle ce nouveau modèle Long Short-Term Memory with Coordinated Gates (CG-LSTM).

Nous avons introduit cette modification dans le but de limiter un comportement gênant qui apparaissait lors de l'apprentissage avec le modèle LSTM standard sur des séquences longues. Un certain nombre (jusqu'à 50%) des cellules LSTM dans une



(a) Problème du gradient évanescent dans un RNN standard. On représente ici, dans sa forme dépliée dans le temps, une couche RNN ayant un unique "neurone" et sa sensibilité à l'information qui est mise en entrée du RNN au pas de temps 1. Sur cette figure, plus le neurone est sombre plus il est sensible à l'entrée du pas de temps 1. Dans un RNN standard, cette sensibilité décroit exponentiellement avec le temps et l'arrivée de nouvelles entrées qui viennent effacer la mémoire de cette première entrée.



(b) Préservation de l'information (et du gradient) dans une couche LSTM. L'état des portes d'entrée, d'oubli et de sortie sont représentées respectivement en dessous, à gauche et au dessus de l'état interne de la couche LSTM. Pour plus de lisibilité, les portes sont représentées soit complètement ouvertes ("o") soit complètement fermées ("-"). Ainsi, l'état interne peut conserver l'information intacte sur plusieurs pas de temps et ce tant que la porte d'oubli est ouverte et la porte d'entrée est fermée. La sensibilité de la couche de sortie peut également varier sans impacter l'état interne par l'action de la porte d'oubli.

FIGURE 2.1 – Différences de capacité d'exploitation du contexte entre une couche RNN standard et une couche LSTM

couche se retrouvaient dans un état de saturation constante (sortie constante à 0 ou à 1 dès les premiers pas de temps). Nous espérions donc atténuer ce phénomène en introduisant de nouveaux liens entre les différents types de portes pour permettre une meilleure interaction. Cette modification n'a pas permis de contrer le phénomène de saturation observé avec le modèle LSTM standard. En revanche, ce nouveau modèle nous a permis d'obtenir de meilleures performances sur différentes tâches. Comme par ailleurs le coût de cette modification sur le temps de calcul est négligeable nous avons décidé de privilégier l'utilisation de ce modèle dans nos travaux.

Il est à noter que nous avons pu résoudre le problème de saturation des cellules LSTM en limitant la durée des séquences audio vue par le modèle lors de l'apprentissage et du test (cf. section §6.2.5).

2.1 Description des modèles LSTM

Nous détaillons ici le modèle neuronal récurrent le plus fréquemment utilisé aujourd'hui : le modèle à mémoire court-terme persistante plus connu sous son acronyme anglais LSTM. Puis nous présentons la version augmentée que nous introduisons dans cette thèse.

2.1.1 LSTM standard

Description du modèle

Une couche de LSTM L''_j est composée de quatre couches RNN telles que définies dans la section 1.1.3 et qui interagissent entre elles. Trois de ces couches RNN ont pour fonction de transfert la fonction *logistique* (et donc des sorties entre 0 et 1) et agissent comme des portes logiques contrôlant :

— la quantité d'information qui entre dans les cellules LSTM de la couche j,

— la quantité d'information qui est retenue par l'état interne des cellules d'un pas sur l'autre,

— la quantité d'information qui sort des cellules LSTM de la couche j.

La dernière des quatre couches fonctionne comme une couche RNN standard et alimente l'état interne des cellules LSTM via les portes d'entrée (voir la figure Fig. 2.2 pour une description synthétique des flux d'information). Finalement, pour modéliser les retours entre l'état interne des cellules et les 3 portes logiques introduits par F. Gers, il faut ajouter 3 vecteurs de paramètres $\boldsymbol{u}_i, \boldsymbol{u}_f, \boldsymbol{u}_o \in \mathbb{R}^{n_j}$ appelés "peepholes". La figure Fig. 2.3 représente le fonctionnement détaillé d'une couche LSTM.

Ainsi, pour une séquence de vecteurs d'entrée $\boldsymbol{x}(t)$ avec $t \in [\![1, t_f]\!]$, on obtient une séquence de vecteurs de sortie $\boldsymbol{z}(t)$ avec $t \in [\![1, t_f]\!]$ en appliquant récursivement pour chaque pas de temps l'ensemble des couches L''_j, $j \in [\![1, N]\!]$ du réseau au vecteur de sortie de la couche précédente. Pour améliorer la lisibilité, nous omettons l'indice $_j$ indiquant que nous travaillons sur les paramètres de la couche L''_j pour tous les calculs internes à la couche j. La sortie de la couche L''_j est alors déterminée de la façon suivante :

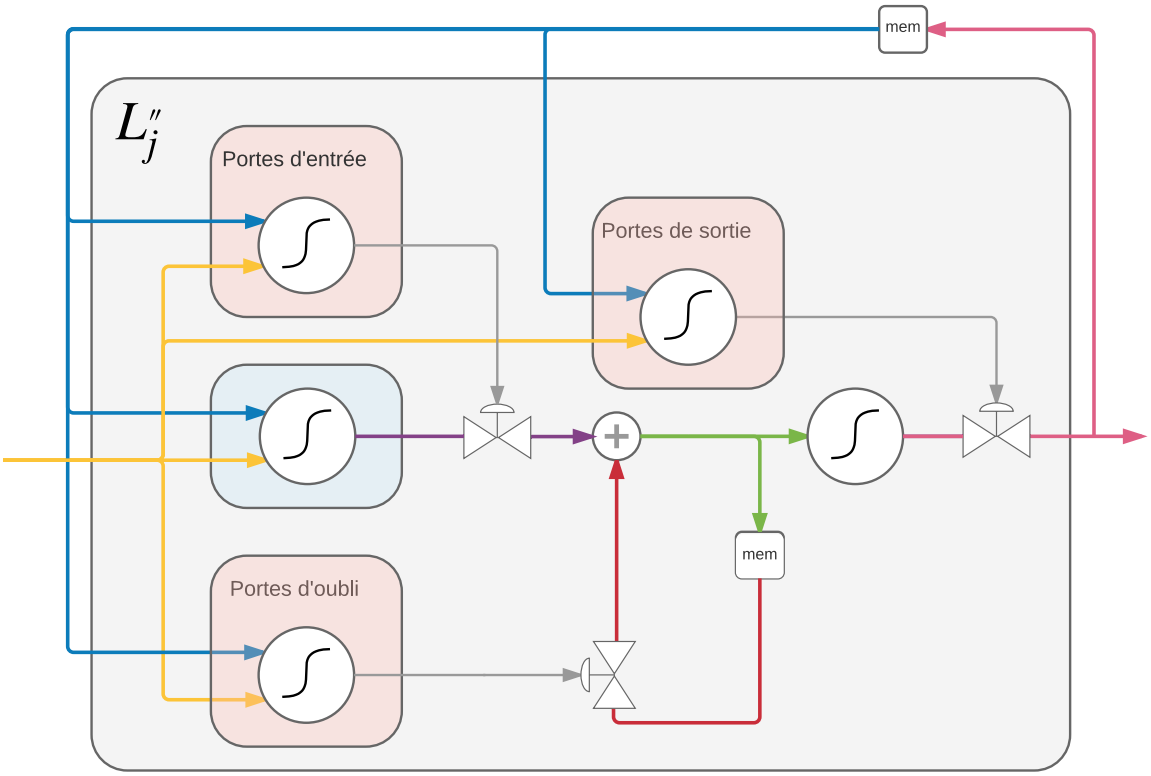


FIGURE 2.2 – Visualisation synthétique de la propagation de l'information lors de la passe avant dans une couche LSTM (sans "peepholes" pour faciliter la lisibilité). Les 3 portes agissent directement sur les vannes qui contrôlent le débit d'information qui provient de l'entrée, celui qui provient de la mémoire interne, mais également l'information qui sort de la couche.

— on commence par déterminer l'état des portes d'entrée en appliquant la transformation affine a_i définie par $\boldsymbol{W}_i$ et $\boldsymbol{b}_i$ au vecteur de sortie de la couche précédente, on y ajoute la transformation linéaire définie par $\boldsymbol{V}_i$ du vecteur de sortie de la couche courante j au pas de temps précédent et on ajoute la contribution du vecteur d'états internes de la couche LSTM au pas de temps précédent via les "peepholes" :

$$\boldsymbol{g}_i(t) = \boldsymbol{W}_i \cdot \boldsymbol{h}_{j-1}(t) + \boldsymbol{V}_i \cdot \boldsymbol{h}_j(t-1) + \boldsymbol{u}_i \odot \boldsymbol{s}(t-1) + \boldsymbol{b}_i \tag{2.1}$$

$$\boldsymbol{i}(t) = \sigma_i\left(\boldsymbol{g}_i(t)\right) \tag{2.2}$$

où $\odot$ correspond à la multiplication terme à terme (produit matriciel de Hadamard) pour que chacune des portes d'entrée de la couche j n'ait accès qu'à la composante des états internes sur laquelle elle a une influence.

— on fait de même pour déterminer l'état des portes d'oubli :

$$\boldsymbol{g}_f(t) = \boldsymbol{W}_f \cdot \boldsymbol{h}_{j-1}(t) + \boldsymbol{V}_f \cdot \boldsymbol{h}_j(t-1) + \boldsymbol{u}_f \odot \boldsymbol{s}(t-1) + \boldsymbol{b}_f \tag{2.3}$$

$$\boldsymbol{f}(t) = \sigma_f\left(\boldsymbol{g}_f(t)\right) \tag{2.4}$$

— ensuite on détermine le nouvel état interne de la couche j en pondérant le nouveau vecteur par les valeurs des portes d'entrée et on ajoute le vecteur d'états internes du pas précédent pondéré par les valeurs des portes d'oubli :

$$\boldsymbol{g}_s(t) = \boldsymbol{W}_s \cdot \boldsymbol{h}_{j-1}(t) + \boldsymbol{V}_s \cdot \boldsymbol{h}_j(t-1) + \boldsymbol{b}_s \tag{2.5}$$

$$\boldsymbol{s}(t) = \boldsymbol{f}(t) \odot \boldsymbol{s}(t-1) + \boldsymbol{i}(t) \odot \sigma_s\left(\boldsymbol{g}_s(t)\right) \tag{2.6}$$

— on détermine alors l'état des portes de sortie de façon similaire aux portes d'entrée et d'oubli mais en utilisant l'état interne courant :

$$\boldsymbol{g}_o(t) = \boldsymbol{W}_o \cdot \boldsymbol{h}_{j-1}(t) + \boldsymbol{V}_o \cdot \boldsymbol{h}_j(t-1) + \boldsymbol{u}_o \odot \boldsymbol{s}(t) + \boldsymbol{b}_o \tag{2.7}$$

$$\boldsymbol{o}(t) = \sigma_o\left(\boldsymbol{g}_o(t)\right) \tag{2.8}$$

— et pour finir on détermine la sortie $\boldsymbol{h}_j(t)$ de la couche j au pas de temps t en pondérant l'état interne courant par les valeurs des portes de sortie :

$$\boldsymbol{h}_j(t) = \boldsymbol{o}(t) \odot \sigma_h\left(\boldsymbol{s}(t)\right) \tag{2.9}$$

Avec ces notations, les vecteurs $\boldsymbol{i}(t)$, $\boldsymbol{f}(t)$, $\boldsymbol{s}(t)$ et $\boldsymbol{o}(t)$ correspondent respectivement aux vecteurs d'activation des :

— *portes d'entrée* qui contrôlent la quantité d'information qui entre dans la couche LSTM,

— *portes d'oubli* qui contrôlent la quantité d'information mémorisée,

— *états internes* de la couche LSTM,

— *portes de sortie* qui contrôlent la quantité d'information qui sort de la couche LSTM.

et on a $\forall t \in [\![1, t_f]\!]$ et $\forall j \in [\![1, N]\!]$:

— $\boldsymbol{h}_j(t),\ \boldsymbol{g}_i(t),\ \boldsymbol{g}_f(t),\ \boldsymbol{g}_s(t),\ \boldsymbol{g}_o(t),\ \boldsymbol{i}(t),\ \boldsymbol{f}(t),\ \boldsymbol{s}(t),\ \boldsymbol{o}(t) \in \mathbb{R}^{n_j}$

— $\boldsymbol{h}_j(0) = \boldsymbol{s}(0) = \boldsymbol{0}$

— $\boldsymbol{h}_0(t) = \boldsymbol{x}(t)$ et $\boldsymbol{z}(t) = \boldsymbol{h}_N(t)$

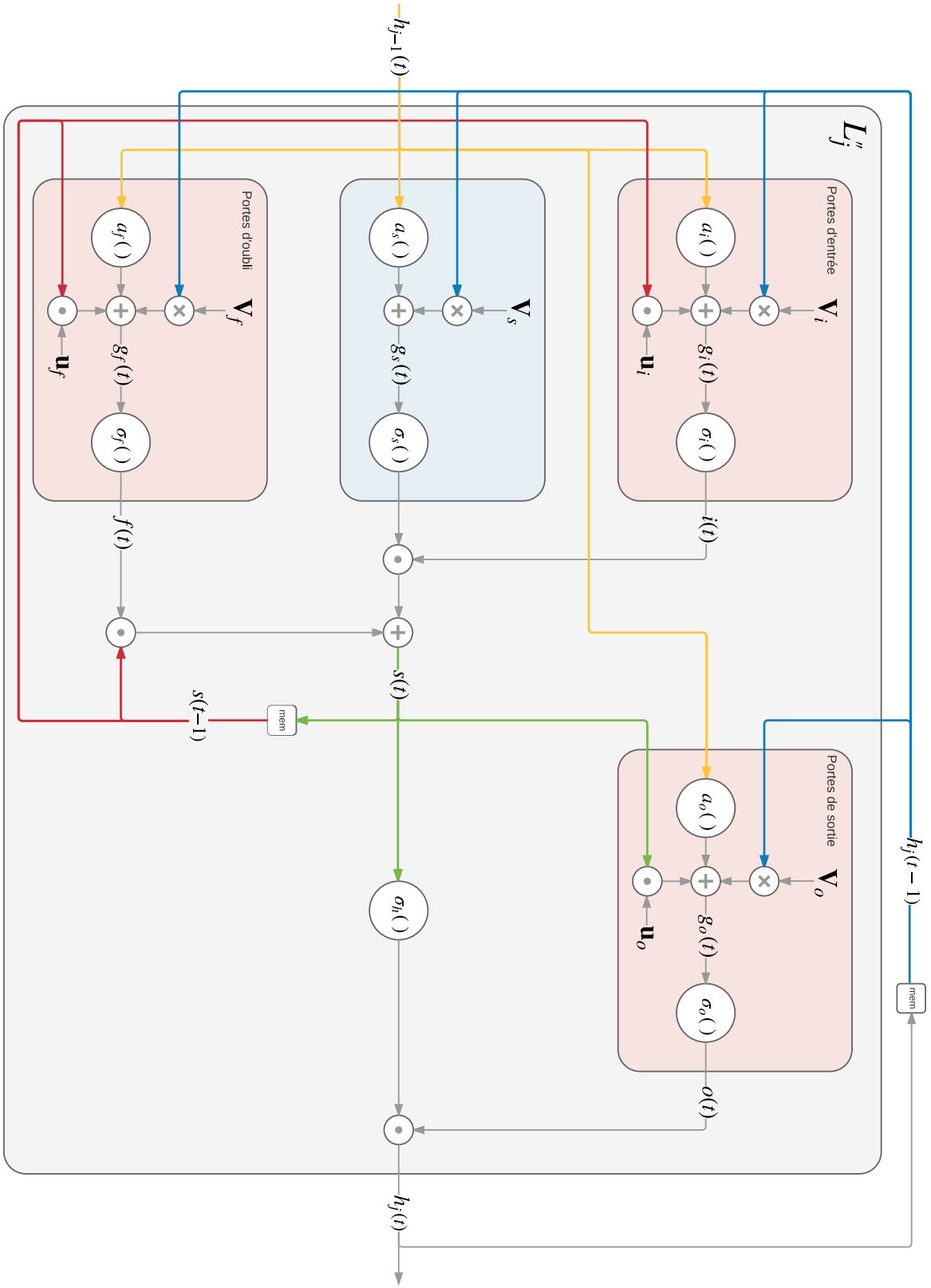


FIGURE 2.3 – Visualisation de la propagation de l'information lors de la passe avant dans une couche d'un LSTM.

Rétro-propagation des erreurs

Comme dans le cas du RNN standard, nous utilisons la technique de BPTT pour calculer les dérivées partielles de la fonction de coût $\mathcal{C}(\boldsymbol{z}, \boldsymbol{c})$ que l'on souhaite minimiser par rapport aux différents paramètres des couches LSTM. Pour ce faire, on parcourt la séquence en remontant le temps : $t : t_f \rightarrow 1$ et pour chaque pas de temps on détermine les gradients par rapport aux paramètres. Ensuite, les gradients globaux sont obtenus en sommant les contributions de chacun des pas de temps. La figure Fig. 2.4 décrit la rétro-propagation du gradient dans la couche L''_j au pas de temps t sous forme de graphe computationnel.

Pour calculer les dérivées partielles de $\mathcal{C}$ au pas de temps t par rapport aux paramètres de la couche L''_j, on commence par rétro-propager le gradient dans l'ensemble de la couche L''_j en commençant au niveau des portes de sortie. En dérivant l'équation Eq. 2.9, on obtient :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{o}}(t) = (\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j}(t) + \boldsymbol{\epsilon}(t+1)) \odot \sigma_h(\boldsymbol{s}(t)) \tag{2.10}$$

où $\boldsymbol{\epsilon}(t+1)$ est défini par l'équation Eq. 2.31 et correspond à la contribution du lien récurrent de la couche L''_j sur elle-même avec $\boldsymbol{\epsilon}(t_f + 1) = \mathbf{0}$ puisqu'aucune contribution ne peut parvenir au gradient d'au-delà de la fin de la séquence.

Ensuite, en rétro-propageant le gradient au travers de l'équation Eq. 2.8, on obtient :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) = \boldsymbol{J}_{\sigma_o}(\boldsymbol{g}_o(t)) \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{o}}(t) \tag{2.11}$$

où, comme dans la partie 1.1, $\boldsymbol{J}_{\sigma_o}(\boldsymbol{g}_o(t))$ correspond à la matrice jacobienne de la fonction d'activation σ_o au point $\boldsymbol{g}_o(t)$.

Nous introduisons alors les notations suivantes pour décrire les 3 contributions des portes de sortie aux différents gradients du reste de la couche L''_j via l'équation Eq. 2.7 :

$$\boldsymbol{\delta}_o(t) = \boldsymbol{W}_o^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) \tag{2.12}$$

$$\boldsymbol{\epsilon}_o(t) = \boldsymbol{V}_o^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) \tag{2.13}$$

$$\boldsymbol{\gamma}_o(t) = \boldsymbol{u}_o \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) \tag{2.14}$$

Nous pouvons alors calculer :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) = \boldsymbol{J}_{\sigma_h}(\boldsymbol{s}(t)) \cdot ((\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j}(t) + \boldsymbol{\epsilon}(t+1)) \odot \boldsymbol{o}(t)) + \boldsymbol{\gamma}(t) \tag{2.15}$$

où

$$\boldsymbol{\gamma}(t) = \boldsymbol{\gamma}_o(t) + \boldsymbol{\gamma}_s(t+1) + \boldsymbol{\gamma}_f(t+1) + \boldsymbol{\gamma}_i(t+1) \tag{2.16}$$

avec $\boldsymbol{\gamma}(t_f) = \boldsymbol{\gamma}_o(t)$ puisqu'aucune contribution ne peut parvenir au gradient d'au-delà de la fin de la séquence.

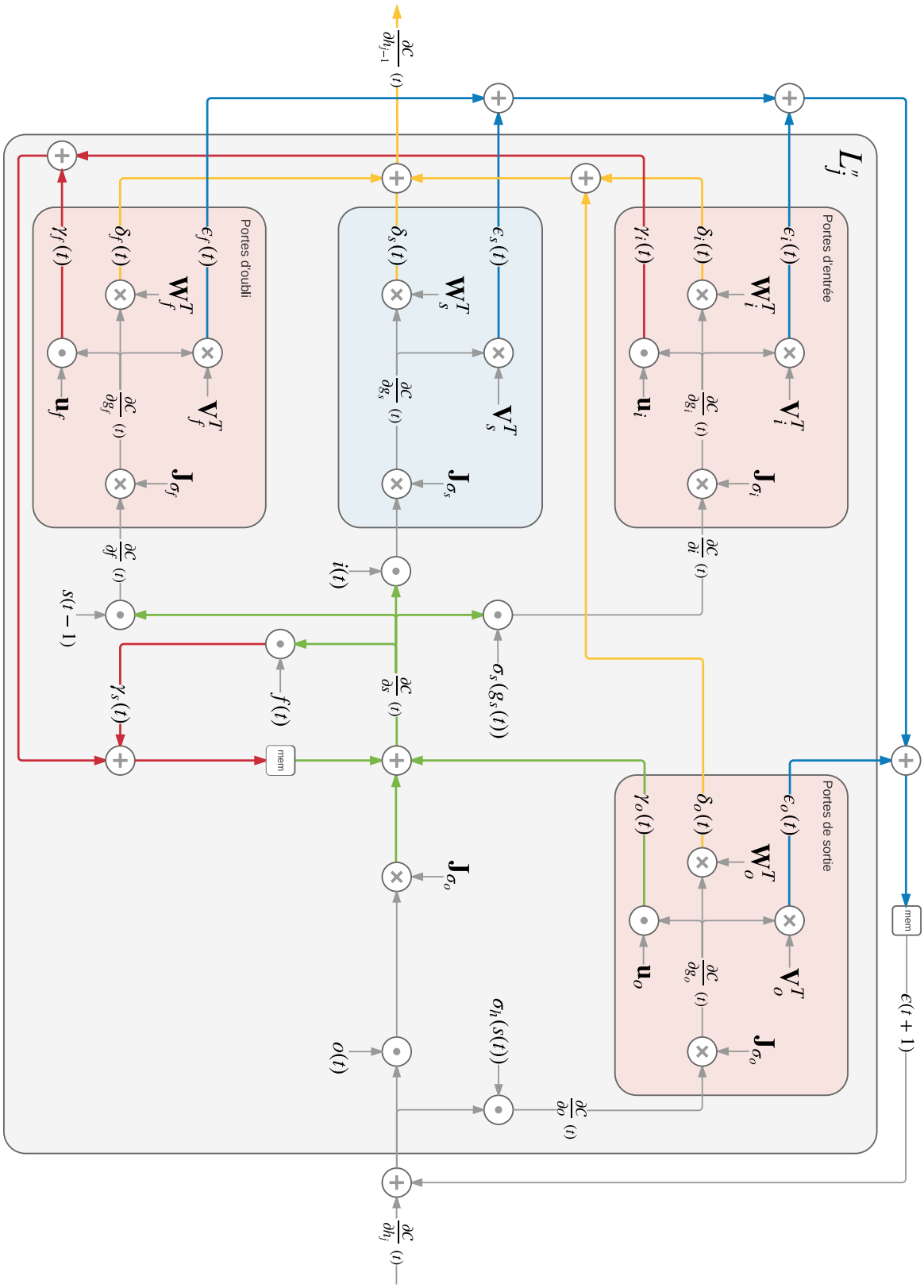


FIGURE 2.4 – Visualisation de la propagation de l'information lors de la passe arrière dans une couche d'un LSTM.

Ensuite, en rétro-propageant le gradient à travers l'équation Eq. 2.6 à l'aide des règles de dérivation des fonctions composées on obtient :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_s}(t) = \boldsymbol{J}_{\sigma_s}(\boldsymbol{g}_s(t)) \cdot (\frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) \odot \boldsymbol{i}(t)) \tag{2.17}$$

et

$$\boldsymbol{\gamma}_s(t) = (\frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) \odot \boldsymbol{f}(t)) \tag{2.18}$$

ainsi que les gradients en sortie des portes d'oubli et des portes d'entrée :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{f}}(t) = \frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) \odot \boldsymbol{s}(t-1) \tag{2.19}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{i}}(t) = \frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) \odot \sigma_s(\boldsymbol{g}_s(t)) \tag{2.20}$$

De façon similaire à ce que nous avons fait pour les portes de sortie, nous pouvons rétro-propager le gradient au travers des équations Eq. 2.4 et Eq. 2.2 :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_f}(t) = \boldsymbol{J}_{\sigma_f}(\boldsymbol{g}_f(t)) \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{f}}(t) \tag{2.21}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_i}(t) = \boldsymbol{J}_{\sigma_i}(\boldsymbol{g}_i(t)) \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{i}}(t) \tag{2.22}$$

Et nous introduisons les notations suivantes pour décrire les contributions des portes d'oubli aux différents gradients du reste de la couche L''_j via l'équation Eq. 2.3 :

$$\boldsymbol{\delta}_f(t) = \boldsymbol{W}_f^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_f}(t) \tag{2.23}$$

$$\boldsymbol{\epsilon}_f(t) = \boldsymbol{V}_f^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_f}(t) \tag{2.24}$$

$$\boldsymbol{\gamma}_f(t) = \boldsymbol{u}_f \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_f}(t) \tag{2.25}$$

et les contributions des portes d'entrée aux différents gradients du reste de la couche L''_j via l'équation Eq. 2.1 :

$$\boldsymbol{\delta}_i(t) = \boldsymbol{W}_i^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_i}(t) \tag{2.26}$$

$$\boldsymbol{\epsilon}_i(t) = \boldsymbol{V}_i^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_i}(t) \tag{2.27}$$

$$\boldsymbol{\gamma}_i(t) = \boldsymbol{u}_i \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_i}(t) \tag{2.28}$$

Pour finir, on rétro-propage le gradient au travers de l'équation Eq. 2.5 :

$$\boldsymbol{\delta}_s(t) = \boldsymbol{W}_s^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_s}(t) \tag{2.29}$$

$$\boldsymbol{\epsilon}_s(t) = \boldsymbol{V}_s^T \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_s}(t) \tag{2.30}$$

On peut maintenant calculer la contribution dans les équations Eq. 2.10 et Eq. 2.15 du lien récurrent de la couche L_j'' sur elle-même pour le pas de temps suivant dans le processus de rétro-propagation (c'est-à-dire $t-1$) :

$$\boldsymbol{\epsilon}(t) = \boldsymbol{\epsilon}_o(t) + \boldsymbol{\epsilon}_s(t) + \boldsymbol{\epsilon}_f(t) + \boldsymbol{\epsilon}_i(t) \tag{2.31}$$

ainsi que le gradient à rétro-propager dans la couche L_{j-1}'' :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_{j-1}}(t) = \boldsymbol{\delta}_i(t) + \boldsymbol{\delta}_f(t) + \boldsymbol{\delta}_s(t) + \boldsymbol{\delta}_o(t) \tag{2.32}$$

Une fois la séquence entièrement parcourue à rebours on peut enfin déterminer les dérivées partielles du coût $\mathcal{C}$ par rapport aux paramètres de la couche L_j''. Ainsi pour $x \in \{i, f, s, o\}$, on a :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{W}_x} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \cdot \boldsymbol{h}_{j-1}(t)^T \tag{2.33}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{V}_x} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \cdot \boldsymbol{h}_j(t-1)^T \tag{2.34}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{b}_x} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \tag{2.35}$$

avec en plus pour les portes d'entrée, d'oubli et de sortie :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{u}_i} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_i}(t) \odot \boldsymbol{s}(t-1)^T \tag{2.36}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{u}_f} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_f}(t) \odot \boldsymbol{s}(t-1)^T \tag{2.37}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{u}_o} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) \odot \boldsymbol{s}(t)^T \tag{2.38}$$

2.1.2 Coordination des portes

Nous introduisons ici une augmentation du modèle LSTM dans laquelle des liens directs entre les 3 types de portes (entrée, oubli et sortie) sont ajoutés de manière à ce que les portes puissent mieux coordonner leurs comportements. On appelle ce nouveau modèle : Long Short-Term Memory with Coordinated Gates (CG-LSTM).

Description du modèle

Pour le modèle CG-LSTM, la plupart des équations du modèle LSTM standard avec "peepholes" sont identiques. Seules les équations Eq. 2.2, Eq. 2.4 et Eq. 2.8 sont modifiées respectivement en Eq. 2.40, Eq. 2.42 et Eq. 2.44 :

$$\tilde{\boldsymbol{g}}_i(t) = \boldsymbol{v}_i \odot \boldsymbol{i}(t-1) + \boldsymbol{w}_i \odot \boldsymbol{f}(t-1) + \boldsymbol{y}_i \odot \boldsymbol{o}(t-1) \tag{2.39}$$

$$\boldsymbol{i}(t) = \sigma_i \left(\boldsymbol{g}_i(t) + \tilde{\boldsymbol{g}}_i(t)\right) \tag{2.40}$$

$$\tilde{\boldsymbol{g}}_f(t) = \boldsymbol{v}_f \odot \boldsymbol{i}(t-1) + \boldsymbol{w}_f \odot \boldsymbol{f}(t-1) + \boldsymbol{y}_f \odot \boldsymbol{o}(t-1) \tag{2.41}$$

$$\boldsymbol{f}(t) = \sigma_f \left(\boldsymbol{g}_f(t) + \tilde{\boldsymbol{g}}_f(t)\right) \tag{2.42}$$

$$\tilde{\boldsymbol{g}}_o(t) = \boldsymbol{v}_o \odot \boldsymbol{i}(t) + \boldsymbol{w}_o \odot \boldsymbol{f}(t) + \boldsymbol{y}_o \odot \boldsymbol{o}(t-1) \tag{2.43}$$

$$\boldsymbol{o}(t) = \sigma_o \left(\boldsymbol{g}_o(t) + \tilde{\boldsymbol{g}}_o(t)\right) \tag{2.44}$$

où nous introduisons 9 nouveaux vecteurs de $\boldsymbol{v}_{\{i,f,o\}}$, $\boldsymbol{w}_{\{i,f,o\}}$ et $\boldsymbol{y}_{\{i,f,o\}}$ qui permettent à chacune des portes de connaître l'état le plus récent des 2 autres portes qui lui correspondent ainsi que son propre état au pas de temps précédent. La figure Fig. 2.5a permet de visualiser les modifications sur les portes d'entrées (les 2 autres types de portes sont modifiées de façon similaire).

Rétro-propagation des erreurs

Là encore il y a peu de différences avec le modèle LSTM standard, nous devons juste prendre en compte les contributions aux gradients des nouveaux liens entre les portes d'une couche LSTM L''_j. Ainsi, l'équation Eq. 2.10 est modifiée comme suit :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{o}}(t) = \left(\frac{\partial \mathcal{C}}{\partial \boldsymbol{h}_j}(t) + \boldsymbol{\epsilon}(t+1)\right) \odot \sigma_h(\boldsymbol{s}(t)) + \boldsymbol{\theta}_i(t+1) + \boldsymbol{\theta}_f(t+1) + \boldsymbol{\theta}_o(t+1) \tag{2.45}$$

avec pour $x \in \{i, f, o\}$

$$\boldsymbol{\theta}_x(t) = \boldsymbol{y}_x \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \tag{2.46}$$

et $\boldsymbol{\theta}_x(t_f + 1) = \mathbf{0}$.

De la même façon, les équations Eq. 2.19 et Eq. 2.20 sont modifiées ainsi :

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{f}}(t) = \frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) \odot \boldsymbol{s}(t-1) + \boldsymbol{\eta}_i(t+1) + \boldsymbol{\eta}_f(t+1) + \boldsymbol{\eta}_o(t) \tag{2.47}$$

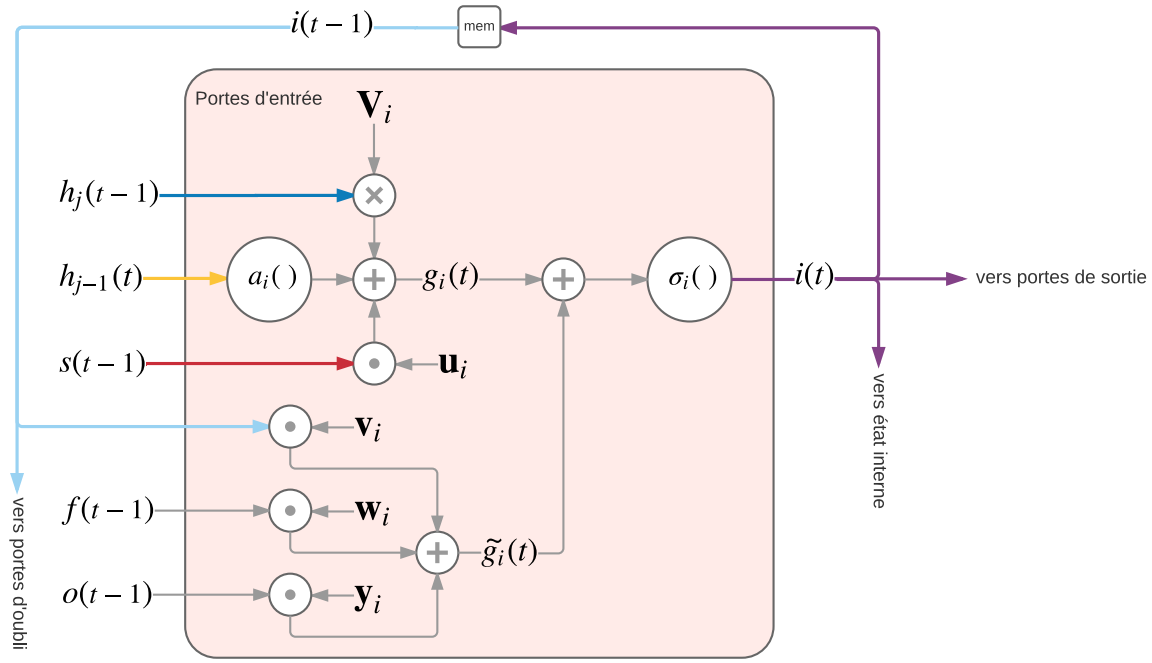
$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{i}}(t) = \frac{\partial \mathcal{C}}{\partial \boldsymbol{s}}(t) \odot \sigma_s(\boldsymbol{g}_s(t)) + \boldsymbol{\zeta}_i(t+1) + \boldsymbol{\zeta}_f(t+1) + \boldsymbol{\zeta}_o(t) \tag{2.48}$$

où pour $x \in \{i, f, o\}$

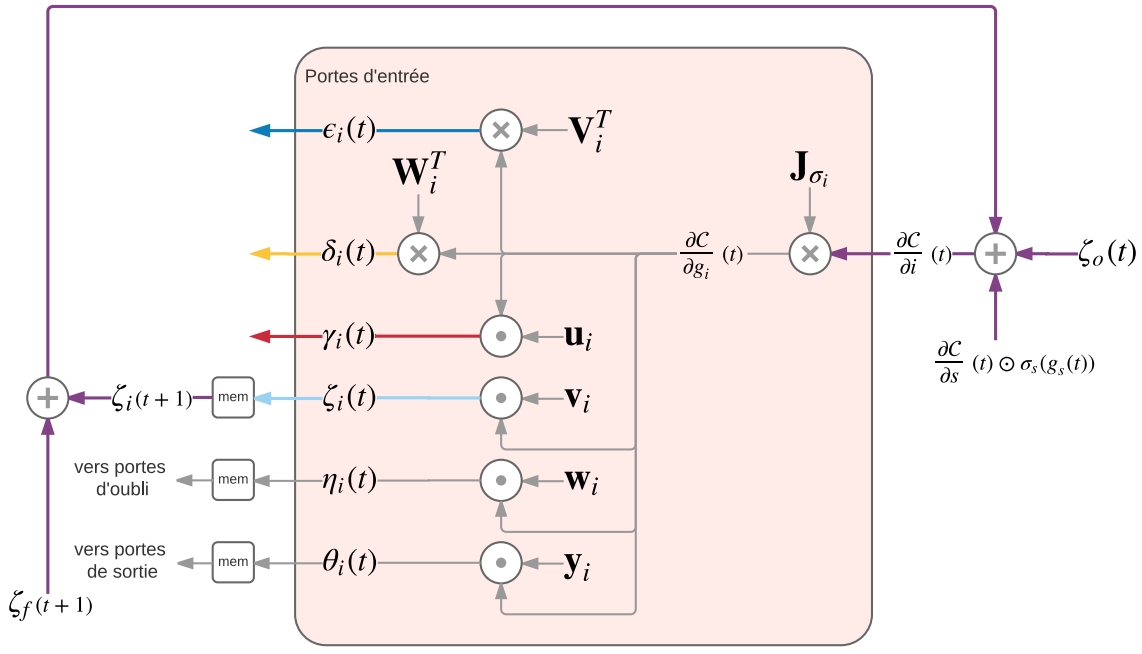
$$\boldsymbol{\eta}_x(t) = \boldsymbol{w}_x \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \tag{2.49}$$

$$\boldsymbol{\zeta}_x(t) = \boldsymbol{v}_x \odot \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \tag{2.50}$$

et pour $x \in \{i, f\}$, $\boldsymbol{\zeta}_x(t_f + 1) = \boldsymbol{\eta}_x(t_f + 1) = \mathbf{0}$.



(a) Graphe computationnel de la passe avant dans les portes d'entrée d'un CG-LSTM



(b) Graphe computationnel de la passe arrière dans les portes d'entrée d'un CG-LSTM

FIGURE 2.5 – Visualisation de la propagation de l'information lors de la passe avant dans les portes d'entrée d'un CG-LSTM et de la rétro-propagation du gradient de la fonction de coût $\mathcal{C}$ lors de la passe arrière dans ces mêmes portes. Les 2 autres types de portes (oubli et sortie) sont modifiées de façon similaire.

La figure Fig. 2.5b permet de visualiser ces modifications sur la rétro-propagation du gradient au travers des portes d'entrée (les 2 autres types de portes sont modifiés de façon similaire).

On voit au passage qu'il est nécessaire de conserver les valeurs de $\boldsymbol{\zeta}_{\{i,f\}}$, $\boldsymbol{\eta}_{\{i,f\}}$ et $\boldsymbol{\theta}_{\{i,f,o\}}$ d'un pas de temps sur l'autre pour pouvoir effectuer les calculs.

De la même façon que pour le LSTM standard, une fois la séquence entièrement parcourue à rebours on peut alors déterminer les dérivées partielles du coût $\mathcal{C}$ par rapport aux paramètres de la couche L''_j. Plus particulièrement, pour les 9 nouveaux vecteurs de paramètres, on a pour $x \in \{i, f\}$:

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{v}_x} = \sum_{t=2}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \odot \boldsymbol{i}(t-1) \tag{2.51}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}_x} = \sum_{t=2}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \odot \boldsymbol{f}(t-1) \tag{2.52}$$

et

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{v}_o} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) \odot \boldsymbol{i}(t) \tag{2.53}$$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}_o} = \sum_{t=1}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_o}(t) \odot \boldsymbol{f}(t) \tag{2.54}$$

mais également pour $x \in \{i, f, o\}$:

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{y}_x} = \sum_{t=2}^{t_f} \frac{\partial \mathcal{C}}{\partial \boldsymbol{g}_x}(t) \odot \boldsymbol{o}(t-1) \tag{2.55}$$

Le reste des équations décrites dans la section §2.1.1 est inchangé.

2.2 Performances des CG-LSTM

Nous présentons ici un aperçu du gain de performance apporté par le modèle CG-LSTM par rapport à d'autres modèles neuronaux sur les deux tâches qui nous intéressent dans cette thèse : la détection de la parole et l'identification de la langue. Des résultats plus détaillés pour chacune des tâches ainsi que des comparaisons avec d'autres types d'algorithmes plus classiques sont présentés dans les chapitres suivants.

2.2.1 Détection de la parole

Nous commençons par présenter le comparatif de quatre réseaux de neurones différents sur les différentes tâches de SAD décrites dans la section §1.2. Les quatre modèles neuronaux testés correspondent aux modèles détaillés dans les sections §1.1 et §2.1 : MLP, RNN d'Elman (dit *standard*), LSTM et CG-LSTM. De manière à obtenir une comparaison juste de la capacité de modélisation de chacun des ANN testés, tous les ANN utilisés sont dimensionnés pour avoir le même nombre de paramètres : 6000. Tous

les RNN ont une structure bidirectionnelle telle que décrite dans la section §1.1.4 avec une seule couche récurrente et avec pour réseau de sortie un MLP avec une unique couche cachée. Dans le cas du MLP utilisé seul, la structure utilisée est la même que pour le réseau de sortie des RNN bidirectionnels : une seule couche cachée dont la taille est augmentée de façon à atteindre le nombre total de paramètres fixé.

La description détaillée de l'intégration de ces différents ANN dans un système de SAD est donnée dans le Chapitre 3 avec le détail de la méthode d'apprentissage. Les algorithmes utilisés pour optimiser les paramètres des ANN sont quant à eux décrits dans la section §6.1.

Les différents ANN sont comparés sur cinq tâches de détection de la parole : minimisation du FER sur les données REPERE, AMI, Game of Thrones et le corpus vietnamien du programme Babel ; et minimisation du DCF sur les données de l'évaluation OpenSAD'15. Nous avons ainsi optimisé les quatre ANN pour ces tâches et les résultats obtenus sur les différents corpus sont détaillés dans le tableau Tab. 2.1. Il est important de préciser que pour ces tests le module de lissage des décisions (cf. §1.2.1) est désactivé de manière à obtenir un meilleur aperçu des capacités brutes des différents modèles neuronaux.

Tableau 2.1 – Performances de différents systèmes de SAD sur les données des corpus Babel, REPERE, AMI, Game of Thrones et OpenSAD'15. La performance est mesurée par le FER ou le DCF sans fonction de lissage (c'est-à-dire pas d'ajout de tampon, ni de suppression des segments ou des silences courts).

Type d'ANN	FER				DCF
	Babel	REPERE	AMI	Game of Thrones	OpenSAD'15
MLP	9.2	17.1	11.4	17.9	5.3
RNN standard	6.4	16.4	6.6	12.7	4.1
LSTM RNN	5.9	13.5	6.2	11.0	3.4
CG-LSTM RNN	5.8	12.9	5.9	11.0	3.0

Nous pouvons ainsi voir que les RNN se montrent particulièrement adaptés aux tâches de SAD puisque, en permettant d'exploiter le contexte temporel librement, ces modèles sont capables d'améliorer très notablement les décisions prises localement à chaque pas de temps. On observe ainsi une diminution des taux d'erreurs allant jusqu'à 42% relatif entre le MLP et le plus simple des modèles récurrents sur les données du corpus AMI.

Nous constatons aussi que le modèle récurrent le plus performant est le modèle CG-LSTM que nous avons introduit dans la section §2.1.2. En effet, ce modèle, quand on le compare au modèle LSTM standard, permet d'améliorer les taux d'erreurs de 5% relatif sur les corpus AMI et REPERE, de 10% relatif sur le corpus OpenSAD'15 et permet d'obtenir une performance équivalente sur les corpus Game of Thrones et Babel.

2.2.2 Identification de la langue

Nous avons ensuite comparé les différents types d'ANN sur une tâche simple d'identification de la langue : séparer une langue (l'arabe) de toutes les autres. Pour que la comparaison soit équitable nous utilisons les mêmes entrées pour tous les ANN testés et nous fixons le nombre de paramètres pour chacun des réseaux à 50k. [1]

Comme pour la segmentation parole/non-parole tous les RNN ont une structure bidirectionnelle telle que décrite dans la section §1.1.4 avec une seule couche récurrente et pour réseau de sortie un MLP avec une unique couche cachée. Dans le cas du MLP utilisé seul, la structure utilisée est la même que pour le réseau de sortie des RNN bidirectionnels : une seule couche cachée dont la taille est augmentée de façon à atteindre le nombre total de paramètres fixé.

Tableau 2.2 – Comparaison de différents ANN pour la tâche d'identification d'une langue (arabe) parmi les 14 langues de l'évaluation LRE-07. Pour chaque modèle neuronal nous présentons les taux d'erreurs de classification obtenus pour chacune des conditions de durée de parole du test officiel : 3 s, 10 s et 30 s.

Type d'ANN	LER sur 2 classes		
	3 s	10 s	30 s
MLP	33.7	25.0	20.0
RNN	33.9	21.4	12.6
LSTM	27.5	14.4	8.4
CG-LSTM	26.2	13.4	7.5

Le tableau Tab. 2.2 détaille les résultats obtenus sur le jeux de test de l'évaluation LRE-07. On peut ainsi voir que comme pour la tâche de SAD, les RNN basés sur des couches LSTM sont bien plus performants qu'un simple MLP ou qu'un RNN standard. On peut également noter que le modèle CG-LSTM apporte un gain de performance par rapport au modèle LSTM : 4.7% sur les segments de 3 s, 6.9% sur ceux de 10 s et 10.7% sur les segments longs de 30 s.

2.3 Conclusion

Nous avons introduit dans ce chapitre une modification du modèle LSTM standard que nous appelons modèle à mémoire court-terme persistante avec coordination des portes (CG-LSTM : Long Short-Term Memory with Coordinated Gates) et qui consiste à ajouter des liens entre les trois portes logiques de la cellule LSTM de base pour que les décisions prises par ces portes soient mieux coordonnées.

Comme l'attestent les résultats préliminaires présentés dans la section §2.2, cette modification permet, sans véritable surcoût sur la charge de calcul, d'améliorer la per-

1. La tâche d'identification de la langue étant plus complexe que la détection de la parole nous avons du augmenter le nombre de paramètres de nos modèles neuronaux pour obtenir un niveau de performance satisfaisant.

formance de ce type de RNN sur les deux tâches de traitement automatique de la parole que nous étudions dans cette thèse : la détection de parole et l'identification de la langue parlée.

Pour la tâche de détection de parole, le modèle CG-LSTM permet de gagner 3% relatifs en moyenne sur le FER obtenu sur quatre corpus très différents : Babel (2%), REPERE (5%), AMI (5%) et Game of Thrones (0%). De même dans le cadre de l'évaluation OpenSAD'15, le modèle neuronal que nous proposons amène une amélioration du DCF de 12% relatifs.

En identification de la langue, nous observons la même hiérarchie et nous obtenons avec le modèle CG-LSTM des gains relatifs sur le taux de d'erreurs de 44%, 34% et 7% quand on le compare respectivement à un MLP, un RNN standard ou un LSTM sur l'ensemble des trois durées de test de l'évaluation LRE-07 (3 s, 10 s et 30 s).

Au vu de tous ces résultats nous avons décidé de privilégier l'utilisation du modèle CG-LSTM dans la suite de nos travaux.

Chapitre 3

Apprentissage d'un RNN pour la détection de parole

Nous présentons dans ce chapitre les différents développements et expérimentations que nous avons réalisés de manière à optimiser le comportement d'un algorithme de détection de la parole basé sur un RNN. Notre but initial était de concevoir un algorithme de détection de la parole (ou Speech Activity Detection soit SAD) et de développer une méthode d'optimisation assez versatile pour pouvoir s'attaquer à différentes tâches telles que simplement minimiser le FER ou bien minimiser le WER d'un système de reconnaissance de parole. De par leurs capacités de reconnaissance de motifs et de prise en compte du contexte temporel les RNN semblaient tout particulièrement adaptés à la détection de parole.

Nous détaillons d'abord le processus d'optimisation (algorithmes et fonctions de coût) que nous avons mis au point pour répondre au problème de détection de la parole. Puis, nous présentons comment nous utilisons un RNN pour les tâches de SAD. Finalement, nous présentons les résultats pour les différentes tâches présentées dans la partie §1.2.

3.1 Processus d'optimisation

Nous détaillons dans cette section le processus d'optimisation que nous avons développé pour entraîner les différents systèmes de SAD soit pour minimiser le FER, soit pour minimiser le WER d'un système de RAP en aval. Pour ces deux tâches, déterminer la paramétrisation optimale d'un système de SAD ne peut être réalisé qu'à l'aide d'un processus itératif où à chaque itération on évalue une ou plusieurs paramétrisations candidates, puis en fonction de leur adéquation à la tâche visée un algorithme génère de nouvelles paramétrisations candidates.

Pour générer de nouvelles paramétrisations candidates à partir des précédentes et de leurs coûts respectifs on utilise des algorithmes heuristiques tels que Quantum-Behaved Particle Swarm Optimization (QPSO) ou les Algorithmes Génétiques (AG), ou encore un algorithme de descente de gradient dans les cas où il est possible de calculer le gradient de la fonction de coût par rapport à chacun des paramètres du système de SAD. Nous décrivons ceux que nous avons utilisés dans la section §3.1.1.

L'évaluation des paramétrisations candidates est quant à elle réalisée à l'aide d'une

fonction de coût qui permet de quantifier l'inadéquation de chacune des paramétrisations testées. Les différentes fonctions de coût que nous utilisons et que nous avons mises au point sont également explicitées dans ce chapitre (cf. §3.1.2).

La figure Fig. 3.1 permet de visualiser le processus itératif d'optimisation dans le cas d'une minimisation du WER d'un système de RAP. Dans le cas de la minimisation du WER d'un système de RAP, la meilleure fonction de coût est le WER lui-même en décodant les segments de parole à l'aide de la RAP. Malheureusement, le décodage est beaucoup trop long pour le faire à chaque itération. On ne réalise cette vérification qu'à la fin du processus d'optimisation qui lui se base sur une fonction de coût beaucoup moins coûteuse en temps de traitement.

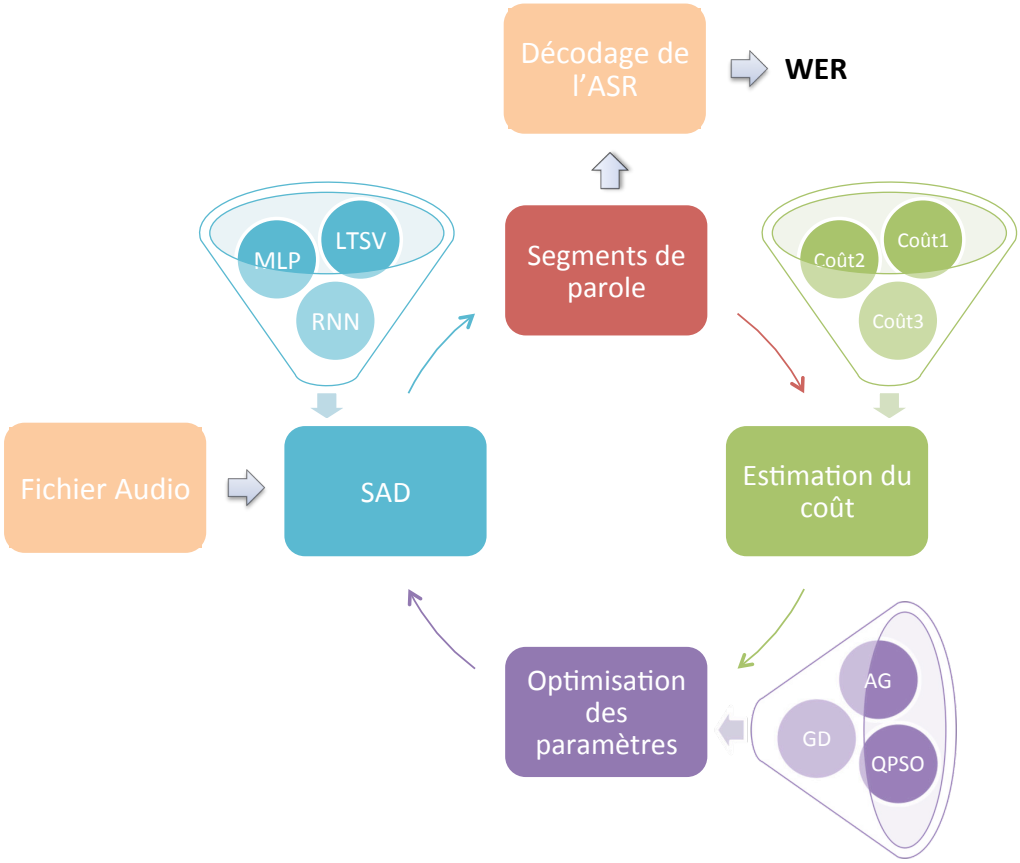


FIGURE 3.1 – Visualisation du processus d'optimisation d'un système de SAD afin de minimiser le WER d'un système de RAP. Ce processus itératif est basé sur l'enchaînement des trois étapes suivantes : 1) à partir de paramétrisations candidates pour l'algorithme de SAD, on estime les positions des segments de parole dans tous les fichiers audio d'apprentissage ; 2) à l'aide d'une fonction de coût adaptée on estime ensuite la pertinence des segments de parole pour la tâche à traiter ; 3) à partir du coût estimé dans l'étape précédente on utilise un algorithme d'optimisation à proprement parler pour déterminer de nouvelles paramétrisations qui devraient permettre de diminuer le coût. Pour cette étape on peut utiliser des algorithmes tels que QPSO ou les AG, ou encore un algorithme de descente de gradient dans les cas où il est possible de calculer le gradient de la fonction de coût par rapport à chacun des paramètres du système.

3.1.1 Algorithmes d'optimisation

Le processus d'optimisation que nous avons mis au point pour déterminer la paramétrisation optimale d'un système de SAD est principalement basé sur la technique d'optimisation heuristique QPSO que nous détaillons dans la section 6.1.2. L'avantage des méthodes heuristiques est que, contrairement aux méthodes utilisant la descente de gradient, il n'est pas nécessaire que le système à optimiser soit différentiable puisque les algorithmes d'optimisation heuristiques tels que QPSO ne se basent que sur l'évaluation de la fonction de coût pour chacune des paramétrisations candidates et non sur le gradient de cette fonction de coût. Avec les techniques heuristiques on peut donc optimiser les paramètres de modèles non-différentiables ayant éventuellement des paramètres à valeurs discrètes comme cela est le cas pour les algorithmes de SAD CrossCor et LTSV ou l'algorithme de décision et de lissage introduits dans la section §1.2.1. On peut également utiliser les méthodes d'optimisation heuristiques pour minimiser une métrique non différentiable voire non continue. C'est le cas par exemple de la métrique WER qui est la plus communément utilisée en RAP.

Pour toutes les expérimentations que nous décrivons dans ce chapitre, nous utilisons QPSO pour optimiser l'ensemble des paramètres des différents systèmes de SAD comme cela est montré sur la figure Fig. 3.2. Ainsi, pour un système de SAD utilisant l'algorithme CrossCor, QPSO optimise simultanément les paramètres de CrossCor et les paramètres de la fonction de décision et de lissage. Dans le cas d'un système de SAD utilisant l'algorithme LTSV, QPSO optimise simultanément les paramètres de l'analyse spectrale, les paramètres de LTSV et les paramètres de la fonction de décision et de lissage. Pour finir, dans le cas d'un système de SAD utilisant un RNN, QPSO optimise simultanément les paramètres d'extraction des coefficients cepstraux, les paramètres du RNN et les paramètres de la fonction de décision et de lissage.

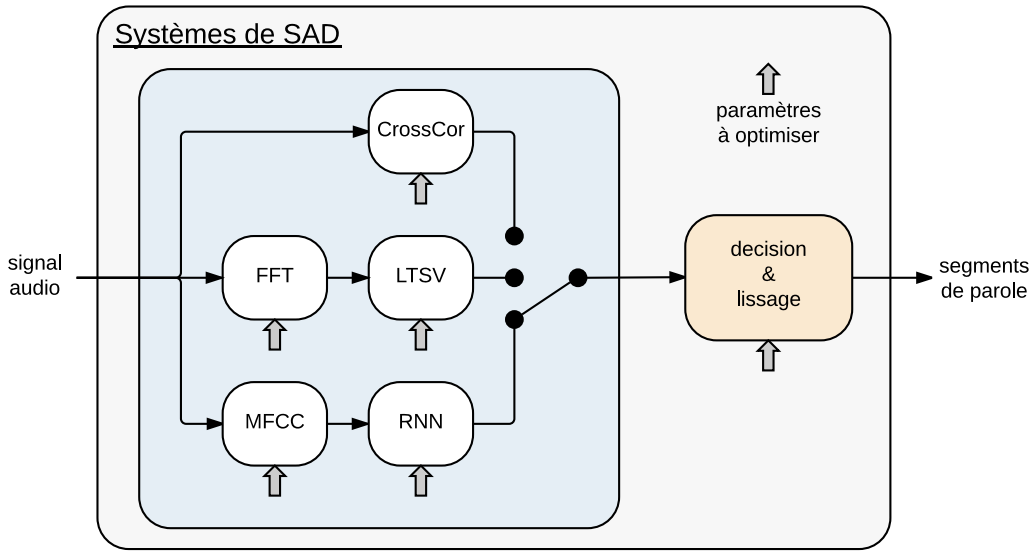


FIGURE 3.2 – Visualisation de l'ensemble des paramètres à optimiser dans les différents systèmes de SAD.

Toutefois, nous nous sommes aperçus que pour les systèmes de SAD utilisant des ANN il est intéressant d'utiliser un algorithme de descente de gradient en complément

de QPSO de manière à améliorer localement la meilleure solution trouvée par celui-ci. L'algorithme de descente de gradient n'est alors utilisé que pour raffiner les paramètres du réseau de neurones uniquement. L'algorithme de descente de gradient utilisé dans cette thèse est l'algorithme SMORMS3 décrit dans la section §6.1.1. Pour les systèmes neuronaux de SAD, le processus d'optimisation se décompose en trois optimisations successives :

1. Optimisation par QPSO de tous les paramètres du système : paramètres d'extraction des MFCC, poids de l'ANN, paramètres de l'algorithme de décision et de lissage.
2. Optimisation par SMORMS3 des poids du réseau de neurones.
3. Optimisation par QPSO des paramètres de l'algorithme de décision et de lissage afin de bien prendre en compte la nouvelle paramétrisation du réseau de neurone.

L'intérêt de cette stratégie alternée est quantifié dans la section §3.3.1.

3.1.2 Fonctions de coût

De façon générale, pour optimiser la paramétrisation d'un système pour une tâche donnée il est nécessaire de définir une mesure qui permette de rendre compte de l'inadéquation du système à cette tâche. On appelle cette mesure une fonction de coût. Ensuite, à l'aide d'un algorithme d'optimisation on s'attache à trouver la paramétrisation optimale du système c'est-à-dire celle qui minimise le coût.

Ici, nous souhaitons optimiser la paramétrisation d'un système de SAD pour deux tâches : la minimisation du FER et la minimisation du WER. Pour la première, la fonction de coût est simplement le FER lui-même puisqu'il est aisément mesurable. En revanche, mesurer le WER réel pour une paramétrisation du système de SAD donnée nécessite de décoder à l'aide du système de RAP chacun des segments de parole ce qui est très coûteux en temps de calcul. A titre indicatif, une optimisation de la paramétrisation d'un système de SAD simple basé sur l'algorithme LTSV nécessiterait près d'un an avec cette approche.

Nous avons donc exploré trois fonctions de coût plus simples à mesurer et qui permettent de minimiser indirectement le WER. Nous nous sommes d'abord intéressés à deux fonctions de coût liées au FER et relativement communes avant de développer une nouvelle fonction de coût qui prenne en compte les spécificités de la tâche que nous souhaitions traiter.

Lorsque nous utilisons ces fonctions de coût pour minimiser le WER d'un système de RAP, les seules hypothèses que nous faisons sont que nous avons à notre disposition :

— des jeux d'apprentissage, de validation et de test contenant des fichiers audio et les transcriptions orthographiques manuelles [1] ;

— le système de RAP dont nous souhaitons minimiser le WER.

1. Il est évidemment possible d'utiliser des annotations plus simples (début et fin des segments de parole uniquement) lorsque l'on souhaite minimiser le FER ou le DCF et non pas le WER.

FER pondéré ($\mathcal{C}_1$)

La première fonction de coût utilisée est une version pondérée du FER avec pour référence les annotations manuelles. Il est défini comme suit :

$$\mathcal{C}_1 = \alpha \sum_{t_p \in \mathcal{P}} (1 - z(t_p)) + (1 - \alpha) \sum_{t_b \in \mathcal{B}} z(t_b) \tag{3.1}$$

où $\mathcal{P}$ est l'ensemble des pas de temps appartenant à des segments de parole dans la référence manuelle, $\mathcal{B}$ est l'ensemble des pas de temps qui ne sont dans aucun segment de parole, α permet de régler l'importance relative entre les erreurs sur la parole et les erreurs sur le bruit, et $z(t)$ correspond à la sortie binaire du système de SAD pour le pas de temps t. On a alors $z(t) = 0$ lorsque t n'appartient pas à un segment de parole selon l'estimation du SAD et $z(t) = 1$ lorsqu'au contraire t appartient à un segment de parole selon l'estimation du SAD.

Pour les ANN et leur optimisation par descente de gradient, il est nécessaire d'utiliser une fonction de coût différentiable pour pouvoir déterminer le gradient de la fonction de coût par rapport aux paramètres de l'ANN. Pour obtenir un comportement similaire à $\mathcal{C}_1$, nous utilisons une version pondérée de la fonction de coût dite du maximum de vraisemblance appliquée à un classifieur binaire :

$$\mathcal{C}'_1 = -\alpha \sum_{t_p \in \mathcal{P}} \ln(1 - z(t_p)) - (1 - \alpha) \sum_{t_b \in \mathcal{B}} \ln(z(t_b)) \tag{3.2}$$

où $\forall t, z(t) \in [0, 1]$ est la sortie à valeurs réelles de l'ANN.

L'intérêt de ces deux métriques réside dans leur simplicité et leur facilité de mise en œuvre. En effet, pour les utiliser seules les annotations manuelles sont nécessaires. En revanche, lorsque l'on souhaite optimiser un système de SAD pour minimiser le taux d'erreur d'un système de RAP ces fonctions de coût ne prennent absolument pas en compte les spécificités du comportement de la RAP ce qui peut se révéler limitant. De plus, quand on l'utilise pour minimiser indirectement le WER il devient nécessaire de trouver la valeur optimale pour l'hyper-paramètre α ce qui rend l'optimisation un peu plus longue et/ou difficile.

FER pour la RAP ($\mathcal{C}_2$)

Nous nous plaçons ici dans le cas où nous souhaitons optimiser un système de SAD afin de minimiser le WER d'un système de RAP.

Lorsque nous avons à notre disposition ce système de RAP, il est intéressant de décoder la totalité du signal audio contenu dans les jeux d'apprentissage et de validation. En effet, la comparaison de sa sortie (mots, timecodes, et scores de confiance) avec la référence humaine peut elle-même servir de référence pour l'apprentissage du système de SAD.

Plus précisément, les segments de signal correspondant à des mots correctement reconnus ou à des substitutions sont considérés comme étant des segments de parole alors que les silences, les mots supprimés et insérés sont considérés comme étant du silence ou du bruit. Nous avons fait le choix de ne pas considérer les mots supprimés comme de la parole parce que nous n'avions aucun moyen de déterminer avec précision

les frontières des segments de signal correspondants. De plus, les mots omis peuvent être regroupés en deux catégories :

— ceux dont le segment de signal correspondant a été rejeté par la RAP comme étant du silence ou du bruit ;

— ceux qui font partie d'un groupe de n mots que la RAP a remplacés par un seul mot générant ainsi une substitution et $n-1$ omissions.

Dans le premier cas, il est inutile que le système de SAD classifie ces segments comme étant de la parole puisqu'ils seront de toute façon ignorés par la RAP et n'auront donc aucun impact sur le taux d'erreur. Dans le second cas, le segment de signal correspondant au mot reconnu par la RAP regroupe en général la plupart des segments correspondant aux n mots. On a donc une annotation correcte en terme de segmentation parole/non-parole.

Une fois cette nouvelle référence de segmentation parole/non-parole établie, nous utilisons la fonction de coût définie par l'équation Eq. 3.3. Cette équation est identique à l'équation Eq. 3.1, seuls les ensembles $\mathcal{P}$ et $\mathcal{B}$ sont différents. Nous appelons cette fonction de coût $\mathcal{C}_2$.

$$\mathcal{C}_2 = \alpha \sum_{t_p \in \mathcal{P}} (1 - z(t_p)) + (1 - \alpha) \sum_{t_b \in \mathcal{B}} z(t_b) \tag{3.3}$$

Nous avons pu noter que lorsque l'on passe de la fonction de coût $\mathcal{C}_1$ à la fonction de coût $\mathcal{C}_2$ sur une même tâche, il est utile d'augmenter la valeur de l'hyper-paramètre α. Cette différence s'explique essentiellement par la confiance que nous pouvons accorder aux deux types d'annotations (humaine et automatique). En effet, les annotateurs humains ont tendance à inclure du silence dans les segments étiquetés comme contenant de la parole (marge en début et en fin de phrase, pauses courtes entre les mots...). A l'inverse, lorsque nous choisissons d'étiqueter uniquement les mots corrects et les substitutions d'un système de RAP comme étant la seule parole contenue dans le fichier audio nous y incluons très peu de silence et de bruit. Il est alors préférable de limiter les erreurs sur les parties étiquetées comme contenant de la parole en utilisant une plus grande valeur de α qu'avec la fonction de coût $\mathcal{C}_1$.

Simili-WER ($\mathcal{C}_3$)

La fonction de coût $\mathcal{C}_2$ utilise les sorties du système de RAP mais elle ne prend pas en compte la principale spécificité de la métrique WER. En effet, dans cette métrique tous les mots qui génèrent une erreur ont le même impact sur le résultat final, et ce quelle que soit la durée du signal audio correspondant ou le type d'erreurs engendrées (insertion, suppression ou substitution). Pour prendre en compte ces traits particuliers de la métrique WER, nous avons mis au point une fonction de coût qui à partir de la connaissance des erreurs de la reconnaissance sur la totalité du signal audio et d'une hypothèse de SAD permet d'obtenir une estimation du WER réel.

Nous définissons C, S, D et I comme étant respectivement les ensembles des mots correctement reconnus, des mots substitués, des mots omis et des mots insérés par le système de RAP. Pour tout mot $w \in C \cup S \cup I$, nous appelons F_w l'ensemble des pas de temps correspondant au mot w dans la sortie de la RAP. Puis nous définissons les ensembles $\bar{S}$, $\bar{D}$ et $\bar{I}$ comme suit :

— un mot $w \in \bar{S}$ si et seulement si $w \in S$ et tous les pas de temps de l'ensemble F_w sont classifiés comme étant de la parole par le SAD.

— un mot $w \in \bar{D}$ si et seulement si soit $w \in D$, soit $w \in C \cup S$ et au moins un pas de temps de l'ensemble F_w est classifié comme étant de la non-parole par le SAD.

— un mot $w \in \bar{I}$ si et seulement si $w \in I$ et au moins un pas de temps de l'ensemble F_w est classifié comme étant de la parole par le SAD.

Nous introduisons également les deux ratios suivants :

— ρ_w est le ratio entre la durée du signal correspondant au mot $w \in C \cup S$ qui n'est pas classifié en tant que parole et la durée du signal correspondant au mot w dans la référence.

— τ_w est le ratio entre la durée du signal correspondant au mot $w \in \bar{I}$ qui est classifié en tant que parole et la durée du signal correspondant au mot w dans l'hypothèse de la RAP.

Pour finir, la fonction de coût $\mathcal{C}_3$ est définie ainsi

$$\mathcal{C}_3 = \frac{\tilde{S} + \tilde{D} + \tilde{I} + \rho + \tau}{N_{words}} \tag{3.4}$$

avec

$$\tau = \sum_{w \in \bar{I}} \tau_w \tag{3.5}$$

et

$$\rho = \sum_{w \in C \cup S} \rho_w \tag{3.6}$$

où $\tilde{S}$, $\tilde{D}$ et $\tilde{I}$ sont respectivement les nombres de mots dans les ensembles $\bar{S}$, $\bar{D}$ et $\bar{I}$.

Les deux termes τ et ρ ont été introduits pour lisser les discontinuités de la métrique WER et faciliter l'optimisation en supprimant les plateaux de la fonction de coût.

De façon similaire à ce que nous avons fait pour la fonction de coût $\mathcal{C}_1$, nous avons mis au point une version différentiable de $\mathcal{C}_3$ de manière à pouvoir optimiser les SAD "neuronaux" en utilisant une méthode de descente de gradient :

$$\mathcal{C}'_3 = -\sum_{w \in C \cup S} \left(\frac{1}{\delta_w} \sum_{t_w \in F_w} \ln(z(t_w)) \right) - \sum_{w \in I} \left(\frac{1}{\delta_w} \sum_{t_w \in F_w} \ln(1 - z(t_w)) \right) \tag{3.7}$$

où $z(t_w) \in [0; 1]$ est la sortie de l'ANN pour le pas de temps t_w appartenant au mot w et où δ_w le nombre de pas de temps correspondant à la totalité du mot w.

3.2 Algorithmes de Segmentation à base de RNN

Les réseaux de neurones ont déjà été proposés comme méthode pour calculer un critère de SAD [55–57] ou pour servir de mécanisme de fusion de plusieurs critères de SAD plus classiques [58, 59]. Dans ce chapitre nous les utilisons exclusivement pour calculer un critère de SAD.

Parmi les techniques basées sur des réseaux de neurones, les RNN et plus spécifiquement les LSTM présentent des qualités qui les rendent attrayants (cf. [60–63]).

Ainsi, contrairement au MLP, un RNN ne travaille pas sur un contexte temporel fixé par un hyper-paramètre à ajuster mais sur l'ensemble de la séquence temporelle.

Dans toutes nos expérimentations avec des RNN (RNN standards, LSTM ou CG-LSTM) nous avons utilisé une architecture bidirectionnelle telle que décrite dans la section 1.1.4 (p. 21) de manière à pouvoir exploiter le contexte passé et futur de chaque trame.

Quel que soit le type d'ANN utilisé, nous nous plaçons dans un contexte de classification binaire (voir section 1.1.1), la couche de sortie de l'ANN n'étant constituée que d'un unique "neurone" avec pour fonction d'activation la fonction logistique :

$$\sigma(x) = \frac{1}{1 + \mathrm{e}^{-x}}, \quad \forall x \in \mathbb{R} \tag{3.8}$$

Ainsi, la sortie du réseau est un réel compris entre 0 et 1 que l'on interprète comme la probabilité que la trame que l'on considère contienne de la parole.

D'autre part, les entrées fournies aux différents réseaux testés sont toujours sous la forme de séquences de trames où chaque trame est la concaténation des coefficients cepstraux (MFCC) avec leurs dérivées premières et secondes. Les paramètres d'extraction de ces MFCC sont optimisés conjointement avec les paramètres du réseau lors de l'optimisation par QPSO (voir §3.1.1). On peut ainsi optimiser la taille et le type de fenêtrage du signal utilisés pour le calcul du spectre, la valeur du coefficient de pré-emphase, l'utilisation ou non de convolutions temporelle ou fréquentielle, les fréquences minimale et maximale du banc de filtres, le nombre de filtres, l'ajout ou non d'un bruit blanc sur le signal...

3.3 Expérimentations

Pour valider notre démarche, nous avons confronté notre processus d'optimisation à des tâches variées autant en termes de langues et d'environnements acoustiques. Nous présentons comment optimiser un système de SAD de manière à minimiser le WER d'un système de RAP en aval. Et nous présentons les résultats obtenus sur des tâches de segmentation pure dans le cadre d'une évaluation internationale et sur des données de types très différents : réunions de travail, séries télévisées...

Le détail des conditions expérimentales (corpus et métriques) dans lesquelles nous nous sommes placés est donné dans la section §1.2.2.

3.3.1 Minimisation du FER

Nous commençons par comparer les performances des différentes techniques d'extraction de critères de SAD sur une tâche de minimisation du FER[2] sur le corpus vietnamien du programme Babel. Nous en profitons pour vérifier la qualité de notre méthode d'optimisation en comparant les résultats obtenus aux performances d'un système de SAD de référence.

Nous avons optimisé six systèmes de SAD différents en utilisant les fonctions de coût correspondant au FER c'est-à-dire ($\mathcal{C}_1$, $\mathcal{C}'_1$) avec un coefficient $\alpha = 0.5$ (cf. p.

2. Le FER est calculé sur l'ensemble des fichiers.

60). Les résultats obtenus avec et sans fonction de lissage (c'est-à-dire pas d'ajout de tampons, ni de suppression des segments ou des silences courts) sont détaillés dans le tableau Tab. 3.1 et sur la figure Fig. 3.3 où nous analysons la distribution des erreurs calculées par fichier de test pour chacun des systèmes.

Tableau 3.1 – Performances de différents systèmes de SAD sur le corpus de test vietnamien du programme Babel. La performance est mesurée par le FER avec et sans fonction de lissage (c'est-à-dire pas d'ajout de tampons, ni de suppression des segments ou des silences courts). Les modèles RNN sont moins dépendants de la fonction de lissage que les autres modèles et le modèle CG-LSTM permet d'obtenir la meilleure performance.

Système de SAD	FER	
	~~lissage~~	lissage
CrossCor	8.95	7.60
LTSV	10.87	7.85
MLP	9.24	6.29
RNN standard	6.42	6.24
LSTM RNN	5.88	5.83
CG-LSTM RNN	5.83	5.76
gmmSAD	13.50	

Nous pouvons ainsi voir que l'utilisation de notre processus d'optimisation sur des données ciblées permet d'obtenir une bien meilleure performance qu'avec un système généraliste tel que *gmmSAD*. On peut aussi noter que les systèmes à base d'ANN sont nettement plus performants que les systèmes basés sur des critères de SAD plus classiques (gain d'au moins 17%). Et parmi les techniques neuronales, les RNN se montrent particulièrement adaptés puisque, en permettant d'exploiter le contexte temporel librement, ces modèles sont capables d'apprendre également une grande partie de la fonction de lissage permettant de minimiser le FER ce qui n'est pas le cas du MLP. Pour finir, nous constatons que le réseau récurrent le plus performant est le modèle que nous avons introduit dans la section §2.1.2 et qui permet d'améliorer le FER moyen de 2% par rapport au modèle LSTM standard.

Durant cette première expérimentation nous avons également voulu quantifier l'impact du choix des MFCC comme entrées de nos modèles neuronaux. Nous avons donc comparé les différences de performances entre des systèmes de SAD basés sur des CG-LSTM ayant en entrée le signal temporel, le log-spectrogramme complet ou les MFCC avec un nombre de neurones identiques. Les résultats sont présentés dans le tableau Tab. 3.2 et c'est bien avec les MFCC que la performance est la meilleure même si les écarts de performances sont relativement faibles. Ces faibles écarts de performance démontrent la grande versatilité des CG-LSTM que l'on peut entraîner, avec succès, directement sur le signal brut malgré la très grande redondance d'information de ce signal et la nécessité d'exploiter un contexte temporel large puisque le signal audio dans le cadre du programme Babel est échantilloné à 8kHz soit 8000 points par seconde.

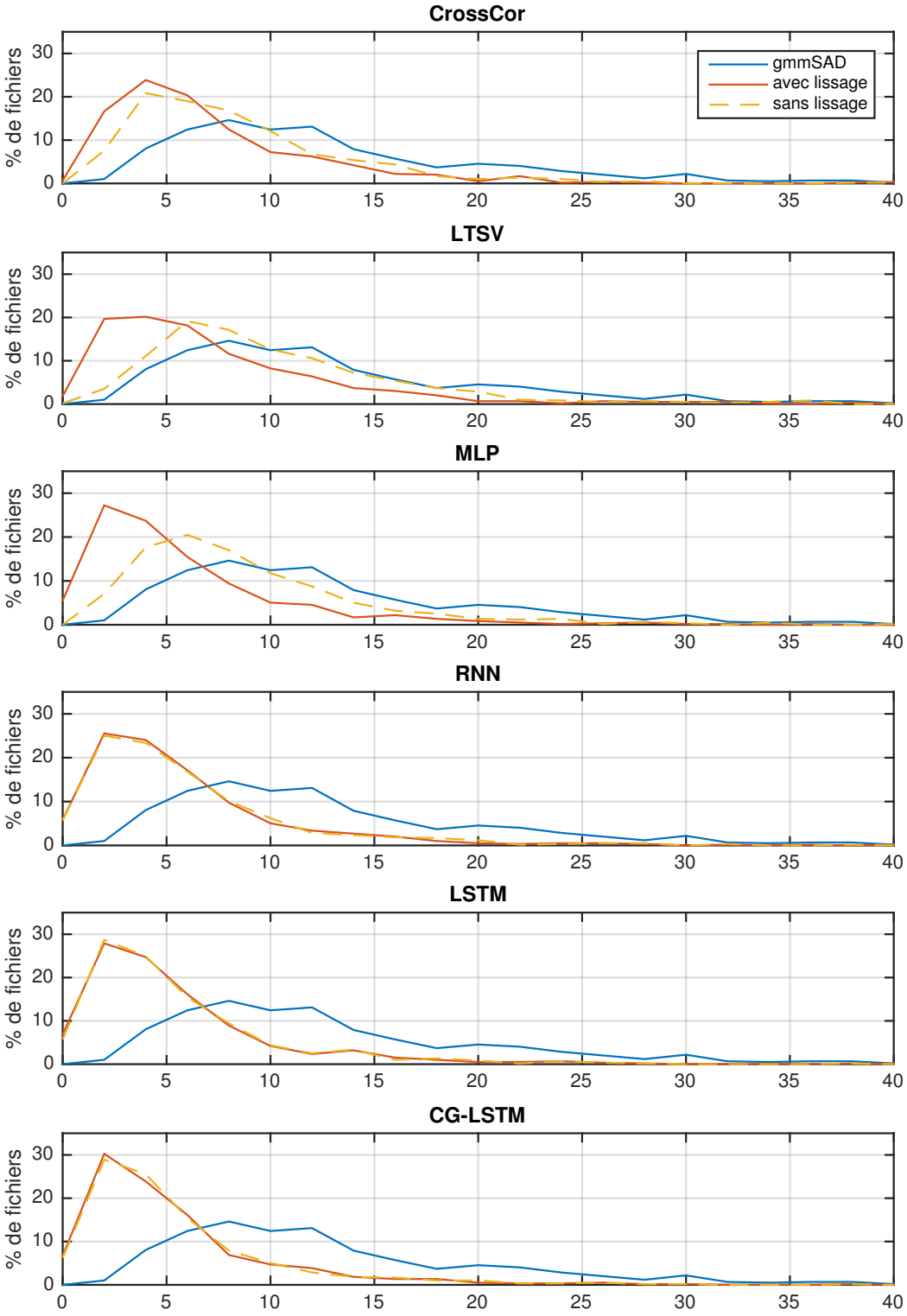


FIGURE 3.3 – Distributions du FER calculé par fichier pour les différents systèmes de SAD avec et sans lissage. Les FER affichés ont été obtenus sur le jeu de test vietnamien du programme Babel. Les RNN sont capables d'apprendre en grande partie la fonction de lissage nécessaire pour minimiser le FER. Le système de SAD produisant les taux d'erreurs les plus bas (distribution le plus à gauche sur le graphe) est obtenu en utilisant le modèle CG-LSTM.

Toutefois, il est important de noter la différence de temps de traitement en fonction du choix de représentation des entrées. Ainsi, compte tenu des performances aussi bien en temps de traitement [3] qu'en performance de SAD, les MFCC apparaissent comme étant le meilleur choix de représentation.

Tableau 3.2 – Performance de segmentation sur le jeu de test vietnamien lorsque l'on fait varier le type d'entrée. Pour ce test, le système de SAD utilise le modèle CG-LSTM pour estimer le critère de SAD. Le temps de traitement moyen est obtenu sur un processeur Intel Xeon CPU L5520 HT.

Type d'entrées	FER par fichier				Temps moyen de traitement
	Cas pire	Dernier décile	Moyenne	Médiane	
MFCC	62.58	11.54	5.76	4.02	$\frac{1}{957}$ RT
Spectrogramme	63.77	11.90	6.61	4.09	$\frac{1}{202}$ RT
Signal brut	60.29	11.95	6.35	4.14	$\frac{1}{58}$ RT

Nous avons également étudié l'intérêt de chacune des phases de notre processus d'optimisation pour valider le choix d'une stratégie d'utilisation alternée de QPSO et de SMORMS3 pour optimiser un système de SAD utilisant un réseau de neurones. Le tableau Tab. 3.3 présente les résultats obtenus avec différents processus d'optimisation utilisant soit QPSO ou SMORMS3 seuls, soit une combinaison des deux. On rappelle que SMORMS3 n'est utilisé que pour optimiser les poids du réseau de neurones et que QPSO n'est utilisé que pour optimiser les paramètres de la fonction de décision et de lissage lorsqu'il est employé après SMORMS3.

Tableau 3.3 – Performance de segmentation (FER) sur le jeu de validation vietnamien lorsque l'on modifie le processus d'optimisation. Pour ce test, le système de SAD utilise le modèle CG-LSTM pour estimer le critère de SAD.

Processus d'optimisation	FER par fichier			
	Cas pire	Dernier décile	Moyenne	Médiane
QPSO	60.24	17.68	10.04	8.02
SMORMS3	62.44	14.20	7.54	5.60
SMORMS3 $\rightarrow$ *QPSO*	61.82	12.91	6.65	4.87
QPSO $\rightarrow$ *SMORMS3*	62.38	11.72	5.83	4.07
QPSO $\rightarrow$ *SMORMS3* $\rightarrow$ *QPSO*	62.58	11.54	5.76	4.02

On voit ainsi que l'utilisation seule de l'un ou l'autre des algorithmes donne déjà de très bons résultats mais que l'utilisation alternée des deux méthodes permet d'améliorer notablement les performances. En particulier, utiliser QPSO pour optimiser les

3. Sur un processeur Intel Xeon CPU L5520 HT

paramètres de la chaîne d'extraction des MFCC permet ensuite de tirer le meilleur parti de l'optimisation par descente de gradient. De même, on peut encore gagner un peu en utilisant QPSO après la descente de gradient pour réajuster les paramètres des fonctions de décision et de lissage (on verra dans le paragraphe suivant que c'est encore plus vrai lorsque l'on cherche à minimiser le WER).

3.3.2 Minimisation du WER

Notre but premier étant de minimiser le WER d'un système de RAP, nous avons déterminé les valeurs du coefficient α permettant de minimiser le WER lorsque l'on utilise les fonctions de coût $\mathcal{C}_1$ et $\mathcal{C}_2$. Ainsi le tableau Tab. 3.4 présente le WER réel obtenu par le système de RAP lorsque l'on segmente les fichiers du jeu de développement avec un système de SAD (CG-LSTM) optimisé à l'aide de la fonction de coût $\mathcal{C}_1$ (respectivement $\mathcal{C}_2$) pour différentes valeurs du coefficient α. A titre de comparaison, nous présentons également dans ce tableau le WER obtenu lorsque l'on utilise la fonction de coût $\mathcal{C}_3$.

Tableau 3.4 – Impact du coefficient α sur le WER réel obtenu par le système de RAP lorsque l'on segmente les fichiers du jeu de développement avec un système de SAD optimisé à l'aide de la fonction de coût $\mathcal{C}_1$ ou $\mathcal{C}_2$. Le WER réel obtenu en utilisant la fonction de coût $\mathcal{C}_3$ qui n'a pas d'hyper-paramètre est présenté à titre de comparaison.

α	WER sur le jeu de développement		
	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
0.10	51.80	100.0	**51.13**
0.15	51.74	-	*idem*
0.20	**51.62**	88.85	*idem*
0.25	51.64	-	*idem*
0.30	51.83	75.79	*idem*
0.40	51.84	66.03	*idem*
0.50	51.89	59.80	*idem*
0.60	52.09	55.56	*idem*
0.70	52.15	53.65	*idem*
0.80	52.85	52.33	*idem*
0.90	53.09	51.97	*idem*
0.95	-	**51.30**	*idem*
0.99	-	52.51	*idem*

On peut voir que les deux fonctions de coût ne se comportent pas du tout de la même façon. Pour $\mathcal{C}_1$ la valeur du coefficient α qui permet d'obtenir le meilleur

taux d'erreur est 0.2 ce qui s'explique essentiellement par le fait que le système de RAP insère beaucoup de mots et qu'il vaut donc mieux diminuer les risques de fausses alarmes quitte à perdre un peu de signal de parole. Au contraire, dans le cas de la fonction de coût $\mathcal{C}_2$ les références de segmentation prennent en compte que le signal provoquant des insertions doit être considéré comme du bruit ; il est donc beaucoup plus important de minimiser le risque de perdre du signal de parole quitte à faire quelques fausses alarmes. C'est exactement ce que l'on observe puisque le coefficient α qui donne le meilleur WER est 0.95.

Enfin, on peut constater que la fonction de coût $\mathcal{C}_3$ permet d'obtenir un meilleur WER sur le jeu de développement que les deux autres fonctions de coût quelle que soit la valeur du coefficient α choisie. On voit donc l'intérêt d'utiliser une fonction de coût telle que $\mathcal{C}_3$ qui prend en compte le comportement du système de RAP dont nous voulons améliorer la performance et qui se rapproche le plus possible de la métrique visée. Nous avons d'ailleurs vérifié la bonne correspondance de la fonction de coût $\mathcal{C}_3$ avec le WER réel obtenu sur le jeu de développement : pour chaque système dont nous avons déterminé WER dans le tableau Tab. 3.4 nous avons calculé le simili-WER sur le même jeu de données à l'aide de la fonction de coût $\mathcal{C}_3$. En effectuant une régression linéaire on obtient un coefficient de détermination de 0.98 ce qui montre la bonne capacité de prédiction du WER par la fonction de coût de simili-WER $\mathcal{C}_3$ (voir aussi la figure Fig. 3.4).

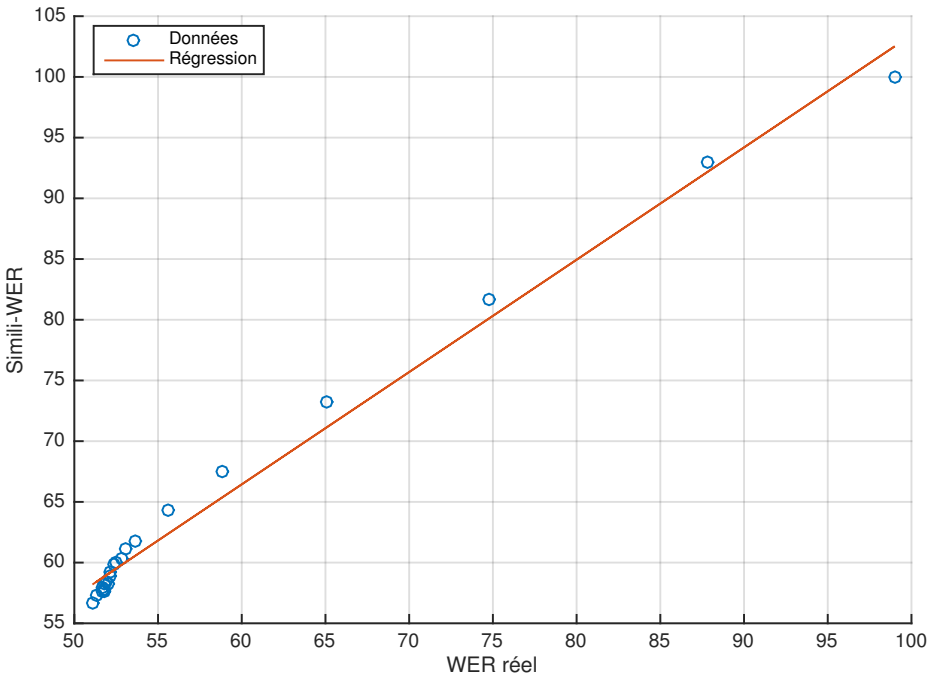


FIGURE 3.4 – Correspondance entre le simili-WER et le WER réel calculés sur le jeu de développement vietnamien du programme Babel. Le coefficient de détermination de la régression linéaire est de 0.98 ce qui montre la bonne capacité de prédiction du WER par la fonction de coût $\mathcal{C}_3$.

Une fois la valeur optimale du coefficient α choisie, nous avons optimisé les différents systèmes de SAD avec les trois fonctions de coût puis nous avons mesuré les

performances obtenues sur le jeu de test. Le tableau Tab. 3.5 permet de comparer le WER obtenu sur le jeu de test pour tous les systèmes de SAD. Ils sont également comparés au système de SAD de référence *gmmSAD* qui a d'ailleurs servi à extraire les segments de parole pour entraîner le système de RAP dont nous voulons optimiser le WER.

On peut voir que l'optimisation des systèmes de SAD sur des données représentatives avec des fonctions de coût qui se rapprochent le plus possible de la métrique d'intérêt (choix du coefficient α pour $\mathcal{C}_1$ et $\mathcal{C}_2$, et $\mathcal{C}_3$ par construction) permet de toujours diminuer le WER quand on le compare au WER obtenu avec une segmentation généraliste. On note également que plus la fonction de coût est similaire à la métrique cible plus le gain est important, et ce quel que soit le système de SAD considéré. La fonction de coût $\mathcal{C}_3$ qui n'a par ailleurs pas d'hyper-paramètre à choisir/régler s'impose comme la fonction de coût de choix pour optimiser un système de SAD de manière à minimiser le WER.

D'autre part, comme nous l'avons observé lors des premières expériences, le modèle CG-LSTM se révèle être le plus performant parmi tous les systèmes de SAD testés. En effet, quelle que soit la fonction de coût utilisée c'est avec ce modèle que l'on obtient le WER le plus faible. En couplant $\mathcal{C}_3$ avec le modèle CG-LSTM, on obtient finalement un gain relatif de 4.5% par rapport au WER obtenu avec le système de SAD *gmmSAD* qui a été utilisé durant l'entraînement du système de RAP.

Tableau 3.5 – Impact de la fonction de coût sur le WER obtenu avec les différents systèmes de SAD sur le jeu de test vietnamien du programme Babel. La fonction de coût $\mathcal{C}_3$ permet d'obtenir la meilleure performance quel que soit le système de SAD utilisé. Et le modèle CG-LSTM permet d'obtenir la meilleure performance quelle que soit la fonction de coût utilisée. En couplant les deux on obtient un gain relatif de 4.5% sur le WER par rapport au système généraliste ayant servi à entraîner le système d'RAP.

Système de SAD	WER sur le jeu de test			
	FER	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$
CrossCor	56.7	56.9	56.4	56.1
LTSV	57.1	56.2	56.0	55.6
MLP	56.6	56.2	55.7	55.4
RNN standard	56.5	56.1	55.8	55.2
LSTM RNN	56.5	55.8	55.4	54.8
CG-LSTM RNN	56.4	55.7	55.3	54.6
gmmSAD	57.2			

Comme cela est montré dans le tableau Tab. 3.6, la réduction du taux d'erreur vient essentiellement d'une forte réduction du nombre d'insertions (on passe de 6.4% à 3.6%) entre le système de référence *gmmSAD* et le système à base de CG-LSTM optimisé avec la fonction de coût $\mathcal{C}_3$. En regardant en détails les segments de signal qui

généraient des insertions mais qui ont été rejetés comme n'étant pas de la parole par le système à base de CG-LSTM, on s'aperçoit que ce modèle est capable d'apprendre à distinguer la parole qui génère des erreurs de la parole qui n'en produit pas.

Tableau 3.6 – Résultats détaillés du meilleur réglage pour chacun des systèmes de SAD. Les gains sur le WER proviennent essentiellement d'une réduction du nombre d'insertions.

Système de SAD	WER	Corrects	Substitutions	Élisions	Insertions
gmmSAD	57.2	49.2	34.7	16.1	6.4
CrossCor	56.1	48.8	34.4	16.8	5.0
LTSV	55.6	48.2	33.9	17.8	3.9
MLP	55.4	49.0	34.2	16.8	4.4
RNN standard	55.2	48.9	33.5	17.6	4.5
LSTM RNN	54.8	49.1	34.0	16.9	3.9
CG-LSTM RNN	54.6	49.0	34.1	16.9	3.6

Comme dans le cas de la minimisation du FER, nous avons étudié l'impact de chacune des phases de notre processus d'optimisation sur la métrique finale de manière à vérifier l'intérêt d'une stratégie d'utilisation alternée de QPSO et de SMORMS3. Le tableau Tab. 3.7 présente le WER obtenu avec différents processus d'optimisation utilisant soit QPSO ou SMORMS3 seuls, soit une combinaison des deux.

Comme pour la minimisation du FER, on observe qu'il est intéressant d'utiliser QPSO pour optimiser les paramètres de la chaîne d'extraction des MFCC avant d'optimiser les paramètres du RNN à l'aide d'une descente de gradient. De même, on voit que l'on peut encore diminuer le WER en utilisant QPSO après la descente de gradient pour réajuster les paramètres des fonctions de décision et de lissage.

Tableau 3.7 – Performance de segmentation (WER) sur le jeu de test vietnamien lorsque l'on modifie le processus d'optimisation. Pour ce test, le système de SAD utilise le modèle CG-LSTM pour estimer le critère de SAD.

Processus d'optimisation	WER sur le jeu de test
SMORMS3	55.7
SMORMS3 $\rightarrow$ *QPSO*	55.4
QPSO	56.6
QPSO $\rightarrow$ *SMORMS3*	55.0
QPSO $\rightarrow$ *SMORMS3* $\rightarrow$ *QPSO*	54.6
gmmSAD	57.2

Finalement, nous avons voulu valider tous ces travaux sur le vietnamien en optimisant, à l'aide de la fonction de coût $\mathcal{C}_3$, tous les systèmes de SAD sur trois autres

Tableau 3.8 – WER sur les jeux de test pachto, turc et tagalog du programme Babel lorsque l'on utilise la fonction de coût $\mathcal{C}_3$. Les systèmes basés sur des LSTM permettent d'obtenir les meilleures performances.

Système de SAD	WER		
	Pachto	Turc	Tagalog
gmmSAD	64.3	61.1	57.2
CrossCorr	63.9	60.6	57.1
LTSV	64.0	60.6	56.7
MLP	63.5	60.3	56.2
RNN standard	63.5	60.2	56.3
LSTM RNN	63.2	60.2	56.2
CG-LSTM RNN	63.2	60.0	56.0

langues du programme Babel : le pachto, le turc et le tagalog. Comme pour le vietnamien on observe que quel que soit le système de SAD utilisé nous parvenons à diminuer le WER d'un système de RAP quand on le compare au WER obtenu avec un système de SAD généraliste même si celui-ci a servi pour construire le système de RAP. Parmi les différentes techniques de SAD que nous avons testées, la hiérarchie des systèmes est la même que celle observée pour le vietnamien et le modèle CG-LSTM reste le plus performant et permet d'obtenir un gain de plus de 1 point de WER sur chacune des langues traitées.

3.3.3 OpenSAD'15

Le NIST organise régulièrement des évaluations ouvertes et internationales sur les différentes tâches de traitement automatique de la parole. En 2015, nous avons participé à l'évaluation OpenSAD'15 dont le but était de fournir un cadre aux développeurs de systèmes de SAD permettant de mesurer, par un organisme indépendant, la performance de leurs systèmes sur les données audio particulièrement difficiles du programme RATS de la DARPA.

Pour préparer au mieux cette évaluation, nous avons optimisé 8 systèmes de SAD différents. La table 3.9 détaille la performance de chacun de ces systèmes avec la métrique officielle (DCF) sur le jeu de validation fourni par le NIST.

Pour optimiser les différents systèmes de SAD à notre disposition, nous avons par conséquent utilisé la fonction de coût $\mathcal{C}_1$ avec un coefficient $\alpha = 0.75$.

Il est important de noter que $\mathcal{C}_1$ avec $\alpha = 0.75$ n'est pas strictement équivalente au DCF. Mais dans le cas de l'évaluation OpenSAD, les données sont des conversations téléphoniques pour lesquelles on a des durées de parole et de non-parole qui sont égales en moyenne. Dans ce cas là, le DCF et $\mathcal{C}_1$ sont très proches.

Comme on peut le voir, les systèmes de SAD basés sur des ANN améliorent très notablement la qualité de la segmentation par rapport aux techniques classiques. On

Tableau 3.9 – Résultats (DCF) obtenus sur le jeu de validation fourni par le NIST pour les différents systèmes de SAD optimisés avec la fonction de coût $\mathcal{C}_1$.

SAD	Nb de param.	DCF (%)
CrossCorr	15	12.0
LTSV	35	10.1
MLP	6000	5.3
RNN	6000	4.1
LSTM	6000	3.4
CG-LSTM	6000	3.0
CG-LSTM	60000	2.5
CG-LSTM	250000	2.0

peut également noter que le modèle CG-LSTM diminue de plus de 12% le DCF par rapport au modèle LSTM standard ce qui fait de ce modèle le meilleur système de SAD.

Nous avons ensuite optimisé des CG-LSTM plus gros qui ont permis d'améliorer encore la performance du système de SAD. En effet, passer de 6000 paramètres à 250000 nous a permis de diminuer le DCF de 30%. Malheureusement, nous n'avons pas eu le temps d'entraîner de plus gros modèle avant la date de l'évaluation officielle. [4]

Nous avons donc présenté à l'évaluation officielle nos deux meilleurs systèmes. En effet, les règles de l'évaluation autorisaient chaque participant à présenter un système principal ("primary") et un système dit de comparaison ("contrastive"). Le tableau Tab. 3.10 détaille les résultats obtenus par nos deux systèmes sur les données du jeu d'évaluation. La performance des systèmes est donnée pour les 7 canaux ainsi que sur l'ensemble du jeu de test. Nous comparons nos résultats à ceux obtenus par le meilleur système de chacun des cinq autres participants.

On observe que comme sur le jeu de validation, plus le modèle CG-LSTM a de paramètres meilleur est le résultat. Cependant, le gain en performance est moindre que celui observé lors du développement (9% de gain relatif au lieu de 20%) mais cela s'explique essentiellement par la présence du canal C inconnu sur lequel nos deux systèmes ont une performance similaire et fortement en retrait. Malheureusement, les annotations du jeu de test officiel de l'évaluation n'ont pas été rendues publiques et nous n'avons donc pas pu investiguer sur les raisons du comportement observé sur le canal C. Nous n'avons d'ailleurs pas réalisé d'autres tests et optimisations puisque nous étions dans l'incapacité de vérifier la performance d'un nouveau système sur le jeu de test.

Malgré les mauvais résultats obtenus sur le canal C, la performance sur tous les autres canaux (y compris sur le second canal inconnu A) nous a permis d'être classé troisième avec une performance quasi-identique à celle du second si l'on omet le canal

4. Les données de test n'ont malheureusement pas été rendues publiques après l'évaluation, il nous était alors impossible d'évaluer correctement de nouveaux systèmes.

Tableau 3.10 – Comparaison de la performance de nos systèmes sur le jeu de test officiel de l'évaluation OpenSAD'15 du NIST.

Système de SAD	DCF (%) par Canal							DCF (%) global
	A	B	C	E	F	G	H	
Participant #5	4.9	6.6	7.7	3.3	6.4	5.4	5.8	5.6
Participant #4	3.0	3.0	9.6	2.3	3.1	3.9	4.3	4.1
Participant #3	2.5	6.0	6.8	3.0	4.1	3.9	5.4	4.3
CG-LSTM (60k)	2.8	3.5	5.7	2.8	3.3	2.9	5.4	3.5
CG-LSTM (250k)	2.2	2.9	5.4	2.2	3.1	3.6	4.6	3.2
Participant #2	2.2	2.7	2.9	2.2	3.4	3.3	4.7	2.7
Participant #1	1.6	2.6	4.0	2.1	2.3	3.2	2.9	2.5

C. De plus, nous avons pu obtenir ce niveau de performance avec un système basé sur un unique CG-LSTM peu profond et de taille raisonnable alors que les deux premiers participants ont développé des systèmes composés d'un ANN profond par canal plus un ANN pour la détection de canal (soit au total au moins un ordre de grandeur de plus en termes de nombre de paramètres).

3.3.4 Autres types de données

Pour valider notre approche plus largement, nous avons mené trois expériences sur des données collectées dans des environnements acoustiques très différents de ceux des conversations téléphoniques de l'évaluation OpenSAD'15 et du programme Babel. Pour ces trois expériences le but final était de segmenter en locuteurs.

La segmentation en locuteurs consiste à segmenter un signal audio en tours de paroles homogènes c'est-à-dire ne contenant qu'un seul locuteur. La métrique de choix pour cette tâche est le DER qui se décompose en deux parties : le FER auquel on ajoute un terme d'erreur correspondant aux confusions entre locuteurs. Par conséquent, pour optimiser un système de segmentation en locuteurs il est préférable de minimiser le FER du système de SAD.

Résultats

Le tableau Tab. 3.11 regroupe les résultats obtenus sur les trois corpus qui sont détaillés dans la section §1.2.2 : REPERE, AMI et Game of Thrones. Pour ces trois corpus, nous affichons le FER obtenu sur le jeu de test pour chacun des systèmes de SAD optimisé avec la fonction de coût $\mathcal{C}_1$ et un coefficient $\alpha = 0.5$. Nous donnons également la performance obtenue sans le lissage des segments (c'est-à-dire pas de padding, ni de suppression des segments ou des silences courts).

Nous pouvons ainsi voir que comme sur les données de type téléphonique les systèmes de SAD utilisant des réseaux de neurones sont toujours plus performants que les techniques classiques lorsque la fonction de lissage est utilisée. On note cependant

que la bonne performance du MLP repose beaucoup sur la fonction de lissage puisque sans cette fonction les taux d'erreurs obtenus sont supérieurs à ceux de LTSV sur les corpus AMI et Game of Thrones. Au contraire, les réseaux de neurones récurrents qui exploitent l'information contextuelle pour prendre leurs décisions reposent beaucoup moins sur la fonction de lissage. La performance des deux modèles à base de LSTM est de loin la moins sensible à la présence ou non de lissage. Pour finir, dans ces trois expériences les meilleures performances sont toujours obtenues avec le modèle CG-LSTM que nous avons introduit dans la section §2.1.2.

Tableau 3.11 – Performances des différents systèmes de SAD sur les données de test des trois corpus REPERE, AMI et Game of Thrones. Le modèle CG-LSTM permet d'obtenir la meilleure performance dans tous les cas. C'est également le modèle qui repose le moins sur la fonction de lissage pour minimiser le taux d'erreurs.

Système de SAD	FER					
	REPERE		AMI		Game of Thrones	
	~~lissage~~	lissage	~~lissage~~	lissage	~~lissage~~	lissage
CrossCor	44.99	17.94	24.09	8.34	30.41	18.62
LTSV	17.12	16.67	9.62	8.29	15.59	14.83
MLP	17.05	15.62	11.41	6.84	17.89	12.36
RNN standard	16.44	15.45	6.55	6.31	12.72	12.71
LSTM RNN	13.45	13.37	6.20	5.97	11.00	10.97
CG-LSTM RNN	12.90	12.66	5.93	5.92	11.00	10.98

3.4 Conclusion

Dans ce chapitre nous avons introduit une méthode permettant d'optimiser la paramétrisation d'un système complet de détection de la parole (analyse spectrale, réseau de neurones et fonction de décision et de lissage) dans le but de minimiser le taux d'erreur d'un système de reconnaissance en aval. Cette méthode innovante repose sur l'algorithme d'optimisation heuristique QPSO et l'algorithme de descente de gradient SMORMS3, ainsi que sur la fonction de coût de simili-WER que nous avons conçue dans le but d'estimer le WER réel sans avoir à décoder à chaque itération du processus d'optimisation tout le corpus avec le système de reconnaissance.

Les expérimentations que nous avons réalisées sur le corpus vietnamien du programme Babel de l'IARPA montrent que cette fonction de coût permet d'obtenir un gain significatif sur le taux d'erreur de 2.5% relatifs en moyenne par rapport à la fonction de coût FER habituellement utilisée. Et ce, quel que soit le système de SAD utilisé parmi les six testés.

Lorsque l'on compare les techniques de SAD à proprement parler, nous avons mené diverses expériences avec des données très différentes et les résultats montrent que le

modèle CG-LSTM que nous avons introduit au chapitre 2 permet toujours d'obtenir la meilleure performance. Quand nous le comparons à d'autres modèles neuronaux tels qu'un MLP, un RNN standard ou un LSTM le modèle CG-LSTM permet de gagner respectivement 37%, 14% ou 3% relatifs en moyenne sur le FER obtenu sur quatre corpus très différents : Babel, REPERE, AMI et Game of Thrones. Sur les données particulièrement difficiles du programme RATS de la DARPA, les gains obtenus sur le DCF sont respectivement de 43%, 27% et 12%.

Pour finir, grâce au processus d'optimisation introduit dans ce chapitre nous avons pu obtenir un gain significatif en reconnaissance de la parole pour quatre langues différentes du programme Babel : le pachto, le turc, le tagalog et le vietnamien. Nous avons également pu être classé troisième de la campagne d'évaluation OpenSAD'15 organisée par le NIST avec un niveau de performance très proche du système classé second tout en ayant dix à cent fois moins de paramètres.

Chapitre 4

Apprentissage D&C pour les problèmes multi-classe

Après avoir étudié l'intérêt des RNN dans un contexte de classification binaire pour la détection de la parole, nous nous sommes intéressés à des problèmes multi-classe c'est-à-dire à des tâches de classification avec un nombre de classes supérieur à deux.

Le passage de la classification binaire à un nombre de classes supérieur à 2 peut se faire de plusieurs façons. On peut par exemple transformer le problème à n classes en n classifications binaires "une-contre-toutes" ou $n(n-1)/2$ classifications binaires "une-contre-une" de manière à pouvoir utiliser l'un des nombreux algorithmes de classification binaire existants tels que une machine à vecteurs de support, une régression logistique, un classifieur bayésien ou un réseau de neurones. On peut également utiliser une approche hiérarchique pour laquelle on utilise des classifieurs binaires pour parcourir un arbre binaire dont les feuilles correspondent aux n classes cibles. Pour finir, on peut traiter directement le problème avec les algorithmes qui s'y prêtent comme c'est le cas pour les réseaux de neurones ayant une couche de sortie *softmax*.

Dans ce chapitre nous introduisons une méthode d'apprentissage s'inspirant de la maxime *diviser pour mieux régner* qui allie deux de ces trois approches pour entraîner un réseau de neurones multi-classe. Nous appelons cette méthode D&C (Divide and Conquer). Le problème multi-classe est d'abord décomposé en n problèmes de classification binaire "une-contre-toutes" que l'on traite avec n petits réseaux de neurones. Ces n réseaux sont ensuite utilisés pour construire un réseau de neurones multi-classe dont on affine les paramètres pour améliorer les performances de classification.

Nous commençons par détailler l'approche D&C puis nous la comparons à l'approche standard pour entraîner un réseau de neurones multi-classe. Pour effectuer cette comparaison nous avons choisi de travailler sur les évaluations LRE-07 et LRE-15 organisées par le NIST dont le but était d'identifier pour chaque séquence audio la langue ou le dialecte parlés parmi plus d'une dizaine de choix.

4.1 Diviser pour régner (D&C)

Nous nous plaçons ici dans le cas d'une utilisation classique d'un ANN comme classifieur multi-classe dans lequel le nombre de neurones de la couche de sortie correspond au nombre de classes à distinguer avec pour fonction d'activation la fonction *softmax*

(voir section 1.1.1). Ainsi, la sortie du réseau de neurones est un vecteur de réels compris entre 0 et 1 dont la somme est égale à 1 que l'on interprète comme le vecteur de probabilités de chacune des classes cibles connaissant le signal audio.

Pour entraîner un tel réseau de neurones nous utilisons la fonction de coût la plus communément utilisée c'est-à-dire la fonction de coût dite d'entropie croisée telle que nous l'avons présentée dans la section §1.1.1 avec l'équation Eq. 1.11. Cette fonction de coût est particulièrement adaptée au problème de classification et a l'avantage de se combiner avantageusement avec la matrice jacobienne de la fonction d'activation *softmax* pour donner une formulation très simple du gradient de la fonction de coût en amont de la fonction d'activation *softmax*.

Lors de nos premiers tests nous avons été confrontés à quelques difficultés pour optimiser les RNN pour des tâches multi-classe. En particulier, on observait en début d'optimisation une phase durant laquelle le taux d'erreur évolue peu avant de réellement diminuer, suggérant que l'optimisation des paramètres des RNN semblait pâtir de l'initialisation aléatoire. Pour remédier à ce problème nous proposons une méthode d'apprentissage en quatre étapes qui s'attache à construire un meilleur point de départ pour l'optimisation d'un RNN multi-classe en décomposant le problème global en n tâches de classification binaire. Les quatre étapes sont les suivantes :

1. Pour chaque classe c parmi les n classes cibles, on entraîne un petit RNN à distinguer la classe c de toutes les autres classes. Ce classifieur binaire a exactement la même architecture que le classifieur multi-classe que l'on souhaite entraîner au final mais en ayant $1/n$ le nombre de neurones dans chacune des couches du réseau de neurones. Comme nous nous plaçons dans le cadre d'une classification binaire (une classe contre toutes) pour cette première étape, la couche de sortie n'a qu'un unique neurone et sa fonction de transfert est la fonction *logistique*. Ces petits RNN ne sont entraînés que sur 200 itérations de descente de gradient.

2. Les n petits RNN sont combinés pour former un plus grand RNN multi-classe. Pour ce faire, les matrices de poids des petits RNN sont assemblées sous forme de matrices diagonales par blocs :

$$\boldsymbol{W_{multi}} = \begin{bmatrix} \boldsymbol{W_{bin_1}} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & \boldsymbol{W_{bin_n}} \end{bmatrix} \tag{4.1}$$

   et les vecteurs de biais ou de "peepholes" sont concaténés :

$$\boldsymbol{b_{multi}} = \begin{bmatrix} \boldsymbol{b_{bin_1}} \\ \vdots \\ \boldsymbol{b_{bin_n}} \end{bmatrix} \tag{4.2}$$

   Il convient toutefois de noter que pour la première couche du RNN qui traite les

entrées il faut juste concaténer les matrices de poids :

$$\boldsymbol{W}_{multi} = \begin{bmatrix} \boldsymbol{W}_{bin_1} \\ \vdots \\ \boldsymbol{W}_{bin_n} \end{bmatrix} \tag{4.3}$$

Avec cet assemblage on obtient un RNN multi-classe contenant n canaux indépendants de traitement de l'information qui ont été entraînés à chacun différencier une classe de toutes les autres. Le RNN obtenu au final a une architecture similaire à celle des petits RNN mais avec des dimensions n fois plus grandes pour toutes ses couches.

3. Comme les n petits réseaux sont entraînés indépendamment les uns des autres, la moyenne et la variance des valeurs obtenues au niveau des n neurones de la couche de sortie *softmax* du réseau multi-classe peuvent varier grandement. Pour équilibrer le poids de chacun des n neurones de sortie dans les décisions finales de classification, le réseau de sortie (voir Fig. 1.8) est adapté seul pendant 100 itérations. Durant cette étape, les gradients ne sont pas rétro-propagés dans les couches récurrentes et leurs paramètres sont gardés constants. L'impact de cette calibration est montré sur la figure Fig. 4.1 où l'on observe, avant et après calibration, les distributions des valeurs obtenues en sortie du réseau de neurones multi-classe.

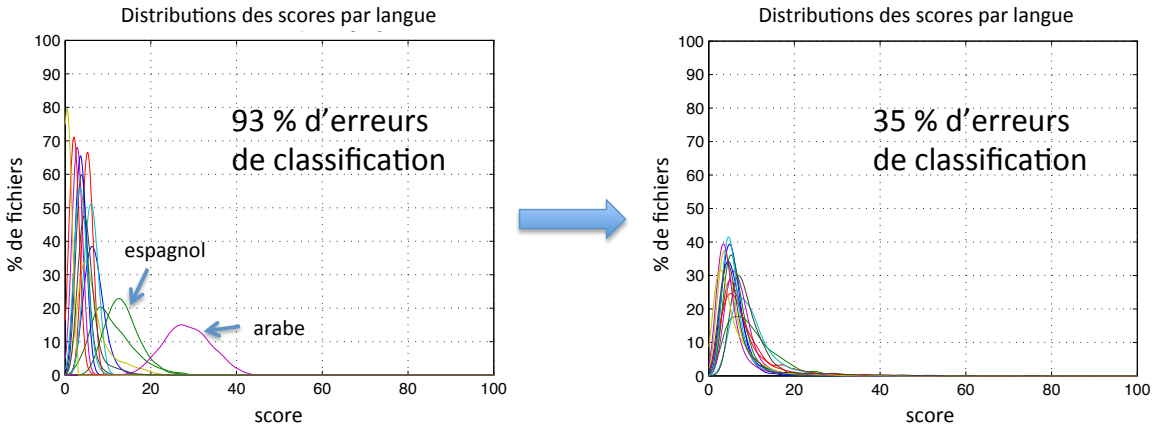


FIGURE 4.1 – Calibration des poids relatifs entre les différents modèles de classe au sein d'un RNN multi-classe lors de la troisième étape de la méthode diviser pour régner (D&C).

4. Pour finir, tous les paramètres du RNN multi-classe sont ré-estimés en prenant comme initialisation les paramètres obtenus à la fin de l'étape précédente. Pour faciliter l'apprentissage, les paramètres en dehors des blocs diagonaux dans les matrices recombinées ne sont pas exactement mis à zéro mais sont initialisés de façon aléatoire en utilisant une distribution suivant une loi normale de moyenne nulle et de faible écart type (10^{-6}).

L'apprentissage classique d'un ANN pour une tâche de classification multi-classe consiste à ne réaliser que l'étape 4 du processus que nous venons de décrire à partir de paramètres initialisés aléatoirement comme expliqué dans la section §6.2.3 (p. 115).

4.2 Expérimentations

Cette section présente les résultats de deux expérimentations multi-classe. Plus spécifiquement, nous avons travaillé sur les données de deux évaluations internationales organisées par le NIST : les 14 langues de l'évaluation LRE-07 et les 20 langues et dialectes de l'évaluation LRE-15. Nous montrons ici l'intérêt de la méthode D&C pour estimer efficacement les paramètres du modèle CG-LSTM pour les tâches d'identification de la langue.

Le détail des conditions expérimentales (corpus et métriques) dans lesquelles nous nous sommes placés est donné dans la section §1.3.2 (p. 35).

4.2.1 Utilisation des RNN pour l'identification de la langue

Comme nous l'avons dit plus haut, nous utilisons des RNN dont la couche de sortie est de dimension n égale au nombre de classes cibles et a pour fonction d'activation la fonction *softmax*. Ainsi, pour toute séquence d'entrée on obtient une séquence de vecteur de réels de dimension n dont les composantes sont toutes comprises entre 0 et 1 et dont la somme vaut 1. Nous utilisons ces vecteurs comme estimations à chaque instant des probabilités a posteriori de chacune des classes. Pour obtenir une unique décision par fichier audio à partir des séquences obtenues en sortie de l'ANN on calcule la moyenne géométrique de l'ensemble des vecteurs de probabilités. D'autres méthodes ont été envisagées mais c'est la moyenne géométrique qui donne les meilleurs résultats.

Comme pour la détection de parole, dans toutes nos expérimentations avec des RNN nous avons utilisé une architecture bidirectionnelle telle que décrite dans la section 1.1.4 (p. 21) de manière à pouvoir exploiter le contexte passé et futur autour de chaque instant pour estimer au mieux les probabilités a posteriori. Dans les systèmes d'IdL dont nous présentons les résultats dans cette partie le nombre de couches RNN empilées est de deux. L'ajout de couches supplémentaires augmente la durée d'apprentissage sans apporter de gain significatif.

Pour l'ensemble des expérimentations que nous présentons dans cette section, nous utilisons exclusivement l'algorithme de descente de gradient SMORMS3 pour optimiser les paramètres des ANN sur le jeu de données d'apprentissage. En effet, sur cette tâche difficile pour laquelle l'exploitation d'un contexte long terme est essentielle pour obtenir de bons résultats, nous n'avons pas réussi à obtenir des résultats intéressants en optimisant le système avec une méthode heuristique telle que QPSO comme nous avions pu le faire pour les tâches de détection de la parole. Cela est en partie dû au fait que l'espace de recherche est de dimension bien plus grande puisque pour obtenir une performance à l'état de l'art les réseaux de neurones que nous utilisons ont nettement plus de paramètres que ceux que nous avons entraînés pour le SAD (on passe de quelques milliers de paramètres à plus d'un million).

Les entrées fournies aux différents ANN testés sont sous la forme de séquences de 8 coefficients PLP avec leurs dérivées premières et secondes (ce qui donne des vecteurs de caractéristiques de dimension 24). Ces coefficients sont calculés toutes les 10 ms sur les segments de parole. Et pour finir, on applique une normalisation VTLN et une normalisation CMVN sur l'ensemble des coefficients issus d'un même fichier audio. Ces pré-traitements du signal audio sont réalisés de manière identique sur les jeux

d'apprentissage, de validation et de test.

Enfin, comme nous l'explicitons dans la section §6.2.5 (p. 116), il est intéressant de découper les séquences audio en sous-séquences de taille fixe qui se chevauchent de manière à améliorer la robustesse des décisions prises à chaque pas de temps. Pour l'identification de la langue nous avons trouvé que le meilleur compromis performance/charge de calcul (voir section §5.3.2, p. 97) était d'utiliser des sous-séquences de 3.2s (soit des sous-séquences de 320 vecteurs d'entrée) qui se chevauchent de 75% (soit une nouvelle sous-séquence tous les 80 vecteurs d'entrée).

4.2.2 Résultats pour l'évaluation LRE-07

Pour valider notre nouvelle méthode nous avons entraîné un système d'IdL basés sur des CG-LSTM en faisant varier la méthode d'apprentissage (D&C ou standard). Les résultats obtenus sur le jeu de test de l'évaluation LRE-07 pour chacune des trois durées [1] de parole et sur l'ensemble des segments de test sont présentés dans le tableau Tab. 4.1. Le système basé sur des CG-LSTM a une taille d'environ 400k paramètres. En comparaison, les systèmes phonotactiques et *i-vector* que nous avons introduits dans la section §1.3.1 et qui nous servent de références ont tous les deux plus de 10 millions de paramètres.

On peut voir sur ce tableau que la méthode d'apprentissage D&C permet d'améliorer notablement la performance par rapport à l'apprentissage standard avec des gains relatifs de 14% sur le LER [2], de 6% sur l'EER [3] et de 14% sur le Cavg [4]. Elle permet même d'obtenir un système plus performant que le système *i-vector* en particulier sur les segments les plus courts avec un gain de 10% relatif.

En revanche, on voit que le système phonotactique est bien meilleur que tous les autres systèmes ce qui s'explique au moins en partie par le fait que les décodeurs de phones ont été entraînés sur des jeux de données volumineux et variés (cf. 1.3.1), et permettent ainsi de mieux gérer les variabilités telles que celles liées au locuteur ou au canal sur un corpus d'apprentissage tels que ceux des évaluations organisées par le NIST.

Pour finir, il est intéressant de noter que la combinaison des sorties du système basé sur des CG-LSTM avec les sorties du système phonotactique permet de diminuer significativement les taux d'erreurs sur toutes les durées de test et tout particulièrement sur les segments très courts. Cette combinaison de système fonctionne mieux que celle obtenue en fusionnant les sorties du système phonotactique avec celles du système *i-vector*. Toutes ces combinaisons de systèmes sont réalisées en calculant la moyenne géométrique des probabilités a posteriori données par chacun des systèmes comme nous l'avons décrit dans la section §1.3.1.

1. Pour l'évaluation LRE-07, le NIST évaluait les systèmes sur trois conditions de test distinctes : 3 s, 10 s et 30 s de parole par fichier de test.

2. Taux d'erreur d'identification de la langue (ou Language Error Rate, voir p. 36).

3. Taux d'égale erreur (Equal Error Rate) qui correspond au point de fonctionnement d'un système pour lequel le taux de fausses acceptations est égal au taux de faux rejets.

4. Métrique dite du coût de performance moyen (Cost average). Elle est utilisée comme métrique principale pour les évaluations organisées par le NIST.

TABLEAU 4.1 – Résultats (LER, EER et Cavg) sur le jeu de test de l'évaluation LRE-07 lorsque l'on modifie la méthode d'apprentissage d'un système acoustique basé sur des CG-LSTM. Les résultats sont comparés à ceux obtenus avec les systèmes de référence (phonotactique (PHO) et *i-vector* (IVC)). Les résultats obtenus en combinant les sorties des différents systèmes sont également présentés. Lorsque l'on combine deux systèmes la sortie finale est obtenue en calculant la moyenne géométrique des probabilités a posteriori données par chacun des systèmes.

	3 sec			10 sec			30 sec			global		
Système	LER	EER	CAVG	LER	EER	CAVG	LER	EER	CAVG	LER	EER	CAVG
PHO	34.53	12.79	18.59	11.66	4.21	6.28	2.48	0.79	1.34	16.22	5.99	8.73
IVC	45.68	18.43	24.60	18.92	8.30	10.19	6.25	3.30	3.36	23.61	10.21	12.72
CG-LSTM (Std)	46.81	16.41	25.21	20.00	7.29	10.77	9.95	4.22	5.36	25.59	9.73	13.78
CG-LSTM (D&C)	42.11	15.57	22.67	17.56	6.81	9.45	6.08	3.25	3.28	21.92	9.11	11.80
PHO + IVC	31.76	13.21	17.10	8.84	3.74	4.76	1.74	0.74	0.94	14.12	5.99	7.60
PHO + CG-LSTM (D&C)	28.40	9.91	15.29	8.44	3.06	4.54	2.41	0.55	1.30	13.08	4.62	7.04

4.2.3 Résultats pour l'évaluation LRE-15

Nous présentons maintenant les résultats obtenus lors de notre préparation pour l'évaluation LRE-15. Nous comparons les systèmes de référence (phonotactique et *i-vector*) avec les systèmes basés sur des CG-LSTM entraînés avec la méthode D&C et un apprentissage standard sur trois jeux de données présentés dans la section §1.3 : jeux de développement, jeu de test officiel et jeu de test post-évaluation.

Résultats sur les jeux de développement

Les résultats obtenus sur les deux jeux de données de développement décrits p. 38 sont présentés dans le tableau Tab. 4.2. De façon similaire à ce que l'on a pu voir pour LRE-07, l'apprentissage D&C améliore notablement la performance du système d'IdL basé sur des CG-LSTM avec un gain relatif de plus de 16% sur les taux d'erreurs sur les segments courts et de 21% sur les segments longs. On obtient de cette manière un meilleur système que les deux systèmes de référence (PHO : phonotactique et IVC : *i-vector*).

Sur les segments courts, le meilleur système acoustique (CG-LSTM entraînés avec la méthode D&C) affiche des performances bien meilleures que le système phonotactique avec un gain de 25% sur le LER. Il est toutefois intéressant de noter que la combinaison de ce système acoustique avec le système phonotactique permet d'obtenir une diminution importante du taux d'erreur avec un gain relatif de 26% sur le LER par rapport au meilleur système utilisé seul. On peut aussi voir que lorsque l'on ajoute le système *i-vector* dans la combinaison, il est possible d'améliorer encore les performances.

Tableau 4.2 – Résultats obtenus sur les deux jeux de données de développement (segments courts et longs) de l'évaluation LRE-15 pour chacun des trois systèmes d'IdL pris individuellement et pour les combinaisons de ces systèmes. Pour le système basé sur des CG-LSTM on présente les résultats obtenus avec l'apprentissage standard (Std) et l'apprentissage D&C. Dans ce tableau, le système phonotactique est nommé PHO et le système *i-vector* est appelé IVC.

Système d'identification	Segments courts (3s < - < 10s)			Segments longs (10s < - < 50s)		
	LER	*EER*	*CAVG*	*LER*	*EER*	*CAVG*
PHO	25.4	13.1	0.151	7.5	2.9	0.044
IVC	19.5	11.2	0.120	5.7	2.5	0.029
CG-LSTM (Std)	22.7	12.4	0.130	7.5	3.7	0.038
CG-LSTM (D&C)	19.0	11.8	0.118	5.9	3.4	0.031
PHO + CG-LSTM (D&C)	14.0	8.1	0.080	2.3	1.6	0.012
PHO + IVC + CG-LSTM (D&C)	12.4	6.7	0.073	2.4	1.5	0.014

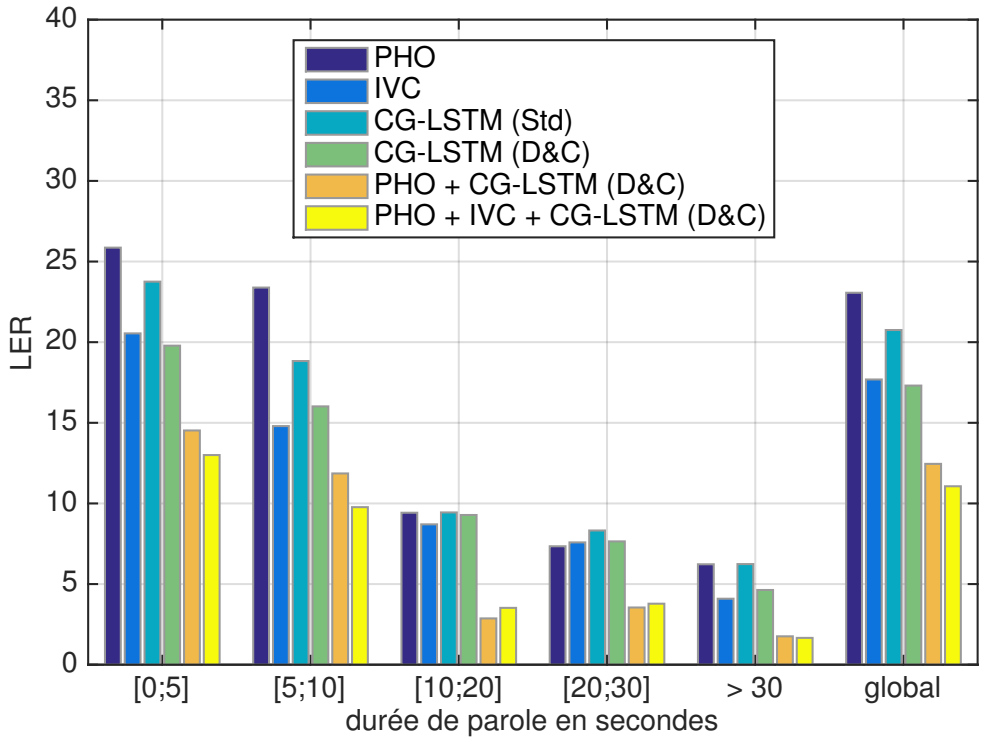


FIGURE 4.2 – LER obtenus sur l'ensemble des segments des deux jeux de développement de l'évaluation LRE-15 en les regroupant selon des intervalles de durée de parole.

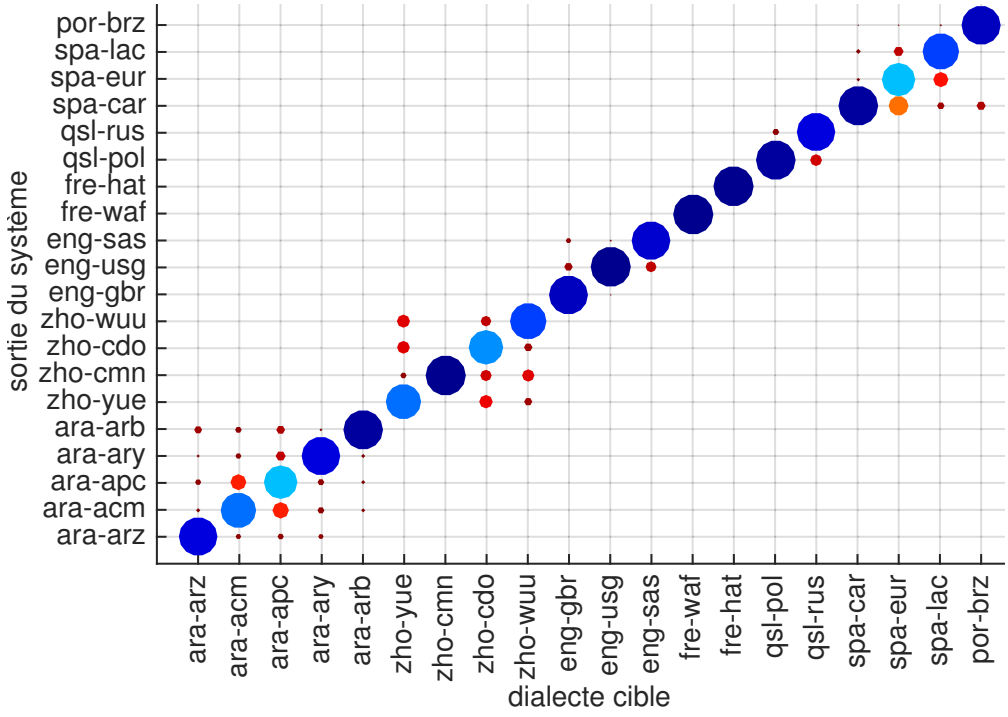


FIGURE 4.3 – Matrice de confusion pour l'ensemble des segments des deux jeux de développement de l'évaluation LRE-15. La surface et la couleur des cercles sont indicatives du pourcentage de segments bien ou mal classifiés.

Pour ce qui est des segments longs, les résultats des systèmes individuels sont bien meilleurs que sur les segments courts (taux d'erreur divisé par 4) mais la hiérarchie des différents systèmes est la même que pour les segments courts. On observe un gain de performance identique entre le système phonotactique et le système basé sur des CG-LSTM entraîné avec la méthode D&C. En revanche, la combinaison de ces deux systèmes apporte un gain bien plus important que sur les segments courts puisque le LER est réduit de 50% par rapport à celui obtenu avec le meilleur système. Contrairement à ce qui était observé sur les segments courts l'ajout du système *i-vector* dans la combinaison n'a que peu d'impact sur la performance.

L'impact de la quantité de parole sur le LER moyen de chacun des systèmes d'IdL est illustré sur la figure Fig. 4.2. Pour obtenir ce graphe nous avons combiné les résultats obtenus sur les deux jeux de développement en les regroupant selon des intervalles de durée de parole. On peut voir que la performance des systèmes acoustiques (RNN et *i-vector*) se dégrade moins rapidement lorsque la durée de parole diminue que pour l'approche phonotactique. Pour les durées de parole supérieures à 10 s les performances des trois systèmes sont très similaires.

Le tableau Tab. 4.3 détaille pour chacun des groupes de langues la performance en matière de Cavg pour la meilleure combinaison de systèmes : PHO+RNN+IVC. On peut voir immédiatement que le Cavg pour les groupes anglais, français et slave est significativement inférieur à celui obtenu sur les trois autres groupes. Le plus grand nombre de confusions observées sur les groupes arabe, chinois et ibérique est visualisable sur la figure Fig. 4.3 où les confusions internes au sein des différents groupes sont représentées. Et on peut également voir qu'au sein du groupe ibérique, le brésilien est très facilement séparé des trois variantes de l'espagnol ce qui s'explique aisément par le fait qu'il y a plus de différences entres deux langues qu'entre deux dialectes.

Tableau 4.3 – Cavg obtenus avec la meilleure combinaison de systèmes pour chacun des six groupes de langues sur les données combinées des deux jeux de données de développement (8680 segments) de l'évaluation LRE-15.

Cavg	*ara*	*zho*	*eng*	*fre*	*qsl*	*spa*
PHO + IVC + CG-LSTM (D&C)	0.12	0.14	0.03	0.001	0.06	0.12

Résultats sur le jeu d'évaluation

Les résultats en matière de LER, EER et Cavg obtenus pour le jeu officiel de test fourni par le NIST sont présentés dans le tableau Tab. 4.4. On peut voir que les performances de chacun des systèmes sont très nettement inférieures à celles obtenues sur les jeux de développement. Après analyse, nous avons pu identifier deux facteurs à l'origine de cette forte dégradation. D'une part, pour plusieurs langues ou dialectes il y a un fort décalage entre les données d'apprentissage et de développement d'un côté et celles de test de l'autre. D'autre part, dans le jeu de test, la proportion de segments courts ayant une durée de parole inférieure à 5 s est très importante ($\approx$ 40%). Par ailleurs, certains de ces segments courts ne contiennent que des rires ou des exclamations.

TABLEAU 4.4 – Résultats obtenus sur le jeu de données de test officiel de l'évaluation LRE-15 pour chacun des trois systèmes d'IdL pris individuellement et pour les combinaisons de ces systèmes. Pour le système basé sur des CG-LSTM on présente les résultats obtenus avec l'apprentissage standard (Std) et l'apprentissage D&C. Dans ce tableau, le système phonotactique est nommé PHO et le système *i-vector* est appelé IVC.

System	*LER*	*EER*	*CAVG*
PHO	27.8	16.9	0.209
IVC	36.2	22.7	0.268
CG-LSTM (Std)	40.2	28.8	0.298
CG-LSTM (D&C)	38.4	27.1	0.282
PHO + CG-LSTM (D&C)	27.6	19.2	0.207
PHO + IVC + CG-LSTM (D&C)	27.6	19.1	0.207

Dans ces conditions, le système phonotactique est clairement le plus robuste et obtient la meilleure performance avec plus de 10% d'écart absolu avec les systèmes acoustiques. Cette robustesse est en grande partie due au fait que les trois décodeurs de phones ont été entraînés sur des données présentant une très grande variété de conditions acoustiques et de locuteurs. Cela permet au système phonotactique de mieux généraliser le problème en particulier lorsque les données de test sont très éloignées des données d'apprentissage. On peut toutefois noter que pour ce qui concerne l'haïtien, les données de test étant tellement éloignées des données d'apprentissage (ce n'est en fait pas la même langue entre les deux jeux de données), le système phonotactique ne peut faire guère mieux que les systèmes acoustiques (voir le tableau Tab. 4.5).

On note cependant que comme sur les jeux de développement l'apprentissage D&C améliore la performance du système d'IdL basé sur des CG-LSTM. Mais compte tenu du décalage entre les données d'apprentissage et de test et de la dégradation de performance correspondante, le gain obtenu est nettement plus faible que celui observé sur les jeux de développement.

Pour finir, compte tenu du mauvais comportement de tous les systèmes acoustiques, la combinaison de systèmes n'apporte rien par rapport au système phonotactique seul.

La performance mesurée par le LER en fonction des quantiles de la durée de parole est présentée sur la figure Fig. 4.4. Comparé à ce que l'on a pu observer sur les jeux de développement, l'impact de la durée de parole est moins prononcé, ce qui s'explique évidemment par les très mauvaises performances obtenues sur certaines langues telles que le français haïtien et ce quelle que soit la durée de parole.

Pour chacun des groupes de langues, la performance en matière de Cavg obtenue avec la meilleure combinaison de systèmes est détaillée dans le tableau Tab. 4.5. Et les confusions internes au sein des différents groupes de langues sont représentées sur la figure Fig. 4.5. On peut ainsi voir que le nombre de confusions augmente considérablement pour tous les groupes de langues à l'exception des dialectes chinois et des langues slaves. Pour les autres groupes, on observe un fort décalage entre les jeux d'apprentissage et de test. Ceci est tout particulièrement vrai pour le français dans lequel

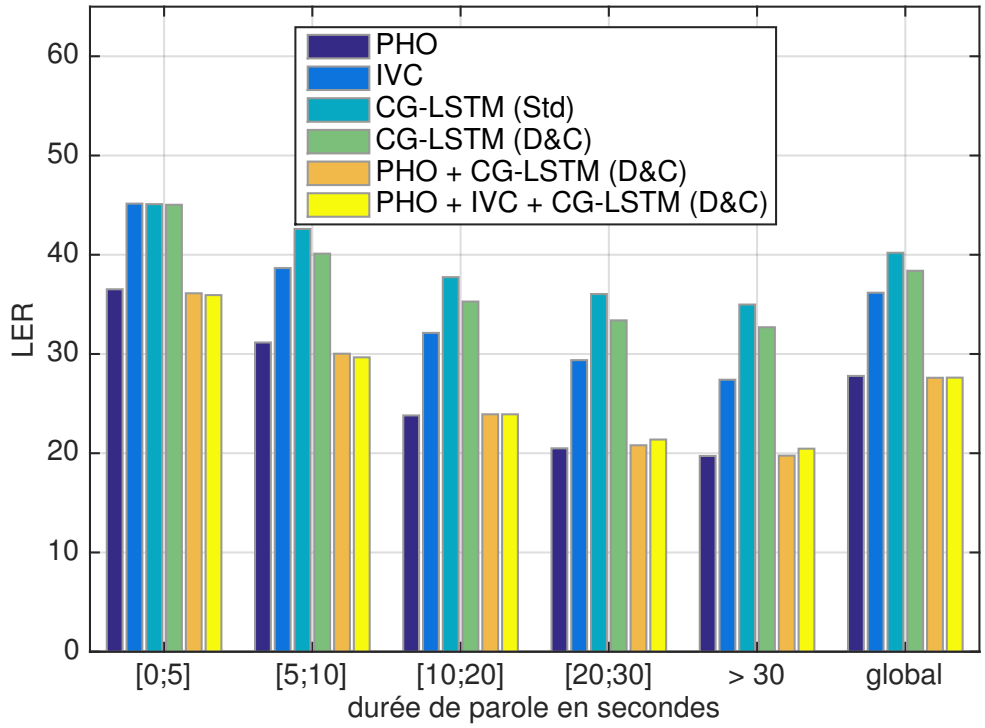


FIGURE 4.4 – LER obtenus sur l'ensemble des segments du jeu de test officiel de l'évaluation LRE-15 en les regroupant selon des intervalles de durée de parole.

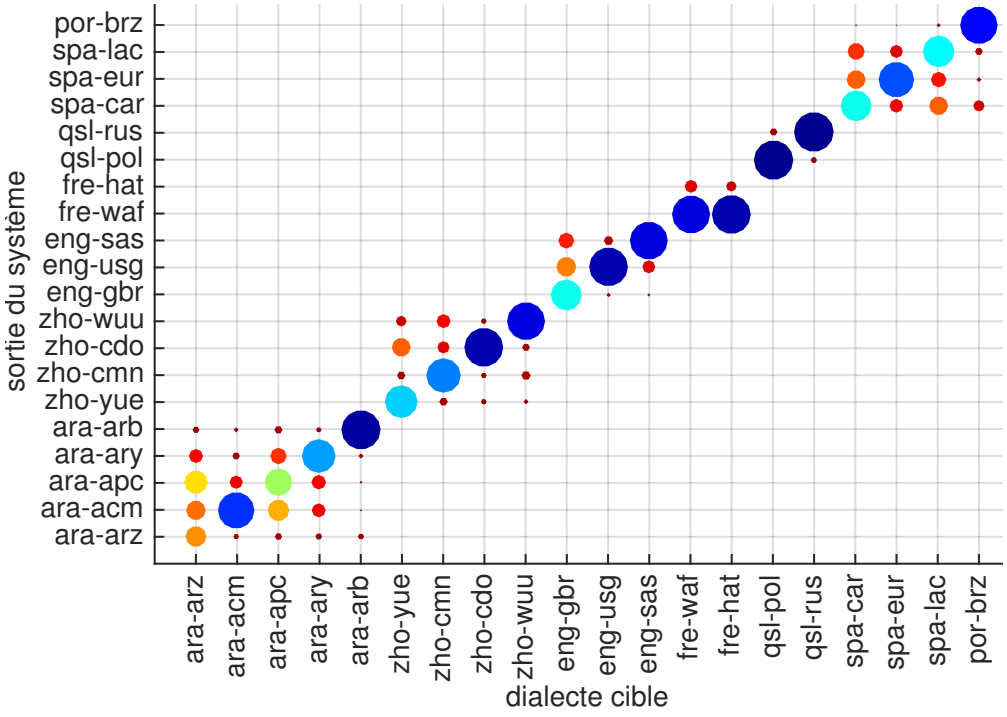


FIGURE 4.5 – Matrice de confusion pour l'ensemble des segments du jeu de test officiel de l'évaluation LRE-15. La surface et la couleur des cercles sont indicatives du pourcentage de segments bien ou mal classifiés.

la quasi-totalité des fichiers étiquetés haïtien sont mal classifiés. En ce qui concerne le groupe anglais, la détérioration du Cavg par rapport aux jeux de développement s'explique essentiellement par la pauvreté du jeu d'apprentissage pour l'anglais britannique résultant naturellement en une mauvaise modélisation de cette variante anglaise par rapport aux deux autres (américaine et indienne). On voit ainsi sur la matrice de confusion que le nombre de bonnes classifications pour "eng-gbr" est moitié moindre que pour les autres variantes anglaises.

Tableau 4.5 – Cavg obtenus avec la meilleure des combinaisons de systèmes pour chacun des six groupes de langues sur les données du jeu de données de test officiel de l'évaluation LRE-15.

Cavg	*ara*	*zho*	*eng*	*fre*	*qsl*	*spa*
PHO + IVC + CG-LSTM (D&C)	0.23	0.14	0.12	0.51	0.03	0.21

Résultats post-évaluation

Compte tenu du fort décalage entre les jeux d'apprentissage et de test, ainsi que de la très grande quantité de données disponibles dans le jeu de test officiel, nous avons décidé de tirer aléatoirement 10% des fichiers du jeu de test pour les ajouter au jeu d'apprentissage. Les trois systèmes d'IdL (phonotactique, *i-vector* et CG-LSTM) ont alors été ré-entraînés sur ce nouveau jeu d'apprentissage avant d'être testés sur les 90% restants du jeu de test officiel. Les résultats de cette expérimentation post-évaluation sont donnés dans le tableau Tab. 4.6 et démontrent clairement qu'en réduisant le décalage entre les données d'apprentissage et les données de test, le comportement des systèmes d'IdL est bien plus proche du comportement observé sur les jeux de développement. Ainsi, le système phonotactique et les systèmes acoustiques fonctionnent avec un niveau de performance proche de celui obtenu lors de études de développement. La seule différence avec les tests sur les jeux de développement est que la performance du système phonotactique est proche de celle obtenue avec le système basé sur des CG-LSTM et entraîné avec la méthode D&C, et meilleure que celle obtenue avec le système *i-vector*. En revanche, comme lors des développements la combinaison des systèmes acoustique et phonotactique est très bénéfique avec un gain relatif de performance en matière de LER de 33%.

On note que dans cette expérimentation, comme lors de nos tests sur les jeux de développement, l'apprentissage D&C améliore très notablement la performance du système d'IdL basé sur des CG-LSTM avec un gain relatif de 26% sur le LER. Grâce à cette méthode le système basé sur des CG-LSTM devient le meilleur système devant les deux systèmes de référence.

La figure Fig. 4.6 permet de visualiser l'impact de la quantité de parole sur la performance des différents systèmes d'IdL et de leurs combinaisons en matière de LER sur le jeu de test post-évaluation contenant 90% du jeu de test officiel de l'évaluation LRE-15. On observe là encore un comportement bien plus similaire à ce que l'on a observé sur les deux jeux de développement (cf. Fig 4.2) qu'au comportement des systèmes sur le jeu de test officiel.

Tableau 4.6 – Résultats obtenus sur le jeu de données de test post-évaluation contenant 90% du jeu de test officiel de l'évaluation LRE-15 pour chacun des trois systèmes d'IdL pris individuellement et pour les combinaisons de ces systèmes. Pour le système basé sur des CG-LSTM on présente les résultats obtenus avec l'apprentissage standard (Std) et l'apprentissage D&C. Dans ce tableau, le système phonotactique est nommé PHO et le système *i-vector* est appelé IVC.

System	*LER*	*EER*	*CAVG*
PHO	23.5	10.1	0.151
IVC	26.6	10.4	0.174
CG-LSTM (Std)	30.9	13.4	0.208
CG-LSTM (D&C)	22.8	8.4	0.146
PHO + CG-LSTM (D&C)	16.2	5.7	0.100
PHO + IVC + CG-LSTM (D&C)	15.5	5.4	0.095

Pour chacun des groupes de langues, la performance en matière de Cavg obtenue avec la meilleure combinaison de systèmes est détaillée dans le tableau Tab. 4.7. La figure Fig. 4.7 permet de visualiser les confusions internes au sein des différents groupes de langues. En comparaison avec les résultats obtenus sur le jeu de test officiel, le nombre de confusions est grandement réduit pour la plupart des groupes de langues. L'amélioration relative du Cavg atteint 58% pour le chinois et 75% pour le français ce qui illustre très clairement l'inadéquation du jeu d'apprentissage. On peut voir en revanche que les dialectes de l'arabe restent le groupe le plus difficile avec un Cavg de 0.2 et de nombreuses confusions entre les dialectes égyptien et syro-libanais.

Tableau 4.7 – Cavg obtenus avec la meilleure des combinaisons de systèmes pour chacun des six groupes de langues sur les données du jeu de données de test post-évaluation contenant 90% du jeu de test officiel de LRE-15.

Cavg	*ara*	*zho*	*eng*	*fre*	*qsl*	*spa*
PHO + IVC + CG-LSTM (D&C)	0.19	0.06	0.05	0.13	0.02	0.12

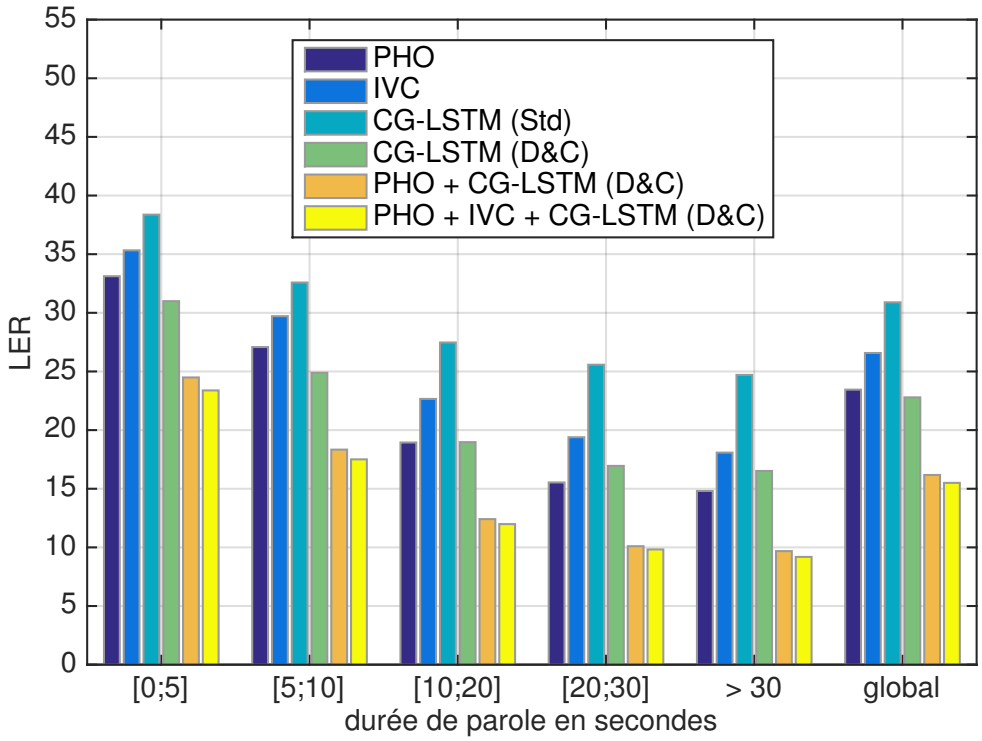


FIGURE 4.6 – LER obtenus sur l'ensemble des segments du jeu de test post-évaluation contenant 90% du jeu de test officiel de LRE-15 en les regroupant selon des intervalles de durée de parole.

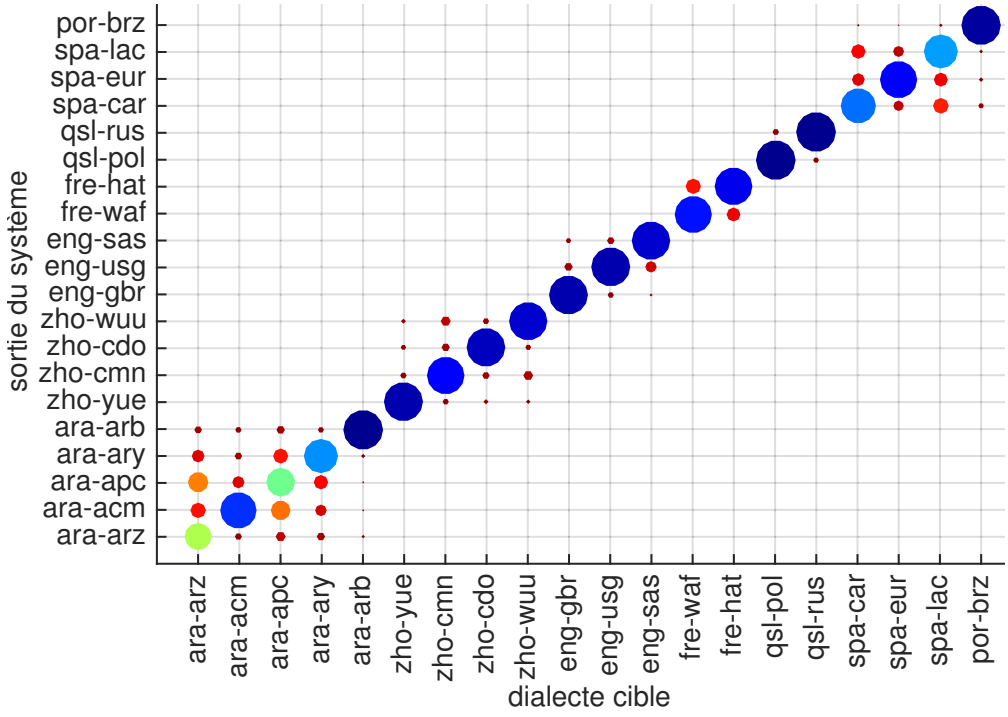


FIGURE 4.7 – Matrice de confusion pour l'ensemble des segments du jeu de test post-évaluation de LRE-15. La surface et la couleur des cercles sont indicatives du pourcentage de segments bien ou mal classifiés.

4.3 Conclusion

Dans ce chapitre nous avons introduit la méthode d'apprentissage D&C qui décompose un problème multi-classe en n classifications binaires "une-contre-toutes" pour initialiser l'apprentissage d'un réseau de neurones multi-classe et ainsi atteindre un meilleur niveau de performance.

Nous avons appliqué cette méthode à deux tâches d'IdL, celles de deux évaluations internationales LRE-07 et LRE-15 organisées par le NIST. Sur les deux évaluations la méthode D&C permet d'améliorer significativement les performances obtenues avec un système d'IdL utilisant des CG-LSTM. On obtient en effet une amélioration du LER de 14% relatifs sur LRE-07 et de 26% relatifs sur LRE-15.

Grâce à cette méthode nous obtenons un système d'IdL meilleur qu'un système *i-vector* sur les deux évaluations et meilleurs qu'un système phonotactique sur la tâche difficile de distinction des langues proches de l'évaluation LRE-15. De plus, nous obtenons ce niveau de performance avec un nombre de paramètres inférieur d'au moins un ordre de grandeur par rapport aux deux systèmes de référence (phonotactique et *i-vector*).

Un trait intéressant qui ressort également de nos développements est que la combinaison d'un système acoustique à base de RNN et d'un système phonotactique est très efficace. En effet, la complémentarité de ces deux approches permet d'obtenir des gains notables sur les taux d'erreurs qui peuvent aller jusqu'à 30% relatifs par rapport au meilleur système seul.

Chapitre 5

Projection vectorielle pour les problèmes multi-classe

En partant du constat que dans une architecture RNN standard le nombre de couches empilées (couches RNN + MLP) dilue les gradients calculés lors de la rétro-propagation et freine la convergence tout particulièrement en début d'apprentissage, nous avons proposé la méthode D&C présentée au chapitre précédent. Cette méthode s'attaque au problème en divisant la tâche principale en sous-tâches plus simples que l'on résout avec de petits RNN. On peut ensuite recombiner ces petits RNN pour mieux initialiser le RNN que l'on souhaite utiliser comme classifieur final et diminuer le taux d'erreur.

Dans ce chapitre nous proposons une nouvelle architecture RNN qui permet de projeter une séquence d'entrée de longueur variable sur un unique vecteur de taille fixe dont la direction est représentative de la classe. Conjointement, nous proposons une fonction de coût dite de proximité angulaire permettant d'entraîner un tel RNN pour des tâches multi-classe. Cette fonction de coût permet de minimiser l'écart angulaire entre deux projections vectorielles issues de séquences de la même classe tout en maximisant l'écart angulaire entre deux projections vectorielles issues de séquences de classes différentes.

5.1 Projection vectorielle par RNN

Nous introduisons ici une architecture neuronale différente de l'architecture classiquement utilisée pour les RNN, qui permet de s'attaquer plus directement aux difficultés d'apprentissage pour les problèmes multi-classe. Cette architecture permet d'extraire un unique vecteur de taille fixe à partir de n'importe quelle séquence d'entrée (voir figure Fig. 5.1b). Par rapport à l'architecture classique (voir figure Fig. 5.1a), nous avons commencé par supprimer le MLP pour réduire le nombre total de couches. Puis de façon à obtenir des gradients plus homogènes dans l'ensemble du réseau, nous avons relié directement toutes les couches du RNN directement à la sortie du réseau. En effet, dans cette architecture la séquence obtenue en sortie du RNN correspond à la concaténation des séquences de sortie de chacune des couches. Cela nous permet également d'obtenir une sortie du RNN qui combine les différents niveaux d'abstraction du réseau et exploite ainsi toute l'information disponible. Ensuite on calcule la moyenne

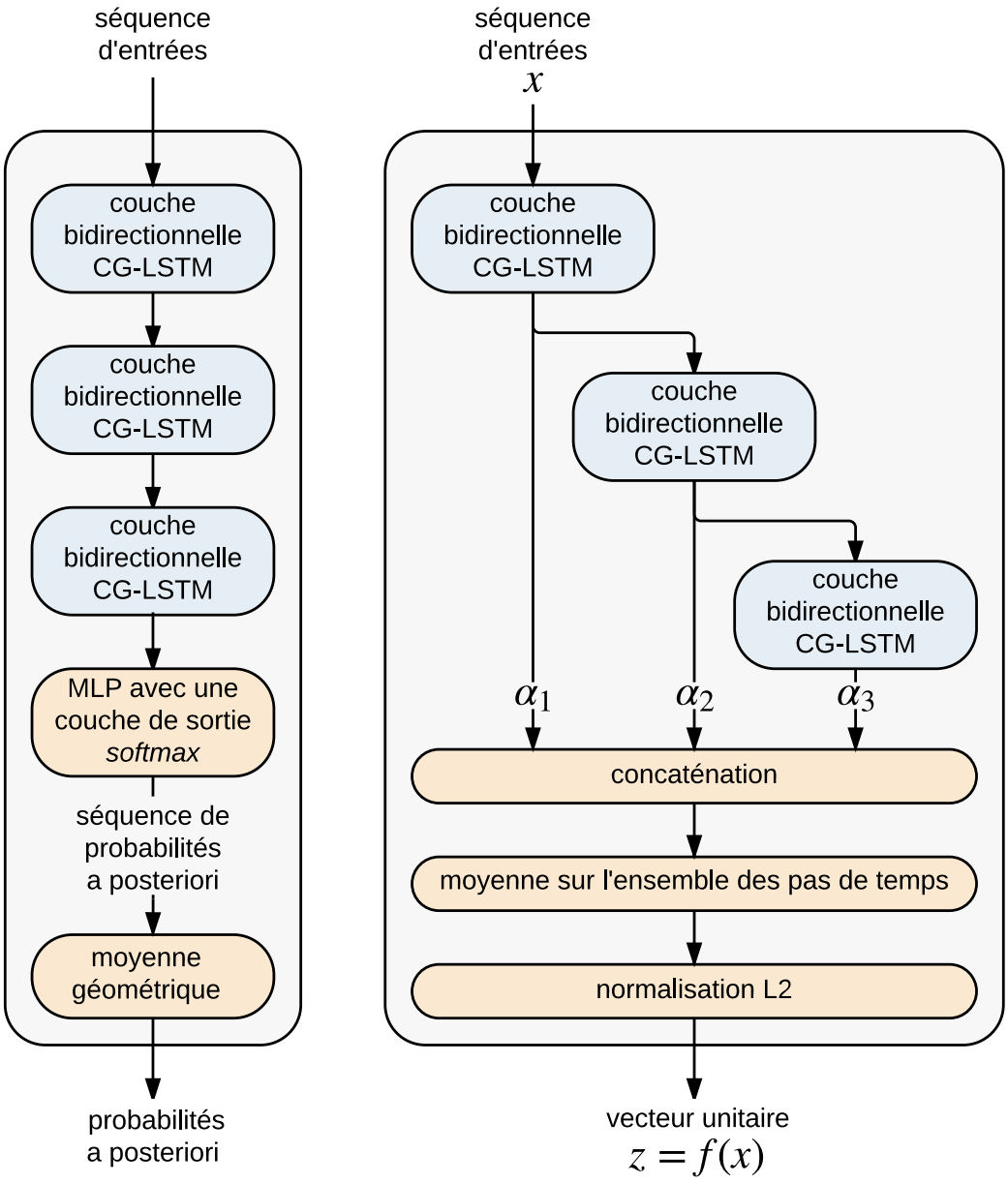


FIGURE 5.1 – Architectures utilisées pour les tâches de LID : a) Architecture standard d'un RNN où l'on empile les couches RNN puis on utilise un MLP pour déterminer les probabilités a posteriori d'appartenance à chacune des classes pour chacun des pas de temps. Pour obtenir les probabilités a posteriori pour l'ensemble de la séquence, on calcule la moyenne géométrique des probabilités obtenues à chaque pas de temps. b) Architecture d'un RNN donnant une représentation vectorielle de la séquence de données d'entrées sous la forme d'un vecteur unitaire de taille égale à la somme des dimensions de toutes les couches du RNN.

des vecteurs concaténés au cours du temps et on normalise le vecteur obtenu de façon à obtenir un vecteur qui se déplace sur l'hypersphère unité (vecteur de norme égale à 1). La dimension du vecteur de sortie est alors égale à la somme des dimensions de chacune de ses couches.

Comme on peut le voir sur la figure Fig. 5.1b les séquences de sortie des différentes couches sont pondérées par un coefficient α_i avant la concaténation. Ces coefficients α_i sont des paramètres du RNN qui sont optimisés par rétro-propagation des gradients comme les autres paramètres du RNN, et permettent d'ajuster l'importance de chacune des couches du RNN dans le vecteur de sortie sans pour autant contraindre directement les sorties des différentes couches (on verra l'intérêt des ces coefficients dans la section §5.3).

5.2 Fonction de coût de proximité angulaire

Avec l'architecture présentée dans ce chapitre on obtient un unique vecteur de norme égale à 1 pour chaque séquence d'entrées. L'information contenue dans ce vecteur de sortie réside donc dans sa direction et non dans sa norme. Pour entraîner ce type de RNN, nous utilisons une fonction de coût particulière qui minimise l'écart angulaire entre des vecteurs représentant des séquences d'entrées appartenant à la même classe et maximise l'écart angulaire entre des vecteurs représentant des séquences d'entrées appartenant à des classes différentes.

En effet, nous inspirant des succès de la fonction de coût appelée *triplet-loss* introduite dans [129] et après avoir effectué des expérimentations sur la détection de changement de locuteur (voir [130]), nous avons mis au point une fonction de coût dite de *proximité angulaire*. Cette fonction de coût utilise des vecteurs référents $\boldsymbol{r}_l, l \in [\![1, N]\!]$ dont la dimension correspond à celle de la sortie du RNN utilisé. Chacun de ces vecteurs correspond alors à une direction dans l'espace vectoriel de sortie du RNN qui va nous servir de référence pour chacune des classes à identifier. Pour déterminer l'hypothèse de la classe la plus probable correspondant à un vecteur de sortie unitaire $\boldsymbol{z}$, on commence par déterminer l'écart angulaire entre $\boldsymbol{z}$ et chacune des directions de référence à l'aide de l'équation suivante

$$\theta_l(\boldsymbol{z}) = \arccos(\delta_l(\boldsymbol{z})) \tag{5.1}$$

où δ_l est le produit scalaire entre $\boldsymbol{z}$ et le vecteur unité suivant la direction de $\boldsymbol{r}_l$

$$\delta_l(\boldsymbol{z}) = \frac{\boldsymbol{r}_l}{||\boldsymbol{r}_l||} \cdot \boldsymbol{z} \tag{5.2}$$

L'hypothèse de la classe la plus probable renvoyée par le système est alors donnée par

$$l^* = \underset{l' \in [\![1,N]\!]}{\operatorname{argmin}}\, \theta_{l'}(\boldsymbol{z}) \tag{5.3}$$

Et on définit le coût de *proximité angulaire* $\mathcal{C}(\boldsymbol{z}, l)$ pour $\boldsymbol{z}$ appartenant à la classe l par

$$\mathcal{C}(\boldsymbol{z}, l) = \sum_{l' \in [\![1,N]\!]}^{l' \neq l} \sigma(\theta_l(\boldsymbol{z}) - \theta_{l'}(\boldsymbol{z})) \tag{5.4}$$

où σ est la fonction logistique qui permet d'accélérer et d'améliorer la convergence en concentrant l'effort d'apprentissage sur les cas proches des frontières entre classes.

Pour la rétro-propagation du gradient cette fonction de coût peut être dérivée et on obtient ainsi

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{z}} = \sum_{l' \in [1,N]}^{l' \neq l} \Delta_{ll'}(\boldsymbol{z}) \left(\frac{\boldsymbol{r}_{l'}}{||\boldsymbol{r}_{l'}|| \sqrt{1 - \delta_{l'}(\boldsymbol{z})^2}} - \frac{\boldsymbol{r}_l}{||\boldsymbol{r}_l|| \sqrt{1 - \delta_l(\boldsymbol{z})^2}} \right) \tag{5.5}$$

avec

$$\Delta_{ll'}(\boldsymbol{z}) = \sigma(\theta_l(\boldsymbol{z}) - \theta_{l'}(\boldsymbol{z}))(1 - \sigma(\theta_l(\boldsymbol{z}) - \theta_{l'}(\boldsymbol{z}))) \tag{5.6}$$

qui correspond à la contribution de la fonction logistique aux dérivées partielles. Mais on a également pour $l' \in [1, N]$ et $l' \neq l$

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{r}_{l'}} = \left(\frac{\Delta_{ll'}(\boldsymbol{z})}{\sqrt{1 - \delta_{l'}(\boldsymbol{z})^2}} \boldsymbol{z} \right) \boldsymbol{J}(\boldsymbol{r}_{l'}) \tag{5.7}$$

et pour finir

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{r}_l} = \left(- \sum_{l' \in [1,N]}^{l' \neq l} \frac{\Delta_{ll'}(\boldsymbol{z})}{\sqrt{1 - \delta_l(\boldsymbol{z})^2}} \boldsymbol{z} \right) \boldsymbol{J}(\boldsymbol{r}_l) \tag{5.8}$$

où $\boldsymbol{J}(\boldsymbol{r}_l)$ est la matrice jacobienne correspondant à la normalisation L2 du vecteur $\boldsymbol{r}_l$.

Les deux dernières dérivées partielles nous permettent d'utiliser le même algorithme de descente de gradient pour optimiser à la fois les vecteurs référents $\boldsymbol{r}_l$ et les paramètres du RNN qui projette les séquences d'entrées dans l'espace vectoriel des vecteurs référents.

5.3 Expérimentations

Cette section présente les résultats de nos diverses expérimentations avec la projection vectorielle par RNN. En particulier, nous avons travaillé sur les données des deux évaluations organisées par le NIST en 2007 et 2015 : les 14 langues de LRE-07 et les 20 langues et dialectes de LRE-15. Nous montrons ici l'intérêt de la méthode de projection qui, couplée à la fonction de coût de proximité angulaire, permet d'accélérer l'apprentissage et d'améliorer la classification. Pour ce faire, nous la comparons à la méthode d'apprentissage classique pour une tâche de classification multi-classe (couche de sortie *softmax* couplée à la fonction de coût d'entropie croisée) mais également à la méthode D&C présentée dans le chapitre 4 (p. 77).

Le détail des conditions expérimentales (corpus et métriques) dans lesquelles nous nous sommes placés est donné dans la section §1.3.2.

5.3.1 Utilisation des RNN pour l'identification de la langue

Comme pour les expérimentations présentées dans le chapitre 4 et pour les mêmes raisons (cf. section §4.2.1), nous utilisons exclusivement l'algorithme de descente de gradient SMORMS3 pour optimiser les paramètres des ANN sur le jeu de données d'apprentissage.

De même, de façon à obtenir une comparaison objective de l'approche de projection vectorielle avec celles présentées dans le chapitre 4 nous utilisons exactement les mêmes entrées (cf. section 4.2.1). Et nous utilisons le même nombre de couches RNN (2) ayant la même taille que dans le chapitre précédent sauf pour quelques tests où cela est mentionné explicitement.

Enfin, comme dans le chapitre 4 nous découpons les séquences audio originales en sous-séquences de 3.2s (soit des sous-séquences de 320 vecteurs d'entrée) qui se chevauchent de 75% (soit une nouvelle sous-séquence tous les 80 vecteurs d'entrée). L'étude et le choix des valeurs optimales pour ces deux paramètres sont présentés dans la section §5.3.2.

5.3.2 Résultats pour l'évaluation LRE-07

Classification binaire

Pour commencer, nous avons validé la méthode de projection vectorielle sur le cas d'une classification binaire que nous avons déjà utilisé pour comparer différents ANN (cf. section 2.2.2), c'est-à-dire séparer une langue (l'arabe) de toutes les autres. Pour que la comparaison soit équitable nous utilisons les mêmes entrées pour tous les ANN testés et nous fixons le nombre de paramètres pour chacun des réseaux à 50k. Toutes les couches des RNN testés sont bidirectionnelles. Nous comparons les résultats obtenus avec l'architecture classique (cf. figure Fig. 5.1a) entraînée avec la fonction de coût d'entropie croisée et l'architecture de projection vectorielle (cf. figure Fig. 5.1b) entraînée avec la fonction de coût de *proximité angulaire* présentée plus haut.

Tableau 5.1 – Comparaison de différents ANN pour séparer une langue (arabe) des 13 autres langues de LRE-07. Les ANN ayant une architecture standard (Std) sont entraînés à l'aide de la fonction de coût d'entropie croisée, ceux ayant une architecture de projection vectorielle (PV) sont entraînés à l'aide de la fonction de coût dite de *proximité angulaire*. La dernière ligne démontre l'intérêt d'ajouter les cas pires observés au cours de l'apprentissage dans les mini-batchs d'entraînement.

Type d'ANN	Cas pires ajoutés [1]	Architecture		LER sur 2 classes		
		Std	PV	3 s	10 s	30 s
LSTM	✓	✓		27.5	14.4	8.4
CG-LSTM	✓	✓		26.2	13.4	7.5
LSTM	✓		✓	25.9	12.3	7.2
CG-LSTM	✓		✓	**24.1**	**11.2**	**5.0**
CG-LSTM			✓	29.0	20.0	10.2

1. Lors de la phase d'entraînement nous utilisons des *mini-batch* de quelques centaines d'exemples tirés aléatoirement mais nous y ajoutons également une centaine de cas pires observés depuis le début de l'apprentissage. L'intérêt de ces cas pires est de forcer le modèle à travailler sur les frontières entre les différentes classes à discriminer plutôt que sur l'espace en entier. Avec cette stratégie, on obtient

Le tableau Tab. 5.1 détaille les résultats obtenus sur le jeu de test de l'évaluation LRE-07. On peut ainsi noter que quelle que soit l'architecture utilisée (et donc la fonction de coût utilisée), le modèle CG-LSTM apporte un gain de performance significatif par rapport au modèle LSTM standard. Par ailleurs, l'intérêt d'utiliser l'architecture de projection vectorielle avec la fonction de coût de *proximité angulaire* est évident puisque les gains de performance sont très importants allant jusqu'à 33% sur les fichiers de test contenant 30 s de parole. Ceci est d'ailleurs vrai quel que soit le modèle utilisé : LSTM standard ou CG-LSTM.

Pour finir, les deux dernières lignes du tableau Tab. 5.1 montrent très clairement qu'il est crucial de conserver les cas pires dans les mini-batchs lors de l'entraînement pour obtenir la meilleure performance (cf. section §6.2.4). En effet, ne pas le faire double les taux d'erreurs sur les fichiers de 10 s et 30 s de parole.

Durant ces premiers tests, nous avons également voulu quantifier l'impact de la stratégie de découpage des séquences de signal audio en sous-séquences se chevauchant (cf. section §6.2.5). Le tableau 5.2 détaille les résultats en matière de taux d'erreurs sur l'ensemble des trois jeux de tests de l'évaluation LRE-07. On voit ainsi clairement l'intérêt de découper les séquences puisque pour toutes les valeurs des deux paramètres de découpage les taux d'erreurs sont toujours inférieurs à celui obtenu sur les séquences d'origine (colonne *inf* dans le tableau). Au vu de nos tests, la meilleure taille pour les sous-séquences est 3.2s quel que soit le taux de recouvrement choisi. Finalement, nous avons retenu un taux de recouvrement de 75% puisque passer à 90% apporte certes un léger gain de performance mais double le temps de calcul par rapport à un taux de recouvrement de 75%.

Tableau 5.2 – Taux d'erreur moyen sur les jeux de test de l'évaluation LRE-07 pour la tâche de classification binaire (arabe ou non) lorsque l'on fait varier la taille et le chevauchement des sous-séquences audio traitées par le système de LID à base de projection vectorielle par CG-LSTM.

% de recouvrement	Durée des sous-séquences				
	1 s	3.2 s	5 s	10 s	*inf*
0	16.7	15.4	18.3	19.2	19.8
25	16.7	15.1	17.4	18.8	-
50	17.1	14.6	16.2	17.9	-
75	17.2	14.4	15.0	17.9	-
90	17.0	14.2	14.6	17.5	-

une meilleure généralisation et donc une meilleure performance sur le jeu de test comme le montre la dernière ligne du tableau.

Classification multilingue

Une fois les tests monolingues réalisés nous avons entraîné un système multilingue basé sur de la projection vectorielle (PV) par CG-LSTM et nous l'avons comparé aux systèmes que nous avons présentés dans le chapitre précédent (apprentissage standard et méthode D&C). Le tableau Tab. 5.3 présente les résultats obtenus sur le jeu de test de l'évaluation LRE-07 pour chacune des trois durées de parole mais aussi sur l'ensemble des segments de test. Tous les systèmes basés sur des CG-LSTM ont une taille similaire d'environ 400k paramètres. On rappelle que les systèmes phonotactiques et i-vector ont tous les deux plus de 10 millions de paramètres.

On peut rapidement voir que la projection vectorielle par CG-LSTM entraînée à l'aide de la fonction de coût de proximité angulaire permet d'améliorer notablement les performances d'identification par rapport à l'utilisation et l'apprentissage standards d'un RNN mais aussi par rapport à la méthode présentée au chapitre 4. En effet, cette nouvelle méthode amène un gain de 8% sur le LER et le Cavg et de 18% sur l'EER pour l'ensemble des trois jeux de tests ce qui en fait un système d'IdL bien meilleur que le système *i-vector* et qui se rapproche du niveau de performance du système phonotactique.

Comme on l'avait déjà observé dans le chapitre précédent, la combinaison des sorties du meilleur système basé sur des CG-LSTM avec les sorties du système phonotactique permet de diminuer significativement les taux d'erreurs sur toutes les durées de test puisque l'on gagne en matière de LER par rapport au meilleur système seul : 20% sur les segments de 3 s, 34% sur les segments de 10 s, 26% sur les segments de 30 s. Comme toujours dans cette thèse cette combinaison de systèmes est réalisée en calculant la moyenne géométrique des probabilités a posteriori données par chacun des systèmes.

Par ailleurs, si l'on analyse les directions obtenues en sorties de notre système de projection vectorielle, on peut aisément visualiser la proximité des langues entre elles et les difficultés d'identification. Une représentation dans le plan des séquences audio des trois jeux de test de l'évaluation LRE-07 est visible sur la figure Fig. 5.2. Cette représentation en deux dimensions est obtenue à l'aide de l'algorithme tSNE [131] directement à partir des vecteurs unitaires en sortie du RNN. Sur cette figure, on peut voir que les langues sont relativement bien séparées dans ce plan et sont groupées par proximité linguistique. Par exemple, les langues asiatiques sont regroupées sur la gauche avec un "îlot" japonais-coréen reflétant la similarité linguistique des deux langues. On remarque que les langues les plus difficiles à identifier se retrouvent au centre de la représentation. On voit ainsi que l'hindi est de loin la langue se rapprochant du plus grand nombre d'autres langues avec une proximité aux langues de la même région (bengali, tamil) mais également de langues européennes telles que l'espagnol et l'anglais.

Ensuite nous avons voulu vérifier l'intérêt des coefficients α_i dans l'architecture de projection vectorielle. Nous avons fait plusieurs tests avec différentes configurations de l'architecture de projection vectorielle : un système ayant six couches RNN ou un système ayant deux couches RNN. Pour chacune des configurations, nous avons réalisé un apprentissage tel que celui décrit dans ce chapitre avec des coefficients α_i libres et entraînés par descente de gradient. Ensuite, nous avons réalisé le même apprentissage en fixant tous les coefficients α_i à 0 à l'exception du coefficient de la dernière couche qui est lui fixé à un. On obtient dans ce cas une architecture qui se rapproche de l'architecture classique dans laquelle la sortie du RNN se base uniquement sur sa dernière couche.

TABLEAU 5.3 – Résultats sur le jeu de test de l'évaluation LRE-07 lorsque l'on utilise un système acoustique basé sur de la projection vectorielle (PV) par CG-LSTM. Les résultats sont comparés à ceux obtenus avec les systèmes de référence (phonotactique (PHO) et *i-vector* (IVC)) ainsi que les résultats obtenus avec une architecture RNN classique (couches RNN empilées et couche de sortie *softmax*) entraînée avec l'apprentissage standard (Std) ou l'apprentissage D&C présentée au chapitre 4. Les résultats obtenus en combinant les sorties des meilleurs systèmes sont également présentés. Lorsque l'on combine deux systèmes la sortie finale est obtenue en calculant la moyenne géométrique des probabilités a posteriori données par chacun des systèmes.

	3 sec			10 sec			30 sec			global		
Système	LER	EER	CAVG	LER	EER	CAVG	LER	EER	CAVG	LER	EER	CAVG
PHO	34.53	12.79	18.59	11.66	4.21	6.28	2.48	0.79	1.34	16.22	5.99	8.73
IVC	45.68	18.43	24.60	18.92	8.30	10.19	6.25	3.30	3.36	23.61	10.21	12.72
CG-LSTM (Std)	46.81	16.41	25.21	20.00	7.29	10.77	9.95	4.22	5.36	25.59	9.73	13.78
CG-LSTM (D&C)	42.11	15.57	22.67	17.56	6.81	9.45	6.08	3.25	3.28	21.92	9.11	11.80
CG-LSTM (PV)	40.40	14.65	21.75	14.44	4.92	7.77	5.85	1.61	3.15	20.23	7.43	10.89
PHO + CG-LSTM (D&C)	28.40	9.91	15.29	8.44	3.06	4.54	2.41	0.55	1.30	13.08	4.62	7.04
PHO + CG-LSTM (PV)	27.39	9.41	14.74	7.73	2.46	4.16	1.84	0.46	0.99	12.32	4.17	6.63

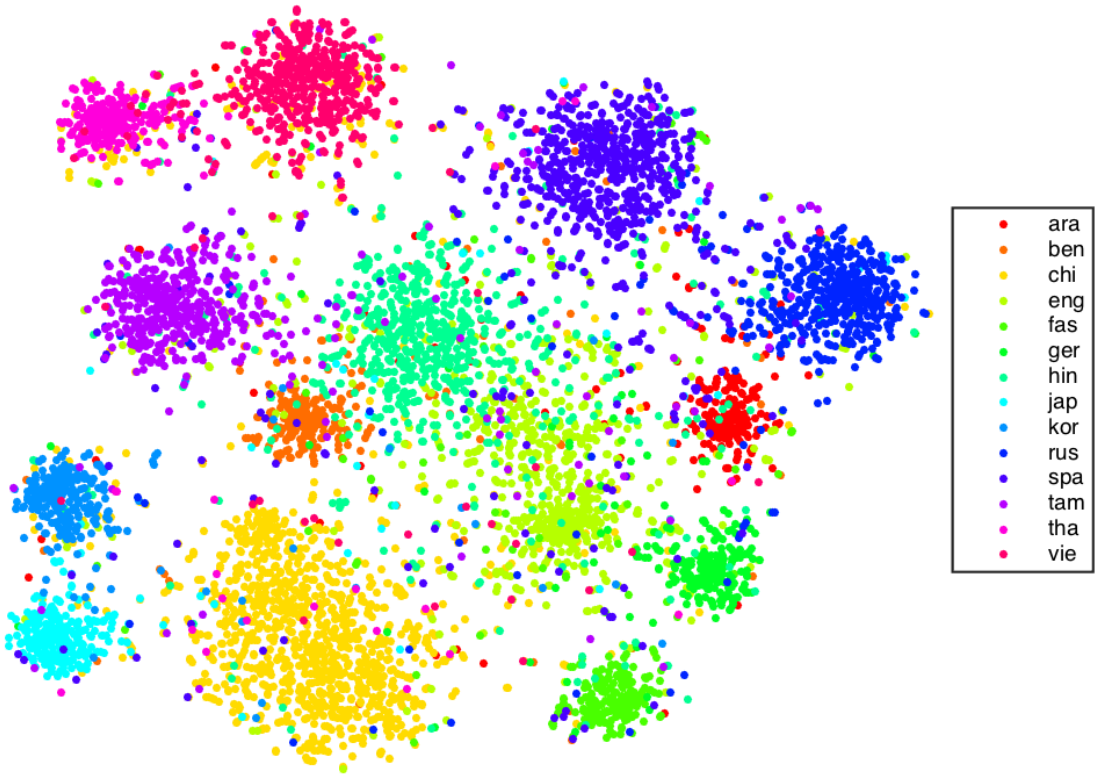


FIGURE 5.2 – Représentation des langues dans le plan obtenue à l'aide de l'algorithme tSNE à partir des projections vectorielles des trois jeux de test de l'évaluation LRE-07.

TABLEAU 5.4 – Résultats sur le jeu de test de l'évaluation LRE-07 pour deux architectures de projection vectorielle : un système ayant six couches CG-LSTM ou un système ayant deux couches CG-LSTM. Pour chacune de ces deux configurations, on entraîne le système en fixant tous les α_i à zéro sauf le dernier, puis en fixant tous les α_i à un, et pour finir en les optimisant comme tous les autres paramètres du RNN.

Nombre de couches dans le RNN	Contrainte sur les coefficients α_i	global		
		LER	EER	CAVG
6	$\alpha_{i\in[1,N-1]} = 0$	92.86	56.46	50.00
6	$\alpha_{i\in[1,N]} = 1$	28.69	10.64	15.45
6	-	23.25	8.22	12.52
2	$\alpha_{i\in[1,N-1]} = 0$	21.92	7.91	11.81
2	$\alpha_{i\in[1,N]} = 1$	21.27	7.69	11.45
2	-	20.23	7.43	10.89

Pour finir, nous avons testé le cas où tous les coefficients α_i sont fixés à 1 pour pouvoir quantifier l'intérêt de les optimiser.

Le tableau Tab. 5.4 présente l'ensemble des résultats et on peut voir que l'utilisation des coefficients α_i facilite l'apprentissage et permet d'obtenir de meilleures performances. C'est particulièrement vrai dans le cas d'un RNN à six couches pour lequel nous avons été dans l'incapacité de lui faire apprendre quoi que ce soit lorsque l'on fixe les cinq premiers α_i à zéro alors que nous obtenons un très bon système d'IdL lorsque les coefficients α_i sont optimisés conjointement avec les autres paramètres du RNN. Nous n'avons pas observé le même comportement sur l'architecture à deux couches puisque nous arrivons à l'entraîner quelques soient les contraintes sur les coefficients α_i ce qui s'explique essentiellement par le fait que les gradients des paramètres de la première couche du RNN sont beaucoup moins ténus dans une architecture à deux couches que dans une architecture à six couches. Nous observons toutefois un gain significatif à optimiser les α_i comme les autres paramètres du RNN quel que soit le nombre de couches utilisées. Ce gain est d'autant plus important que le nombre de couches est grand.

La figure Fig. 5.3 montre l'évolution des coefficients α_i au cours de l'apprentissage pour un RNN à six couches. On voit ainsi que dans un premier temps tous les coefficients restent au même niveau : toutes les couches ont alors le même poids dans la représentation vectorielle. Dans un second temps on observe les coefficients des premières couches diminuer fortement pour donner plus de poids aux couches plus profondes du RNN. Cette évolution dénote un transfert du pouvoir discriminant des premières couches du réseau (proches des entrées et donc avec une plus faible capacité d'abstraction) vers les dernières couches profitant d'une plus grande capacité de modélisation. La première phase durant laquelle les coefficients des premières couches sont élevés permet donc essentiellement de faciliter et d'accélérer l'apprentissage en homogénéisant les gradients dans toutes les couches du RNN. On note toutefois que les coefficients des premières couches sont atténués mais pas annulés ce qui suggère que l'information discriminante est répartie dans tout le réseau.

La figure Fig. 5.4 présente un comparatif de la durée d'apprentissage d'un RNN lorsque l'on utilise l'architecture standard (couches empilées) avec l'apprentissage standard ou la méthode D&C introduite dans le chapitre 4, et l'architecture de projection vectorielle entraînée avec la fonction de coût de proximité angulaire introduites dans ce chapitre. Le gain de temps est évident puisqu'on observe une diminution du nombre de mini-batchs à présenter au système de 50% par rapport à la méthode D&C et de 66% par rapport à l'apprentissage standard pour atteindre un niveau de performance similaire voire légèrement meilleur. On peut en effet observer que l'architecture de projection vectorielle converge bien plus vite que l'apprentissage standard et sans aucun plateau de performance au tout début de l'apprentissage ce qui lui permet même de rattraper très rapidement le retard initial de performance avec l'approche D&C. Cette architecture associée à la fonction de coût de proximité angulaire permettent de réaliser un apprentissage équilibré sur l'ensemble des classes à discriminer qui se révèle très efficace autant en matière de performances de classification qu'en matière de rapidité d'apprentissage.

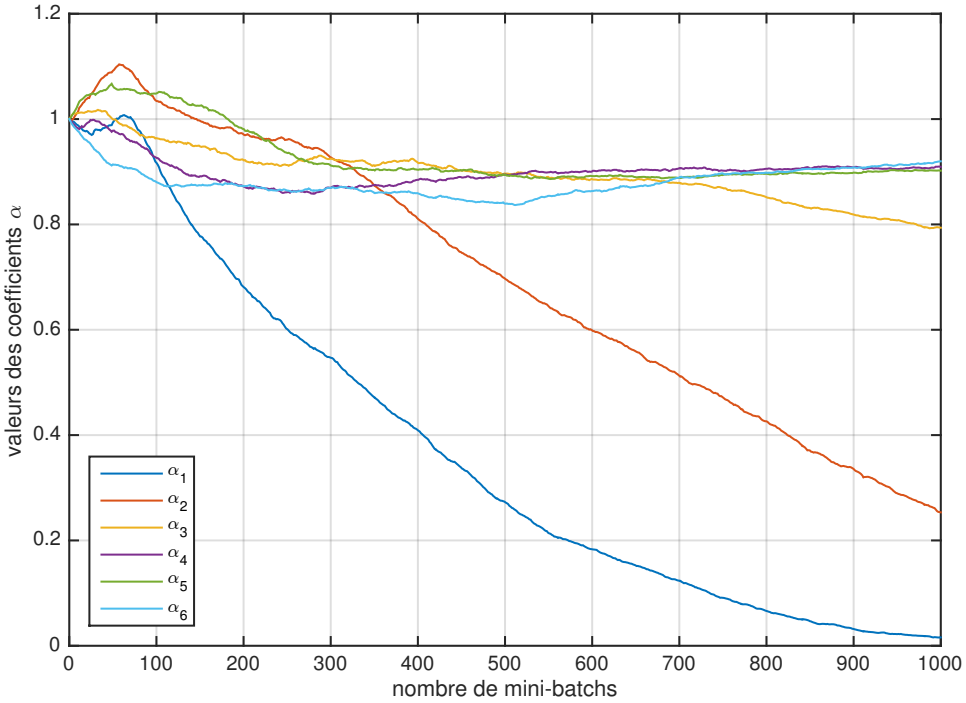


FIGURE 5.3 – Évolution des coefficients α_i durant l'entraînement d'une projection vectorielle par CG-LSTM avec six couches sur le jeu d'apprentissage de l'évaluation LRE-07.

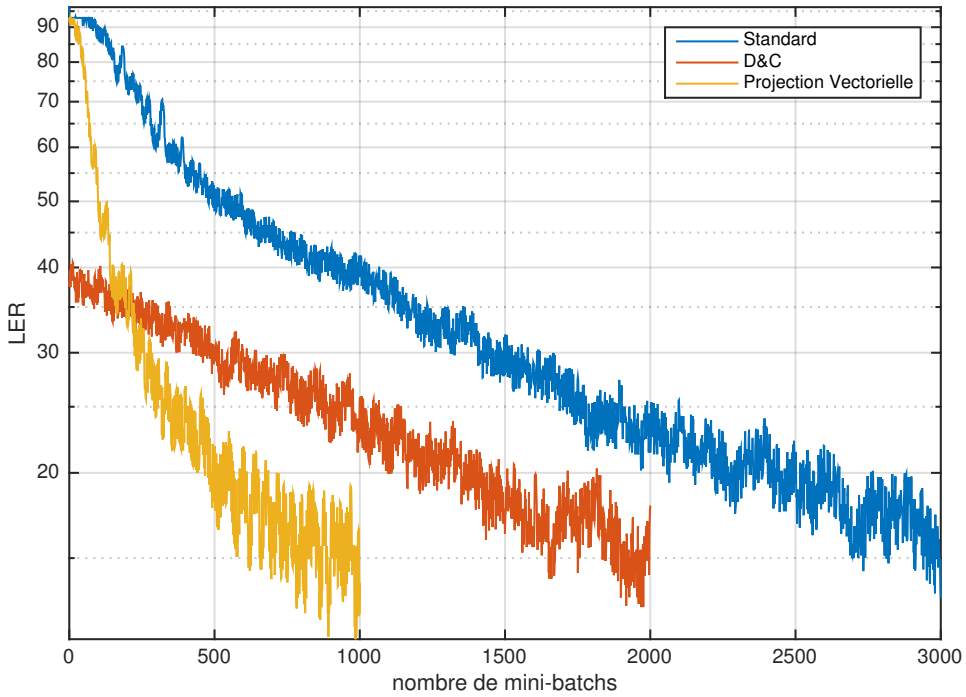


FIGURE 5.4 – Comparaison des durées d'apprentissage d'un système de LID basé sur des CG-LSTM lorsque l'on utilise différentes méthodes d'apprentissage.

5.3.3 Résultats pour l'évaluation LRE-15

Nous avons également vérifié la pertinence de notre approche sur les données de l'évaluation LRE-15. Compte tenu des problèmes rencontrés sur les jeux de données officiels, nous avons utilisé les jeux d'apprentissage et de test de notre étude post-évaluation (cf. sections 1.3.2 et 4.2.3). Les résultats de cette expérimentation post-évaluation sont donnés dans le tableau Tab. 5.5 et comme pour l'évaluation LRE-07 le système par projection vectorielle permet d'améliorer notablement les performances avec un gain relatif de plus de 9% sur le LER par rapport à ce que l'on obtient avec la méthode D&C. Ce gain est également observé lorsque l'on combine ce système avec les systèmes *i-vector* et phonotactique, ce qui fait de l'architecture de projection vectorielle par CG-LSTM entraînée à l'aide de la fonction de coût de proximité angulaire le meilleur système d'IdL basé sur des RNN que nous avons pu obtenir durant les travaux de cette thèse.

Tableau 5.5 – Résultats obtenus sur le jeu de données de test post-évaluation contenant 90% du jeu de test officiel de l'évaluation LRE-15 pour chacun des trois systèmes de LID pris individuellement et pour les combinaisons de ces systèmes.

System	*LER*	*EER*	*CAVG*
PHO	23.5	10.1	15.1
IVC	26.6	10.4	17.4
CG-LSTM	30.9	13.4	20.8
CG-LSTM (D&C)	22.8	8.4	14.6
CG-LSTM (PV)	20.7	7.8	13.3
PHO + IVC + CG-LSTM (D&C)	15.5	5.4	9.5
PHO + IVC + CG-LSTM (PV)	15.0	5.2	9.4

Comme nous l'avons fait sur les données de l'évaluation LRE-07, nous pouvons analyser les directions obtenues en sorties de notre système de projection vectorielle et visualiser la proximité des langues/dialectes entre eux et les difficultés d'identification. Des représentations dans le plan des séquences audio du jeu de test post-évaluation LRE-15 sont visibles sur les figures Fig. 5.5 et Fig. 5.6. Ces représentations en deux dimensions sont obtenues à l'aide de l'algorithme tSNE [131] directement à partir des vecteurs unitaires en sortie du RNN. Sur la première figure les points sont colorés par groupes de langues tels que définis par le NIST, sur la seconde on peut distinguer la totalité des vingt dialectes de l'évaluation.

Sur la figure Fig. 5.5 on peut observer que la notion de groupe de langue se retrouve dans la localisation dans le plan à l'exception de l'anglais indien qui se retrouve plus proche du chinois et de l'arabe que des deux autres accents anglais. On peut également noter que dans le groupe ibérique, le portugais brésilien est relativement éloigné des trois dialectes espagnols se qui s'explique aisément par le fait que ce n'est pas la même langue.

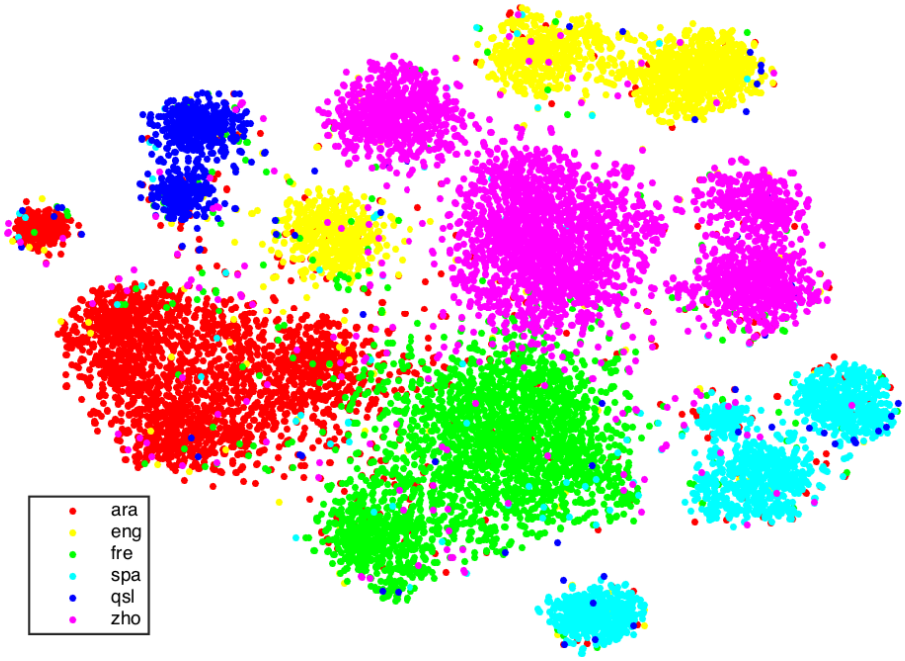


FIGURE 5.5 – Représentation des clusters de langues dans le plan obtenue à l'aide de l'algorithme tSNE à partir des projections vectorielles du jeu de test de post-évaluation de LRE-15.

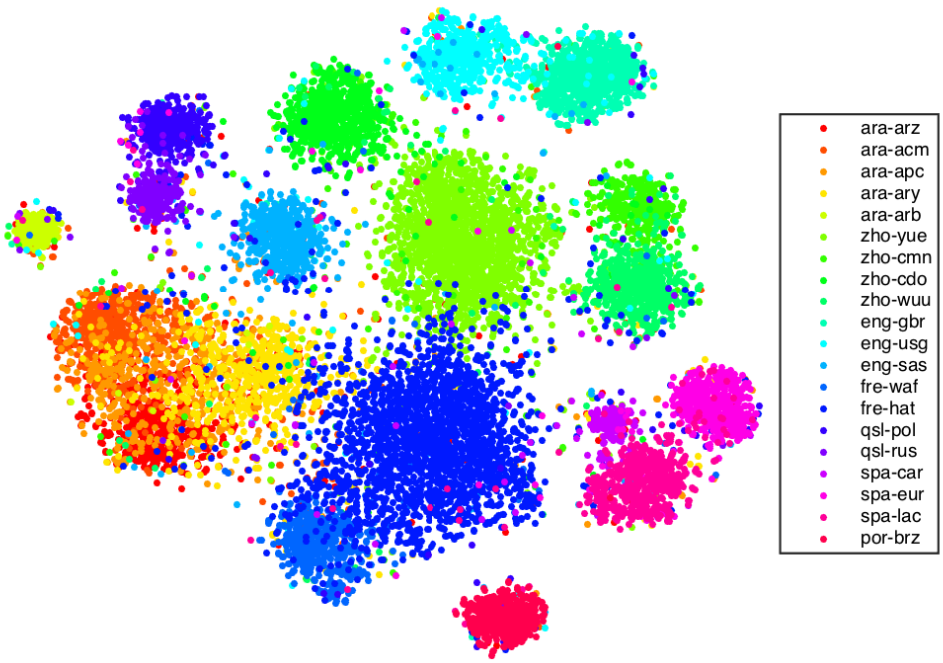


FIGURE 5.6 – Représentation des dialectes de langues dans le plan obtenue à l'aide de l'algorithme tSNE à partir des projections vectorielles du jeu de test de post-évaluation de LRE-15.

Tableau 5.6 – Cavg obtenus avec le système de projection vectorielle pour chacun des six groupes de langues sur les données du jeu de données de test post-évaluation contenant 90% du jeu de test officiel de l'évaluation LRE-15.

Cavg	*ara*	*zho*	*eng*	*fre*	*qsl*	*spa*
CG-LSTM (PV)	0.23	0.11	0.08	0.14	0.09	0.14

Sur la figure Fig. 5.6 on note surtout la grande difficulté de séparer dans le plan les dialectes arabes excepté l'arabe standard. On retrouve ainsi ce que l'on peut voir au travers du Cavg par groupe de langue donné dans le tableau Tab. 5.6 avec un Cavg pour le groupe arabe qui est de loin le plus élevé.

5.4 Conclusion

Dans ce chapitre nous avons proposé une architecture neuronale qui permet de projeter une séquence d'entrée de longueur variable sur un unique vecteur de taille fixe dont la direction est représentative de la classe. Conjointement, nous avons proposé une fonction de coût dite de proximité angulaire permettant d'entraîner un tel RNN pour des tâches multi-classe. Cette fonction de coût permet de minimiser l'écart angulaire entre deux projections vectorielles issues de séquences de la même classe tout en maximisant l'écart angulaire entre deux projections vectorielles issues de séquences de classes différentes.

Nous avons évalué la pertinence de cette approche sur la tâche d'identification de la langue dans le cadre des évaluations LRE-07 et LRE-15 du NIST. Dans les deux cas, cette méthode de projection vectorielle par RNN basée sur des CG-LSTM améliore très significativement les performances par rapport à une approche plus standard telle que celles comparées dans le chapitre précédent. En effet, on obtient ainsi un gain d'au moins 20 % relatifs sur le LER obtenu sur les deux évaluations LRE-07 et LRE-15 avec l'apprentissage classique et de près de 10 % relatifs par rapport à la méthode D&C.

De plus, ce gain de performance s'accompagne d'une importante réduction de la durée d'apprentissage puisque l'on entraîne un système d'IdL par projection vectorielle en seulement la moitié du temps nécessaire avec l'apprentissage D&C présenté dans le chapitre 4.

Chapitre 6

Outils d'optimisation et bonnes pratiques

Nous présentons dans ce chapitre les outils et les bonnes pratiques que nous utilisons pour entraîner nos différents systèmes de traitement de la parole basés sur des ANN. En effet, les ANN sont de puissants outils pour le traitement automatique de la parole mais peuvent cependant s'avérer difficiles à optimiser.

Les algorithmes et les bonnes pratiques présentés ici sont communs à tous les travaux de cette thèse et ne sont pas indispensables pour présenter les innovations décrites dans les chapitres précédents. Ils peuvent être précieux et éviter un certain nombre d'écueils à ceux qui souhaiteraient reproduire les expériences présentées dans cette thèse.

Nous détaillons d'abord les outils permettant d'estimer les paramètres des modèles neuronaux que nous avons utilisés lors de nos diverses expérimentations. Puis, nous présentons toutes les bonnes pratiques, telles que la méthode d'initialisation des paramètres d'un ANN ou l'augmentation de données, qui nous ont permis d'entraîner avec succès des ANN sur des tâches de détection de la parole et de reconnaissance de la langue.

6.1 Outils d'optimisation

Cette section est dédiée aux différentes approches que nous avons utilisées et/ou combinées pour estimer les paramètres optimaux pour les modèles neuronaux.

6.1.1 Descente de gradient

La première d'entre elles est certainement la plus classique : il s'agit de la technique itérative dite de la descente du gradient. Elle n'est utilisable que pour des modèles différentiables tels que les différents réseaux de neurones décrits dans les chapitres 1.1 et 2 lorsqu'on cherche à minimiser une fonction de coût $\mathcal{C}$ elle-même différentiable. Ainsi, si l'on appelle $\boldsymbol{w}_n$ le vecteur de réels contenant tous les paramètres de notre modèle à l'itération n du processus d'optimisation et qu'on est capable de calculer le gradient du coût $\mathcal{C}$ par rapport à $\boldsymbol{w}_n$, l'algorithme le plus simple de descente de gradient consiste à modifier chacun des paramètres dans la direction opposée au gradient. On

obtient le nouveau vecteur de paramètres $\boldsymbol{w}_{n+1}$

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \lambda \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_n) \tag{6.1}$$

où $\lambda \ll 1$ est le taux d'apprentissage permettant d'ajuster la modification des poids.

Cette modification peut-être réalisée après chaque paire d'apprentissage (entrée, cible) - on parle alors d'apprentissage séquentiel - ou de façon groupée après un certain nombre de paires - on parle alors d'apprentissage par paquet (*batch-learning* en anglais).

Le problème de cet algorithme, en particulier dans le cas où une modification des paramètres est effectuée après chaque exemple ou un petit groupe d'exemples, est qu'il est très susceptible de se retrouver coincé dans un minimum local de la fonction de coût. Afin de contrer ce phénomène il est très utile de rajouter un terme de moment (cf. [132] et [133]) ou d'utiliser la méthode Nesterov's Accelerated Gradient (NAG) ([134]). Comme cela est décrit par I. Sutskever dans [135], ces deux méthodes sont proches et peuvent s'écrire de la façon suivante :

— Ajout d'un moment $\boldsymbol{v}$:

$$\boldsymbol{v}_n = \mu \boldsymbol{v}_{n-1} - \lambda \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_{n-1}) \tag{6.2}$$

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \boldsymbol{v}_n \tag{6.3}$$

où $\mu \in [0, 1]$ est le coefficient de moment.

— Méthode NAG :

$$\boldsymbol{v}_n = \mu \boldsymbol{v}_{n-1} - \lambda \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_{n-1} + \mu \boldsymbol{v}_{n-1}) \tag{6.4}$$

$$\boldsymbol{w}_n = \boldsymbol{w}_{n-1} + \boldsymbol{v}_n \tag{6.5}$$

De nombreux autres algorithmes plus sophistiqués ont été développés tels que RPROP et ses variations ([136] et [137]), L-BFGS ([138]) et plus récemment RMSPROP ([139]), Adam ([140]), SFO ([141]) et SMORMS3 ([142]). Ces méthodes sont en général bien plus performantes autant en rapidité de convergence qu'en matière de performance finale du modèle à optimiser, y compris pour les différents types de RNN auxquels nous nous intéressons dans cette thèse.

Parmi ces algorithmes, SMORMS3 s'est révélé être le meilleur algorithme pour optimiser nos différents systèmes à base de LSTM. Cet algorithme inspiré de RMSPROP et des travaux de T. Shaul et Y. LeCun ([143]) estime à quel point le gradient de la fonction de coût $\mathcal{C}$ est bruité et à partir de cette estimation détermine le coefficient d'apprentissage le plus adapté pour chacun des paramètres. Les détails de l'algorithme sont décrits dans Alg. 3.

L'impact du choix de l'algorithme de descente de gradient est présenté sur les figures Fig 6.1a et Fig 6.1b. Nous détaillons sur ces deux figures l'évolution du coût sur le jeu de validation (voir §6.2.1) au fur et à mesure que l'apprentissage progresse pour une tâche de segmentation parole/non-parole et une tâche d'identification de la langue. Pour chacune de ces deux tâches, nous utilisons un système basé sur des CG-LSTM ayant des paramètres initiaux tirés aléatoirement (voir §6.2.3) mais identiques à la première itération de chacun des algorithmes d'apprentissage.

On voit clairement que le choix de l'algorithme de descente de gradient peut s'avérer crucial. En effet, dans les deux cas présentés ici on obtient avec SMORMS3 une convergence rapide vers un coût plus faible qu'avec n'importe quel autre algorithme dont certains convergent vers un minimum local en ayant du mal à s'en extirper (RPROP et SFO). C'est pourquoi pour cette thèse nous avons utilisé SMORMS3 dans toutes les expérimentations où nous avons utilisé la rétro-propagation du gradient pour entraîner un ANN.

Il est important de noter que si SMORMS3 est le meilleur algorithme de descente de gradient que nous avons testé il ne permet en aucun cas d'atteindre l'optimum global avec certitude. C'est d'ailleurs la raison pour laquelle nous proposons dans cette thèse de l'associer à d'autres méthodes d'optimisation pour les tâches de détection de la parole (cf. Chapitre 3) ou de modifier la structure des RNN pour atteindre de meilleurs optimum locaux pour les tâches multi-classes (cf. Chapitres 4 et 5).

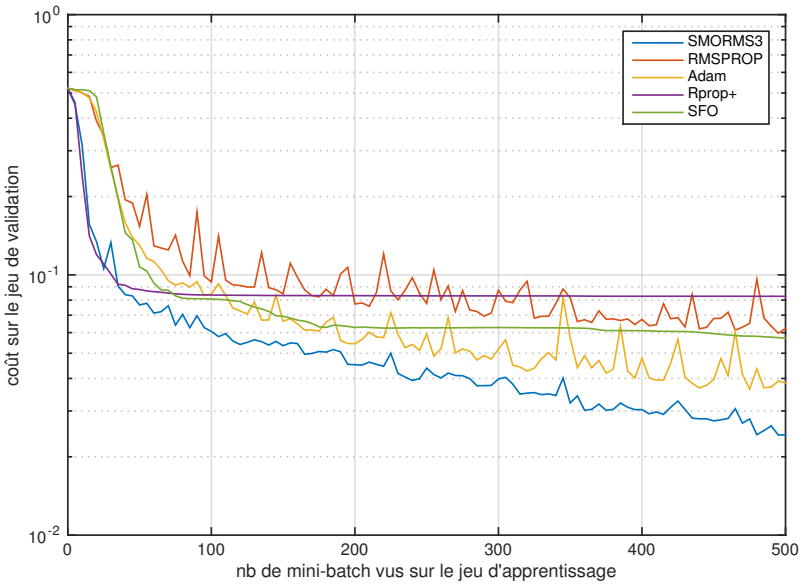
Algorithm 3 Détails de l'algorithme SMORMS3.
Les multiplications, divisions et la fonction "min" sont effectuées terme à terme. Et le terme $\epsilon = 1e-12$ n'est là que pour éviter les divisions par 0.

Initialisation aléatoire du vecteur de paramètres $\boldsymbol{w}_0$ de l'ANN à optimiser.
Initialisation des variables internes de l'algorithme sous la forme de vecteurs de même dimension que $\boldsymbol{w}_0$:
$\boldsymbol{\delta} = \mathbf{1}$ et $\boldsymbol{g} = \boldsymbol{g}_{sq} = \mathbf{0}$
$n = 0$
while critère d'arrêt == faux **do**
En utilisant l'algorithme de rétro-propagation des erreurs adapté à notre ANN (cf. chapitre 1.1), on détermine $\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_n)$:

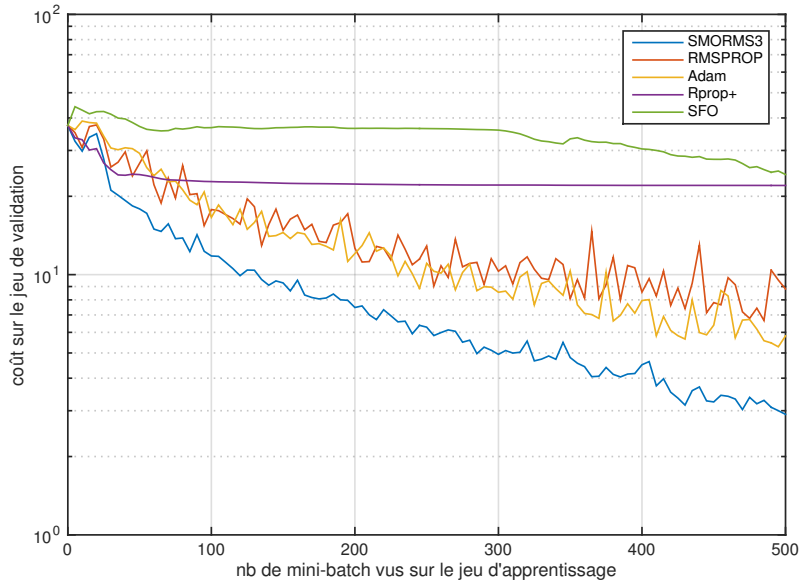
Mise à jour des variables internes de l'algorithme :
$\boldsymbol{r} = 1/(\boldsymbol{\delta} + 1)$
$\boldsymbol{g} = (1 - \boldsymbol{r}) \cdot \boldsymbol{g} + \boldsymbol{r} \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_n)$
$\boldsymbol{g}_{sq} = (1 - \boldsymbol{r}) \cdot \boldsymbol{g}_{sq} + \boldsymbol{r} \cdot (\frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_n))^2$
$\boldsymbol{\delta} = 1 + \boldsymbol{\delta} \cdot (1 - \boldsymbol{g}^2/(\boldsymbol{g}_{sq} + \epsilon))$

On détermine le vecteur $\boldsymbol{\lambda}$ des coefficients d'apprentissage :
$\boldsymbol{\lambda} = \min(0.01, \boldsymbol{g}^2/(\boldsymbol{g}_{sq} + \epsilon))$

Modification des paramètres de l'ANN :
$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \boldsymbol{\lambda} \cdot \frac{\partial \mathcal{C}}{\partial \boldsymbol{w}}(\boldsymbol{w}_n)/(\sqrt{\boldsymbol{g}_{sq}} + \epsilon)$
$n = n + 1$
end while



(a) Évolution du coût de segmentation parole/non-parole sur le jeu de validation au fur et à mesure que l'apprentissage progresse. Le coût est calculé ici sur les données de l'évaluation OpenSAD'15 organisée par le NIST (voir la section §3.3.3 pour plus de détails sur cette évaluation).



(b) Évolution du coût de classification sur le jeu de validation au fur et à mesure que l'apprentissage progresse. Le coût est calculé ici sur les données de l'évaluation LRE07 organisée par le NIST (voir la section §4.2 pour plus de détails sur cette évaluation).

FIGURE 6.1 – Performance de plusieurs algorithmes de descente de gradient pour optimiser un système à base de CG-LSTM sur deux tâches différentes.

6.1.2 Optimisation heuristique

L'intérêt des méthodes d'optimisation heuristiques telles que les algorithmes génétiques pour minimiser des fonctions de coût non-dérivables et ce même pour un grand nombre de paramètres $1 < n < 10^5$ (voir par exemple [144] où un algorithme basé sur un MLP est optimisé pour guider avec succès un atterrisseur martien). Depuis lors, de nombreuses méthodes similaires mais plus efficaces encore ont vu le jour. L'algorithme QPSO introduit par Sun en 2004 [145] est l'un d'eux, c'est cet algorithme que nous présentons ici et que nous utilisons dans cette thèse.

QPSO est une variante de l'algorithme Particle Swarm Optimization (PSO) qui fut lui-même introduit par Kennedy et Eberhart [146] et amélioré par Clerc [147]. PSO est une technique d'optimisation par agents qui s'inspire essentiellement du comportement social dans les volées d'oiseaux ou les bancs de poissons. Ainsi, le comportement d'un individu qui explore un espace de recherche en quête du maximum de ressources va être déterminé par sa trajectoire courante (position, vitesse), sa mémoire personnelle (position du maximum de ressources trouvée par l'individu jusque là) mais aussi par la connaissance collective de l'espace de recherche par la volée dans son ensemble (position du maximum global de ressources découverte par tous les individus). L'algorithme PSO modifie à chaque itération l'accélération des individus en tenant compte de ces trois composantes pour optimiser la recherche du maximum de ressources (ou du minimum de la fonction de coût dans notre cas).

Après avoir analysé les trajectoires produites par PSO, Sun et al. se sont inspirés de la mécanique quantique pour remplacer la cinématique des individus dans PSO par une stratégie de tirages aléatoires basée sur un puits de potentiel quantique autour des meilleures positions déjà trouvées. Cette modification fait que, bien que PSO soit comparable en performance aux algorithmes génétiques, QPSO les surpasse tous les deux sur des tâches d'optimisation difficiles (cf. [148]). C'est pourquoi dans cette thèse nous utilisons QPSO pour optimiser les systèmes non-différentiables ayant par exemple une paramétrisation discrète et/ou des fonctions de coût non-dérivables telles que le WER.

Dans la pratique, chaque jeu de d paramètres à optimiser pour un système donné est encodé sous la forme d'un vecteur $\boldsymbol{p} \in \mathbb{R}^d$. QPSO est un processus itératif initié à partir d'une population de n candidats $\boldsymbol{p}_j(0)$, $j \in [\![1, n]\!]$ tirés aléatoirement et de façon uniforme dans tout l'espace de recherche. Ensuite, à chaque itération t, QPSO commence par évaluer la fonction de coût pour chacun des candidats actuels $\mathcal{C}(\boldsymbol{p}_j(t))$ pour $j \in [\![1, n]\!]$ puis propose de nouveaux candidats $\boldsymbol{p}_j(t+1)$, $j \in [\![1, n]\!]$ basés sur :

— la meilleure position trouvée par chacun des candidats depuis le début de l'optimisation :

$$\boldsymbol{m}_j = \boldsymbol{p}_j \left(\underset{s \in [\![1,t]\!]}{\operatorname{argmin}} \mathcal{C}(\boldsymbol{p}_j(s)) \right) \tag{6.6}$$

pour $j \in [\![1, n]\!]$

— et la meilleure position trouvée par la volée dans son ensemble depuis le début de l'optimisation :

$$\boldsymbol{g} = \boldsymbol{p}_k, \quad k = \underset{j \in [\![1,n]\!]}{\operatorname{argmin}} \mathcal{C}(\boldsymbol{m}_j) \tag{6.7}$$

Les détails de l'algorithme QPSO décrits dans [148] sont rappelés dans Alg. 4.

Algorithm 4 QPSO

$t = 0$
Initialisation aléatoire de la population $\boldsymbol{p}_j(t)$ pour $j \in [\![1, n]\!]$
for $j = 1..n$ **do**
  Evaluation de la fonction de coût $\mathcal{C}(\boldsymbol{p}_j(t))$
  $\boldsymbol{m}_j = \boldsymbol{p}_j(t)$
  if $\mathcal{C}(\boldsymbol{m}_j) < \mathcal{C}(\boldsymbol{g})$ **then**
    $\boldsymbol{g} = \boldsymbol{m}_j$
  end if
end for
$t = 1$
while stopping criterion = false **do**
  for $j = 1..n$ **do**
    for $i = 1..d$ **do**
      On tire aléatoirement ϕ, u et α à partir d'une distribution uniforme sur $]0, 1]$
      $y_{i,j} = \phi_{i,j} \cdot m_{i,j} + (1 - \phi_{i,j}) \cdot g_i$
      if $\alpha_{i,j} > 0.5$ **then**
        $p_{i,j}(t) = y_{i,j} + |p_{i,j}(t-1) - m_{i,j}| \ln(1/u_{i,j})$
      else
        $p_{i,j}(t) = y_{i,j} - |p_{i,j}(t-1) - m_{i,j}| \ln(1/u_{i,j})$
      end if
    end for
    Evaluation de la fonction de coût $\mathcal{C}(\boldsymbol{p}_j(t))$
    if $\mathcal{C}(\boldsymbol{p}_j(t)) < \mathcal{C}(\boldsymbol{m}_j)$ **then**
      $\boldsymbol{m}_j = \boldsymbol{p}_j(t)$
      if $\mathcal{C}(\boldsymbol{m}_j) < \mathcal{C}(\boldsymbol{g})$ **then**
        $\boldsymbol{g} = \boldsymbol{m}_j$
      end if
    end if
  end for
  $t = t + 1$
end while

6.2 Bonnes pratiques

6.2.1 Généralisation

Bien que le coût $\mathcal{C}$ que l'on minimise lors d'un apprentissage soit par essence défini par le jeu de données d'entraînement, le but réel de l'apprentissage est d'optimiser la performance sur un jeu de données de test auquel le modèle n'a pas été préalablement confronté. S'assurer que la bonne performance d'un système sur le jeu d'apprentissage entraîne une performance similaire sur le jeu de test est primordial, et on parle alors d'une bonne généralisation lors de l'apprentissage. Cette capacité à généraliser est fondamentale dans le domaine de l'apprentissage machine (cf. [149]). De manière générale, plus le jeu d'apprentissage est grand et varié meilleure est la généralisation. Mais comme on ne peut pas toujours (souvent) augmenter indéfiniment la taille du jeu d'apprentissage de nombreuses méthodes dites de *régularisation* ont vu le jour pour améliorer la généralisation des systèmes à base d'ANN : arrêt prématuré de l'apprentissage sur un critère de performance obtenu sur un jeu de validation, ajout de bruit sur les entrées, ajout de bruit sur les poids de l'ANN, régularisations L1 et L2 sur les poids du réseau, *dropout*, etc.

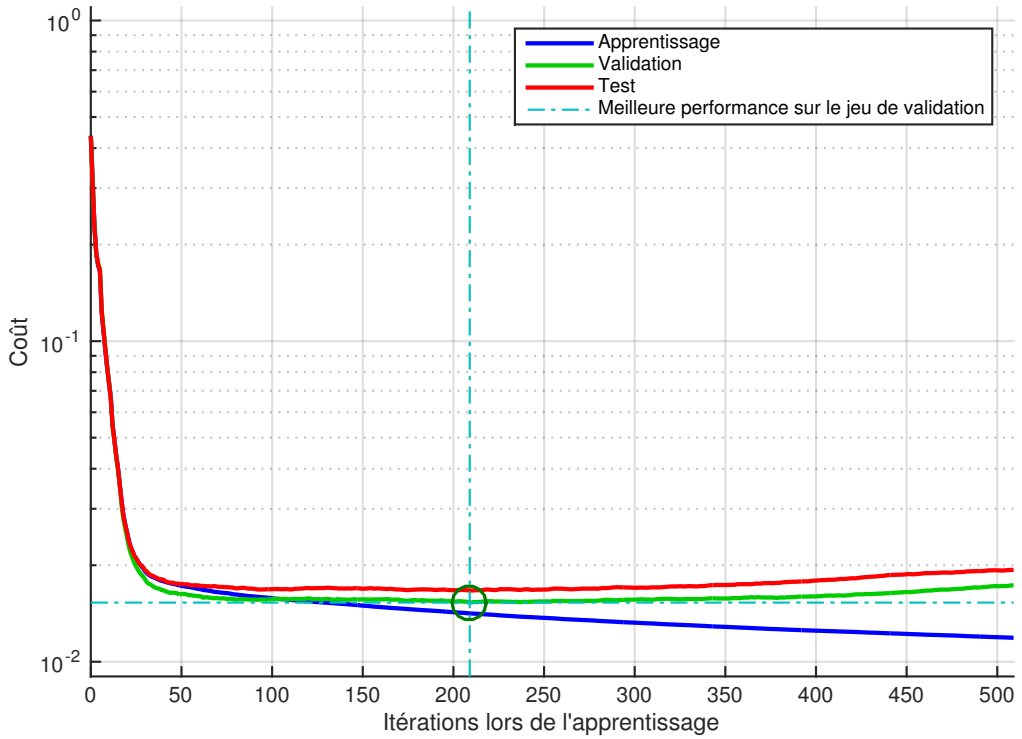


FIGURE 6.2 – Sur-apprentissage et arrêt prématuré. On observe d'abord une diminution du coût sur les 3 jeux de données (apprentissage, validation et test) puis au bout d'un certain temps on observe que le coût se met à augmenter sur les jeux de validation et de test alors qu'il continue de descendre sur le jeu d'apprentissage. On arrête alors l'apprentissage et on récupère les paramètres du modèle qui donnaient la meilleure performance sur le jeu de validation (indiquée par la ligne en pointillé). En faisant ce choix, on espère obtenir la meilleure performance possible sur le jeu de test ou tout autre jeu de données sur lequel le système ne s'est pas entraîné.

Pour cette thèse nous avons testé l'ensemble des techniques citées et à l'exception de l'arrêt prématuré elles n'ont jamais permis d'améliorer les résultats sur le jeu de test. Ceci est sans doute dû à la petite taille des modèles que nous avons entraînés et pour laquelle les risques de sur-apprentissage sont faibles.

Nous n'avons donc pas utilisé d'autre régularisateur que l'arrêt prématuré pour lequel une partie du jeu d'apprentissage est mise de côté pour être utilisée comme jeu de validation. Durant l'apprentissage, on peut alors vérifier que lorsqu'on améliore les performances sur le jeu d'apprentissage on améliore également les performances sur le jeu de validation. Si ce n'est plus le cas pendant un certain nombre d'itérations, on arrête l'apprentissage et on récupère les paramètres du modèle qui donnaient la meilleure performance sur le jeu de validation. En faisant ce choix, on espère obtenir la meilleure performance possible sur le jeu de test ou tout autre jeu de données sur lesquelles le système ne s'est pas entraîné.

Lorsque le modèle est assez gros (en nombre de paramètres) on observe lors de l'apprentissage d'abord une diminution du coût sur les 3 jeux de données (apprentissage, validation et test) puis au bout d'un certain temps on observe que le coût se met à augmenter sur les jeux de validation et de test alors qu'il continue de descendre sur le jeu d'apprentissage (cf. Fig. 6.2). On appelle ce phénomène *sur-apprentissage*.

6.2.2 Choix de représentation des entrées

Le choix d'une bonne représentation des données d'entrée est primordiale pour toute tâche d'apprentissage machine. Il arrive en effet que ce choix se révèle plus important pour la performance finale que l'algorithme utilisé. Les ANN peuvent toutefois être relativement robustes à ce choix de représentation des entrées : plusieurs expériences sur des séquences audio (voir par exemple l'expérience chapitre 9 de [1]) montrent que l'on peut atteindre le même niveau de performance en utilisant trois représentations très différentes du signal audio : signal temporel, spectrogramme, MFCC. Nous avons nous même pu l'observer sur une tâche de segmentation parole/non-parole pour laquelle nous avons obtenu des performances similaires pour ces trois mêmes représentations d'un signal audio (cf. 3.3.1). Les temps de traitement étaient en revanche très différents et c'est là que le choix de la représentation prend de l'importance.

Une bonne représentation des entrées se doit de contenir toute l'information pertinente et nécessaire à la réalisation de la tâche mais doit également être la plus compacte possible et limiter les redondances de manière à minimiser le nombre de calculs. Le choix d'une représentation des entrées très complète mais également de très grande dimension engendre automatiquement un ANN avec un très grand nombre de paramètres dans sa couche d'entrée et donc un risque plus grand de mauvaise généralisation du problème en plus du nombre de calculs élevé.

Quelle que soit la représentation des entrées choisie, il y a une procédure qu'il est préférable de suivre : normalisation des vecteurs d'entrée présentés à un ANN afin que chaque composante des vecteurs d'entrée ait une moyenne de 0 et un écart-type de 1 sur l'ensemble du jeu d'apprentissage. Pour ce faire, on commence par calculer la moyenne μ_i de chacune des composantes x_i des vecteurs d'entrée ainsi que leur écart-type σ_i sur l'ensemble du jeu d'apprentissage. Ensuite, pour chaque vecteur on normalise chacune

de ces composantes de la façon suivante :

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sigma_i} \tag{6.8}$$

Cette manipulation n'altère pas l'information contenue dans le jeu d'apprentissage mais améliore l'apprentissage lui-même et souvent la performance finale en recentrant les valeurs mises en entrée de la fonction d'activation de la première couche d'un ANN sur un intervalle intéressant (cf. [150]). Ceci est particulièrement vrai lorsqu'on utilise la fonction logistique ou la tangente hyperbolique comme fonction d'activation puisque ces 2 fonctions sont centrées sur zéro et saturent en dehors de l'intervalle $[-5, 5]$. Dans tous les travaux effectués pour cette thèse, cette procédure de normalisation des données d'entrée à toujours été utilisée.

On note que lorsque l'on normalise les vecteurs d'entrée du jeu d'apprentissage avec la méthode qui vient d'être explicitée, il faut bien entendu reproduire la même procédure sur les jeux de validation et de test en utilisant les moyennes et écart-types calculés sur le jeu d'apprentissage.

6.2.3 Initialisation des poids d'un ANN

Pour toute recherche itérative de paramétrisation optimale, la qualité du processus peut être grandement impactée par le choix initial des paramètres. Les ANN n'échappent pas à cette règle voire y sont particulièrement sensibles. L'initialisation des paramètres d'un ANN peut s'avérer cruciale pour la performance finale du système en particulier pour des réseaux profonds ou récurrents utilisant des fonctions d'activation comme la fonction logistique ou la tangente hyperbolique et qui sont touchés par les problèmes dits de gradient évanescent ou divergent (*vanishing and exploding gradient problem*) lors de l'apprentissage.

Pour minimiser les chances que ce type de problèmes impacte l'apprentissage, il est préférable d'initialiser les paramètres d'une couche d'ANN en utilisant la règle de Glorot [151] qui permet d'assurer que lorsque l'information se propage dans toutes les couches (ou à travers les liens récurrents pour un RNN), les entrées des fonctions d'activation sont toujours dans un intervalle pertinent ($[-5, 5]$ par exemple). En effet, si les paramètres sont trop petits on finit par avoir en entrée des fonctions d'activation les plus profondes dans le réseau (ou au bout d'un certain temps pour un RNN), un signal proche de zéro pour lequel la fonction d'activation (par exemple, tanh) est quasi-linéaire et on perd alors l'intérêt d'avoir de nombreuses couches (ou de la récurrence). On obtient un problème similaire avec des valeurs de paramètres trop grandes pour lesquelles le signal se retrouve dans les régions où la fonction d'activation est très proche de son asymptote et où on obtient un comportement binaire avec des gradients quasi-nuls ce qui limite fortement la capacité d'apprentissage. La stratégie proposée par Glorot pour une couche L_j est la suivante : les paramètres $\boldsymbol{W}_j$ sont initialisés en tirant uniformément dans l'intervalle

$$\left[-\sqrt{\frac{6}{n_{j-1}+n_j}}, \sqrt{\frac{6}{n_{j-1}+n_j}}\right] \tag{6.9}$$

Il est important de noter que l'utilisation des fonctions d'activation telles que ReLU qui offrent une invariance d'échelle autour de 0 et qui ne saturent pas pour les grandes

valeurs permettent de s'affranchir d'une partie de ces problèmes, ce qui explique leurs succès dans les architectures profondes. Cependant, des méthodes d'initialisation inspirée de l'analyse de Glorot on récemment été proposées pour ce type de fonctions d'activation (voir [152]).

6.2.4 Mini-batch

Les méthodes décrites dans la section 6.1.1 telles que RMSPROP, Adam, SFO et SMORMS3 sont des méthodes de descente de gradient stochastiques qui sont devenues la norme pour les tâches d'apprentissage profond. Par construction, ces méthodes fonctionnent en décomposant le jeu d'apprentissage en petits sous-ensembles de quelques centaines d'exemples (souvent appelés *mini-batch*) que l'on traite dans un ordre aléatoire pour construire une estimation pertinente du gradient qui nous intéresse.

Il a été observé (voir [153], [154] et le chapitre 8 de [155]) que, de façon contre-intuitive, l'augmentation de la taille des *mini-batch* pénalise la capacité de généralisation du modèle qu'on entraîne. Il est donc préférable de travailler avec de petits *mini-batch* pour obtenir la meilleure performance sur le jeu de test.

Pour cette thèse, nous utilisons toujours des *mini-batch* de quelques centaines d'exemples tirés aléatoirement mais nous y ajoutons également une centaine de cas pires observés depuis le début de l'apprentissage. L'intérêt de ces cas pires est de forcer le modèle à travailler sur les frontières entre les différentes classes à discriminer plutôt que sur l'espace en entier. Nous avons toujours obtenu une meilleure généralisation et donc une meilleure performance sur le jeu de test en utilisant cette stratégie quelle que soit la tâche de classification que nous souhaitions traiter (voir section §5.3.2).

6.2.5 Segmentation et points de vue multiples

Dans la quasi-totalité des cas où l'on traite des séquences audio, le contexte nécessaire à un RNN pour réaliser une tâche de classification (détection de la parole, reconnaissance des phonèmes, reconnaissance de la langue, reconnaissance du locuteur, etc.) n'excède pas les quelques secondes autour du point d'intérêt. Il est alors préférable de découper les longues séquences en tronçons de quelques secondes (par exemple 5s) de manière à faciliter l'apprentissage et éviter que le RNN ne diverge et/ou sature sur des séquences plus longues qui peuvent facilement contenir des dizaines de milliers de pas de temps. Cela permet également d'éviter le cas où l'on aurait à utiliser le RNN sur une séquence de test d'une durée bien supérieure à toutes celles vues durant l'apprentissage et pour laquelle il serait difficile de prédire son comportement.

Afin d'éviter une perte de performance liée aux effets de bords en début et en fin de tronçon, nous transformons la séquence initiale en une série de sous-séquences qui se chevauchent d'au moins la moitié de manière à ce que chaque pas de temps soit couvert par au moins deux sous-séquences. Ensuite, pour déterminer la séquence de sortie nous reconstituons une séquence de la même taille que la séquence d'entrée en moyennant les contributions des sorties obtenues sur les sous-séquences (voir Fig. 6.3).

Cette stratégie augmente le nombre de calculs mais s'est toujours révélée bénéfique et c'est pourquoi nous l'avons utilisée dans l'ensemble des travaux réalisés pour cette thèse. Ainsi pour les tâches de segmentation décrites dans le chapitre 3, nous découpons

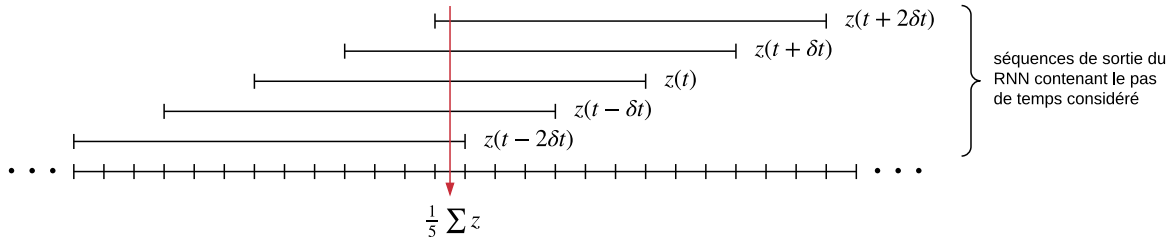


FIGURE 6.3 – Pour améliorer la robustesse d'un système basé sur un RNN bidirectionnel, on découpe la séquence d'entrée en sous-séquences de durée T qui se chevauchent de $T-\delta t > T/2$ que l'on met en entrée du RNN. Ensuite, la sortie finale du système à un pas de temps donné correspond à la moyenne des sorties obtenues sur les sous-séquences contenant le pas de temps considéré.

les séquences d'entrées en sous-séquences de 5s qui se chevauchent de 50% de manière à obtenir un comportement robuste du système de segmentation quelle que soit la durée de la séquence audio à segmenter. Pour la tâche d'identification de la langue étudiée dans les chapitres 4 et 5, nous avons pu observer qu'il était intéressant d'augmenter le chevauchement des sous-séquences (voir section 5.3.2). Dans tous les cas, ce découpage est réalisé de la même façon sur les jeux d'apprentissage et de validation mais également sur le jeu de test pour être cohérent.

6.2.6 Augmentation des données

Comme cela est décrit par Ko dans [156], on appelle augmentation de données toute méthode permettant d'augmenter artificiellement la quantité de données d'apprentissage et par voie de conséquence d'améliorer les performances, en particulier la robustesse, des modèles entraînés sur ces données. Plusieurs méthodes d'augmentation de données audio ont été proposées par le passé comme par exemple l'ajout de bruit au signal d'origine [157], l'augmentation ou la diminution de la fréquence d'échantillonnage pour modifier à la fois la vitesse de la parole mais aussi la hauteur de la voix [156], ou encore la méthode Vocal Tract Length Perturbation (VTLP) introduite par Jaitly et Hinton dans [158] et étendue par Cui dans [159].

Ce type d'approche est particulièrement adapté pour entraîner des systèmes de reconnaissance de la parole sur des langues peu dotées (ayant peu de données annotées disponibles) ou dans le cas d'une tâche de classification pour laquelle au moins une des classes n'est pas assez représentée dans le corpus d'apprentissage.

En ce qui concerne les travaux réalisés dans le cadre de cette thèse, pour les tâches de segmentation de la parole nous n'avons pas observé de gain significatif lié à l'utilisation de cette méthode. Cela s'explique par le fait que pour ces tâches les corpus d'apprentissage sur lesquels nous avons travaillé étaient amplement suffisants et représentatifs des jeux de test et validation. En revanche, pour les tâches d'identification de la langue, certaines langues étant très mal représentées dans le corpus d'apprentissage nous avons eu recours à la méthode d'augmentation de données proposée dans [156]. Ainsi pour chaque fichier audio on crée deux copies additionnelles en modifiant la fréquence d'échantillonnage respectivement à 90% et à 110% de l'original.

Par ailleurs, il est intéressant de noter que la transformation des séquences audio

d'origine en sous-séquences qui se chevauchent (cf. 6.2.5) est elle-même une forme d'augmentation de données puisque l'on présente aux RNN les mêmes données sous plusieurs points de vue différents (contextes passé et futur accessibles différents).

Conclusion

Pour cette thèse nous souhaitions étudier un type de RNN particulier appelé LSTM et leur utilisation pour des tâches de traitement automatique de la parole. Les deux tâches sur lesquelles nous présentons des résultats sont la détection de la parole et l'identification de langue parlée. Au cours de ces travaux, nous avons été amenés à proposer une modification du modèle LSTM que nous avons appelée Long Short-Term Memory with Coordinated Gates (CG-LSTM) en rajoutant des liens entre les portes logiques des cellules pour améliorer leur capacité de modélisation. Et pour chacune des deux tâches que nous désirions traiter nous avons proposé de nouvelles méthodes d'apprentissage dans le but d'améliorer les performances tout en essayant de diminuer le temps nécessaire à cet apprentissage.

Pour la détection de la parole, nous avons proposé un processus d'optimisation qui repose sur l'utilisation alternée de l'algorithme d'optimisation heuristique QPSO et de la méthode de descente de gradient SMORMS3 qui s'est révélé tout particulièrement efficace pour régler les milliers de paramètres des systèmes de SAD basés sur des CG-LSTM. Ce processus nous a permis d'optimiser un système de SAD de manière à minimiser le WER d'un système de reconnaissance de la parole grâce en particulier à une fonction de coût proche du WER spécifiquement développée pour cette tâche. Mais il nous a aussi permis d'optimiser des systèmes de segmentation pure (minimisation du FER) sur des corpus très différents : OpenSAD'15, REPERE, AMI et Game of Thrones.

Ces développements ont également été l'occasion de comparer différentes techniques de SAD. Tous nos tests montrent que le modèle CG-LSTM est plus performant que des réseaux de neurones de types MLP, RNN de Elman ou LSTM. Avec ce modèle, nous avons été classés troisième dans la campagne d'évaluation OpenSAD'15 organisée par le NIST avec un niveau de performance très proche du système classé second tout en ayant dix à cent fois moins de paramètres.

Dans le cas d'une tâche de classification multi-classe telle que l'identification de la langue, nous avons présenté une stratégie d'apprentissage s'inspirant de la maxime *diviser pour mieux régner*. Cette apprentissage que nous désignons par D&C (Divide and Conquer) divise la tâche multi-classe en sous-tâches de classification binaire que l'on résout avec de petits réseaux de neurones. On peut ensuite combiner ces petits réseaux pour initialiser un réseau multi-classe. Cette méthode nous a permis à la fois d'obtenir une meilleure identification de langue, mais également de diminuer le temps d'apprentissage. Nous avons appliqué cette méthode à deux tâches d'identification, celles de deux évaluations internationales LRE-07 et LRE-15 organisées par le NIST. Sur ces deux corpus la méthode D&C permet d'améliorer significativement les performances obtenues avec un système d'identification basé sur des CG-LSTM. Grâce

à cette méthode nous obtenons de meilleurs résultats que ceux d'un système *i-vector* sur les deux évaluations et meilleurs que ceux d'un système phonotactique sur la tâche difficile de distinction des langues proches de l'évaluation LRE-15. De plus, ce niveau de performance est atteint avec un nombre de paramètres inférieur d'au moins un ordre de grandeur par rapport aux deux systèmes de référence (phonotactique et *i-vector*).

Ensuite, toujours dans le cas d'une tâche multi-classe nous avons introduit une architecture neuronale permettant de projeter les séquences d'entrées sur un unique vecteur dont la direction est représentative de la classe. Conjointement, nous avons introduit la fonction de coût dite de *proximité angulaire* qui permet de minimiser l'écart angulaire entre deux projections vectorielles issues de séquences de la même classe tout en maximisant l'écart angulaire entre deux projections vectorielles issues de séquences de classes différentes. Nous avons confronté cette nouvelle approche aux mêmes tâches d'identification de la langue que la méthode D&C et nous avons pu améliorer significativement les performances tout en diminuant très fortement la durée d'apprentissage (30% de la durée d'apprentissage classique et 50% de la durée d'apprentissage avec la méthode D&C). Cette méthode est applicable à d'autres tâches de classification en traitement automatique de la parole (locuteurs, genres musicaux) ainsi qu'à d'autres problèmes.

Liste des publications

2015

Gelly, G. et J.-L. Gauvain. 2015, «Minimum word error training of rnn-based voice activity detection», *Interspeech, Dresden, 2015. 25, 64*

2016

Gelly, G., J. Gauvain, L. Lamel, A. Laurent, V. Le et A. Messaoudi. 2016a, «Language recognition for dialects and closely related languages», *Odyssey 2016, Bilbao, Spain.*

Bredin, H. et G. Gelly. 2016, «Improving speaker diarization of tv series using talking-face detection and clustering», *Proceedings of the 2016 ACM on Multimedia Conference, Amsterdam, p. 157–161.*

Gelly, G., J.-L. Gauvain, V. Le et A. Messaoudi. 2016b, «A divide-and-conquer approach for language identification based on recurrent neural networks», *Interspeech, San Francisco, 2016, p. 3231–3235. 35*

2017

Gelly, G. et J.-L. Gauvain. 2017, «Spoken language identification using LSTM-based angular proximity», *Interspeech, Stockolm, 2017.*

Wisniewksi, G., H. Bredin, G. Gelly, C. Barras. 2017, «Combining speaker turn embedding and incremental structure prediction for low-latency speaker diarization», *Interspeech, Stockolm, 2017.*

Gelly, G. et J.-L. Gauvain. 2017, «Recurrent Neural Networks for Speech Activity Detection», *IEEE, Transactions on Audio, Speech and Language Processing, soumis le 05.03.2017.*

Abstracts

Minimum word error training of rnn-based voice activity detection, Interspeech 2015

Voice Activity Detection (VAD) is critical in speech recognition systems as it can dramatically impact the recognition accuracy especially on noisy data. This paper presents a novel method which applies Minimum Word Error (MWE) training to a Long Short-Term Memory RNN to optimize Voice Activity Detection for speech recognition. Experiments compare speech recognition WERs using RNN VAD with other commonly used VAD methods for two corpora : the conversational Vietnamese corpus used in the NIST OpenKWS13 evaluation and a corpus of French telephone conversations. The proposed VAD method combining MWE training with RNN yields the best ASR results. This MWE training scheme appears to be particularly useful for low resource ASR tasks, as exemplified by the IARPA BABEL data.

Language Recognition for Dialects and Closely Related Languages, Odyssey 2016

This paper describes our development work to design a language recognition system that can discriminate closely related languages and dialects of the same language. The work was a joint effort by LIMSI and Vocapia Research in preparation for the NIST 2015 Language Recognition Evaluation (LRE). The language recognition system results from a fusion of four core classifiers : a phonotactic component using DNN acoustic models, two purely acoustic components using a RNN model and i-vector model, and a lexical component. Each component generates language posterior probabilities optimized to maximize the LID NCE, making their combination simple and robust. The motivation for using multiple components representing different speech knowledge is that some dialect distinctions may not be manifest at the acoustic level. We report experiments on the NIST LRE15 data and provide an analysis of the results and some post-evaluation contrasts. The 2015 LRE task focused on the identification of 20 languages clustered in 6 groups (Arabic, Chinese, English, French, Slavic and Iberic) of similar languages. Results are reported using the NIST Cavg metric which served as the primary metric for the OpenLRE15 evaluation. Results are also reported for the EER and the LER.

Improving Speaker Diarization of TV Series Using Talking-Face Detection and Clustering, ACM on Multimedia 2016

While successful on broadcast news, meetings or telephone conversation, state-of-the-art speaker diarization techniques tend to perform poorly on TV series or movies. In this paper, we propose to rely on state-of-the-art face clustering techniques to guide acoustic speaker diarization. Two approaches are tested and evaluated on the first season of Game Of Thrones TV series. The second (better) approach relies on a novel talking-face detection module based on bidirectional long short-term memory recurrent neural network. Both audio-visual approaches outperform the audio-only baseline. A

detailed study of the behavior of these approaches is also provided and paves the way to future improvements.

A Divide-and-Conquer Approach for Language Identification Based on Recurrent Neural Networks, Interspeech 2016

This paper describes the design of an acoustic language recognition system based on BLSTM that can discriminate closely related languages and dialects of the same language. We introduce a Divide-and-Conquer (D&C) method to quickly and successfully train an RNN-based multi- language classifier. Experiments compare this approach to the straightforward training of the same RNN, as well as to two widely used LID techniques : a phonotactic system using DNN acoustic models and an i-vector system. Results are reported on two different data sets : the 14 languages of NIST LRE07 and the 20 closely related languages and dialects of NIST OpenLRE15. In addition to reporting the NIST Cavg metric which served as the primary metric for the LRE07 and OpenLRE15 evaluations, the EER and LER are provided. When used with BLSTM, the D&C training scheme significantly outperformed the classical training method for multi-class RNNs. On the OpenLRE15 data set, this method also outperforms classical LID techniques and combines very well with a phonotactic system.

Spoken language identification using LSTM-based angular proximity, Interspeech 2017

This paper describes the design of an acoustic language identification (LID) system based on LSTMs that directly maps a sequence of acoustic features to a vector in a vector space where the angular proximity corresponds to a measure of language/dialect similarity. A specific architecture for the LSTM-based language vector extractor is introduced along with the angular proximity loss function to train it. This new LSTM-based LID system is quicker to train than a standard RNN topology using stacked layers trained with the cross-entropy loss function and obtains significantly lower language error rates. Experiments compare this approach to our previous developments on the subject, as well as to two widely used LID techniques : a phonotactic system using DNN acoustic models and an i-vector system. Results are reported on two different data sets : the 14 languages of NIST LRE07 and the 20 closely related languages and dialects of NIST LRE15. In addition to reporting the NIST Cavg metric which served as the primary metric for the LRE07 and LRE15 evaluations, the average LER is provided.

Combining speaker turn embedding and incremental structure prediction for low-latency speaker diarization, Interspeech 2017

Real-time speaker diarization has many potential applications, including public security, biometrics or forensics. It can also significantly speed up the indexing of increasingly large multimedia archives. In this paper, we address the issue of low-latency speaker diarization that consists in continuously detecting new or reoccurring speakers

within an audio stream, and determining when each speaker is active with a low latency (e.g. every second). This is in contrast with most existing approaches in speaker diarization that rely on multiple passes over the complete audio recording. The proposed approach combines speaker turn neural embeddings with an incremental structure prediction approach inspired by state-of-the-art Natural Language Processing models for Part-of-Speech tagging and dependency parsing. It can therefore leverage both information describing the utterance and the inherent temporal structure of interactions between speakers to learn, in supervised framework, to identify speakers. Experiments on the Etape broadcast news benchmark validate the approach.

Recurrent Neural Networks for Speech Activity Detection, IEEE Transactions on Audio, Speech and Language Processing, soumis le 05.03.2017

Speech activity detection (SAD) is an essential component of automatic speech recognition (ASR) systems impacting the overall system performance. This paper details how to optimize an SAD component based on recurrent neural networks (RNNs) with task-dependent training loss functions, investigating different NN models and optimization methods. Both Frame Error Rate and Word Error Rate related loss functions are considered as the latter is more appropriate for ASR systems. As a result of this study, we propose a novel coordinated-gate long short- term memory (CG-LSTM) model optimized with a combination of gradient descent (GD) and quantum-behaved particle swarm optimization (QPSO). QPSO being suited for non-differentiable optimization problems allows to efficiently tune the features extraction and the smoothing parameters as well as to find a very good initialization point for the CG-LSTM model. GD can then improve the behavior of the CG-LSTM model by tailoring its parameters. Results are reported following the conditions of two NIST evaluations for SAD and for keyword spotting (KWS). The proposed CG-LSTM model gives the best results compared to all other tested SAD methods and the proposed optimization method significantly improves the results compared to a gradient-descent training alone.

Liste des figures

Liste des tableaux

Liste des acronymes

Adam Adaptive moment estimation. 108, 116

AG Algorithmes Génétiques. 57, 58

AMI Augmented Multi-party Interaction. 30, 54, 74

ANN réseau de neurones artificiels (*Artificial Neural Network*). 6, 8, 23–25, 31, 33, 53–55, 59–61, 63–65, 72, 74, 77, 79, 80, 96, 97, 107, 109, 113–115, 125, 127

Babel programme de l'IARPA dont le but était de faire émerger des technologies de RAP qui puissent être rapidement appliquées à n'importe quelle langue parlée dans le monde. 27–29, 54, 64–66, 69, 70, 72, 74–76, 127

BiRNN réseau de neurones récurrent bidirectionnel (*Bidirectional Recurrent Neural Network*). 21, 22, 125

BPTT rétro-propagation dans le temps (*Back-Propagation Through Time*). 19, 47

Cavg métrique dite du coût de performance moyen. Elle est utilisée comme métrique principale pour les évaluations organisées par le NIST. 36, 81, 85, 86, 88, 89, 99, 106

CG-LSTM Long Short-Term Memory with Coordinated Gates. 2, 3, 5, 41, 50–56, 64–68, 70–76, 80–83, 85, 86, 88, 89, 91, 98–101, 103, 104, 106, 108, 110, 119, 125, 127

CMLLR Constrained Maximum Likelihood Linear Regression. 34

CMVN Cepstral Mean and Variance Normalization. 80

CrossCor critère de SAD basé sur l'auto-corrélation du signal. 24, 59

D&C méthode d'apprentissage d'une tâche d'apprentissage multi-classes inspirée de la maxime "diviser pour régner" (Divide and Conquer en anglais soit D&C). 3, 77, 79–81, 83, 85, 86, 88, 89, 91, 93, 96, 99, 100, 102, 104, 106, 119, 120, 125

DARPA Defense Advanced Research Projects Agency. 1, 23, 29, 72, 76

DCF Detection Cost Function. 29, 54, 56, 60, 72, 73, 76, 127

DER Diarization Error Rate. 29, 74

DNN réseau de neurones profond (*Deep Neural Network*). 2

DTW Dynamic Time Warping. 1

EER le taux d'égale erreur (ou Equal Error Rate) correspond au point de fonctionnement d'un système pour lequel le taux de fausses acceptations est égal au taux de faux rejets. 36, 81, 85, 99

EM Expectation–Maximization. L'algorithme espérance-maximisation (en anglais expectation-maximization algorithm), proposé par Dempster et al. en 1977, est un algorithme itératif qui permet de trouver les paramètres de vraisemblance maximum d'un modèle probabiliste lorsque ce dernier dépend de variables latentes non observables. 33

FER Frame Error Rate. 2, 27, 29, 54, 56, 57, 60, 61, 64–67, 71, 74–76, 119, 125, 127

FFNN réseau de neurones acyclique (*Feed-Forward Neural Network*). 7, 16

GMM modèle de mélanges gaussiens (ou Gaussian Mixture Model). 25, 32

GPU Graphics Processing Unit. 5

HF La bande des Hautes Fréquences est la partie du spectre radioélectrique s'étendant de 3 MHz à 30 MHz. 29

HMM Modèle de Markov caché (ou Hidden Markov Model). 1, 35

HMM-DNN combinaison d'un réseau de neurones profond et de modèles de Markov cachés (*Hidden Markov Model-Deep Neural Network*). 2, 34, 36, 38

IARPA Intelligence Advanced Research Projects Activity. 27

IdL Identification de la langue. 31–38, 80, 81, 83, 85, 86, 88, 89, 91, 94, 98, 99, 102–104, 106, 125, 127

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno. 108

LER taux d'erreur d'identification de la langue (ou Language Error Rate). 36, 81, 83–88, 90, 91, 99, 104, 106, 125

LIMSI Le Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur (LIMSI) est un laboratoire de recherche pluridisciplinaire du CNRS qui rassemble des chercheurs et enseignants-chercheurs relevant de différentes disciplines des Sciences de l'Ingénieur et des Sciences de l'Information ainsi que des Sciences du Vivant et des Sciences Humaines et Sociales. 34

LNE Laboratoire National de métrologie et d'Essais. 29

LPC Linear Predictive Coding. 1

LRE-07 NIST Language Recognition Evaluation of 2007. 35–38, 55, 56, 77, 80–83, 91, 96–101, 103, 104, 106, 119, 125, 127

LRE-15 NIST Language Recognition Evaluation of 2015. 35, 38, 40, 77, 80, 83–91, 96, 104–106, 119, 120, 125–127

LSTM Long Short-Term Memory. 16, 21, 25, 41–48, 50, 51, 53–56, 63–65, 72, 73, 75, 76, 98, 108, 119, 125

LTSV Long-Term Signal Variability, critère de SAD basé sur la variabilité du signal sur le long-terme. 24, 59, 60, 75

MFCC coefficients cepstraux (Mel Frequency Cepstral Coefficients) calculés par une transformée en cosinus discrète appliquée au spectre de puissance d'un signal. Les bandes de fréquence de ce spectre sont espacées logarithmiquement selon l'échelle de Mel. 25, 33, 64, 65, 67, 68, 71

MLP perceptron multi-couches (*MultiLayer Perceptron*). 2, 5, 7–13, 15–17, 19, 21, 22, 25, 27, 34, 35, 53–56, 64, 65, 75, 76, 93, 94, 111, 119, 125

NAG Nesterov's Accelerated Gradient. 108

NIST National Institute of Standards and Technology. 1, 23, 27, 31, 35–38, 72, 76, 77, 80, 81, 85, 91, 96, 104, 110, 119, 129, 130

PLDA Probabilistic Linear Discriminant Analysis. 32, 33

PLP Perceptual Linear Predictive coefficents. 34, 80

PPRLM Parallel Phone Recognizer followed by Language Modeling. 34, 125

PSO Particle Swarm Optimization. 111

QPSO Quantum-Behaved Particle Swarm Optimization. 57–60, 64, 67, 68, 71, 75, 80, 111, 119

RAP reconnaissance automatique de la parole. 23, 25, 27, 57–64, 68–70, 72, 129

RATS Robust Automatic Transcription of Speech. 23, 29, 72, 76

ReLU Rectified Linear Unit. 16, 115

REPERE Campagne d'évaluation REconnaissance de PERsonnes dans des Emissions audiovisuelles (REPERE). 29, 30, 74

RMSPROP RPROP where the gradient is divided by a the root of the running average of squared (or Root Mean Squared) gradient. 108, 116

RNN réseau de neurones récurrent (*Recurrent Neural Network*). 5, 16–22, 25, 33, 41–43, 47, 53–57, 59, 63–66, 71, 76–80, 85, 91, 93–97, 99, 101, 102, 104, 106, 108, 109, 115–119, 125

RPROP Resilient back-propagation. 108, 109

SAD détection de la parole (ou Speech Activity Detection). 23–26, 29, 30, 53–55, 57–75, 80, 119, 125, 127

SDC Shifted Delta Cepstral. 33

SFO Sum of Functions Optimizer. 108, 109, 116

SMORMS3 Squared Mean Over Root Mean Squared Cubed. 60, 67, 71, 75, 80, 96, 108, 109, 116, 119

SNR rapport signal sur bruit (ou Signal to Noise Ratio). 23, 24

UHF La bande des Ultra Hautes Fréquences est la bande du spectre radioélectrique comprise entre 300 MHz et 3 000 MHz. 29

VHF La bande des très hautes fréquences (Very High Frequency) est la partie du spectre radioélectrique s'étendant de 30 MHz à 300 MHz. 29

VTLN Vocal Tract Length Normalization. 80

VTLP Vocal Tract Length Perturbation. 117

WER Word Error Rate ou taux d'erreur de mots est une unité de mesure classique pour mesurer les performances d'un système de reconnaissance de la parole. Le WER est dérivé de la distance de Levenshtein, en travaillant au niveau des mots au lieu des caractères. Il indique le taux de mots incorrectement reconnus par rapport à un texte de référence. Plus le taux est faible (minimum 0) plus la reconnaissance est bonne. Le taux maximum n'est pas borné et peut dépasser 1 en cas de très mauvaise reconnaissance s'il y a beaucoup d'insertions. Après avoir aligné de manière optimale le texte reconnu avec la référence grâce à un algorithme de programmation dynamique, le taux d'erreur de mots est donné par:

$$WER = \frac{S + D + I}{N} \tag{6.10}$$

où N est le nombre de mots dans la référence, S est le nombre de substitutions (mots incorrectement reconnus), D est le nombre de suppressions (mots omis), I est le nombre d'insertions (mots ajoutés). 3, 25, 27, 57–64, 68–72, 111, 119, 125, 127

Références

[1] A. Graves, "Supervised sequence labelling with recurrent neural networks," *Springer*, vol. 385, 2012. 2, 21, 114

[2] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6645–6649, IEEE, 2013. 2, 21

[3] W. Y. Huang and R. P. Lippmann, "Neural net and traditional classifiers," in *Neural information processing systems*, pp. 387–396, 1988. 5

[4] J. L. Elman and D. Zipser, "Learning the hidden structure of speech," *The Journal of the Acoustical Society of America*, vol. 83, no. 4, pp. 1615–1626, 1988. 5

[5] T. Kohonen, "The neural phonetic typewriter," *Computer*, vol. 21, no. 3, pp. 11–22, 1988. 5

[6] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[7] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural networks*, vol. 3, no. 1, pp. 23–43, 1990. 5, 41

[8] T. Robinson and F. Fallside, "A recurrent error propagation network speech recognition system," *Computer Speech & Language*, vol. 5, no. 3, pp. 259–274, 1991. 5

[9] T. Robinson, M. Hochberg, and S. Renals, "The use of recurrent neural networks in continuous speech recognition," in *Automatic speech and speaker recognition*, pp. 233–258, Springer, 1996.

[10] P. Matejka, L. Burget, P. Schwarz, and J. Cernocky, "Brno university of technology system for nist 2005 language recognition evaluation," in *Speaker and Language Recognition Workshop, 2006. IEEE Odyssey 2006 : The*, pp. 1–7, IEEE, 2006.

[11] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification : labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376, ACM, 2006. 5

[12] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943. 7

[13] F. Rosenblatt, "The perceptron, a perceiving and recognizing automaton," *Cornell Aeronautical Laboratory*, 1957. 7

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. 7

[15] D. Lowe and D. Broomhead, "Multivariable functional interpolation and adaptive networks," *Complex systems*, vol. 2, no. 3, pp. 321–355, 1988. 7

[16] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, no. 1, pp. 59–69, 1982. 7

[17] D. O. Hebb, "The organization of behavior : A neuropsychological theory," *Psychology Press*, 1949. 7

[18] P. Werbos, "Beyond regression : New tools for prediction and analysis in the behavioral sciences," *Harvard University*, 1974. 8

[19] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System modeling and optimization*, pp. 762–770, Springer, 1982. 8

[20] D. B. Parker, "Learning logic," *MIT*, 1985. 8

[21] Y. LeCun, "Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks)," in *Proceedings of Cognitiva 85, Paris, France*, 1985.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986. 8

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., DTIC Document, 1986. 8, 12

[24] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989. 8

[25] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990. 16

[26] M. I. Jordan, "Serial order : A parallel distributed processing approach," *Advances in psychology*, vol. 121, pp. 471–495, 1997. 16

[27] H. Jaeger and H. Haas, "Harnessing nonlinearity : Predicting chaotic systems and saving energy in wireless communication," *science*, vol. 304, no. 5667, pp. 78–80, 2004. 16

[28] B. Hammer, "On the approximation capability of recurrent neural networks," *Neurocomputing*, vol. 31, no. 1, pp. 107–123, 2000. 17

[29] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, 2011. 17

[30] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628, IEEE, 2013.

[31] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks.," *ICML (3)*, vol. 28, pp. 1310–1318, 2013. 17

[32] P. J. Werbos, "Backpropagation through time : what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990. 19

[33] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997. 21

[34] P. Baldi, S. Brunak, P. Frasconi, G. Soda, and G. Pollastri, "Exploiting the past and the future in protein secondary structure prediction," *Bioinformatics*, vol. 15, no. 11, pp. 937–946, 1999. 21

[35] K. Itoh and M. Mizushima, "Environmental noise reduction based on speech/non-speech identification for hearing aids," in *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, vol. 1, pp. 419–422, IEEE, 1997. 23

[36] K. Ngo, A. Spriet, M. Moonen, J. Wouters, and S. H. Jensen, "Variable speech distortion weighted multichannel wiener filter based on soft output voice activity detection for noise reduction in hearing aids," *Proc. 11th IWAENC*, 2008. 23

[37] H.-G. Hirsch and C. Ehrlicher, "Noise estimation techniques for robust speech recognition," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 1, pp. 153–156, IEEE, 1995. 23

[38] J. Ramirez, J. M. Górriz, and J. C. Segura, *Voice activity detection. fundamentals and speech recognition system robustness*. INTECH Open Access Publisher, 2007. 23, 24

[39] J. Sohn and W. Sung, "A voice activity detector employing soft decision based noise spectrum adaptation," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 1, pp. 365–368, IEEE, 1998. 23

[40] J. Sohn, N. S. Kim, and W. Sung, "A statistical model-based voice activity detection," *IEEE signal processing letters*, vol. 6, no. 1, pp. 1–3, 1999.

[41] D. Enqing, Z. Heming, and L. Yongli, "Low bit and variable rate speech coding using local cosine transform," in *TENCON'02. Proceedings. 2002 IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering*, vol. 1, pp. 423–426, IEEE, 2002. 23

[42] S. Van Gerven and F. Xie, "A comparative study of speech detection methods.," in *Eurospeech*, vol. 97, 1997. 24

[43] P. C. Khoa, "Noise robust voice activity detection," *Nanyang Technological University*, 2012. 24

[44] L. Lamel, L. Rabiner, A. E. Rosenberg, and J. G. Wilpon, "An improved endpoint detector for isolated word recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 29, no. 4, pp. 777–785, 1981. 24

[45] G. Evangelopoulos and P. Maragos, "Speech event detection using multiband modulation energy.," in *Interspeech*, pp. 685–688, 2005. 24

[46] A. Craciun and M. Gabrea, "Correlation coefficient-based voice activity detector algorithm," in *Electrical and Computer Engineering, 2004. Canadian Conference on*, vol. 3, pp. 1789–1792, IEEE, 2004. 24

[47] T. Kristjansson, S. Deligne, and P. Olsen, "Voicing features for robust speech detection," *Entropy*, vol. 2, no. 2.5, p. 3, 2005. 24

[48] H. Ghaemmaghami, B. J. Baker, R. J. Vogt, and S. Sridharan, "Noise robust voice activity detection using features extracted from the time-domain autocorrelation function," *Proceedings of Interspeech 2010*, 2010. 24

[49] L. Yoon-Chang and A. Sang-Sik, "Statistical model-based vad algorithm with wavelet transform," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 89, no. 6, pp. 1594–1600, 2006. 24

[50] P. Renevey and A. Drygajlo, "Entropy based voice activity detection in very noisy conditions," *threshold*, vol. 5, no. 5.5, p. 6, 2001. 24

[51] R. Prasad, H. Saruwatari, and K. Shikano, "Noise estimation using negentropy based voice-activity detector," in *Circuits and Systems, 2004. MWSCAS'04. The 2004 47th Midwest Symposium on*, vol. 2, pp. II–149, IEEE, 2004. 24

[52] J. Ramırez, J. C. Segura, C. Benıtez, A. De La Torre, and A. Rubio, "Efficient voice activity detection algorithms using long-term speech information," *Speech communication*, vol. 42, no. 3, pp. 271–287, 2004. 24

[53] P. K. Ghosh, A. Tsiartas, and S. Narayanan, "Robust voice activity detection using long-term signal variability," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 3, pp. 600–613, 2011. 24

[54] J. Gauvain, L. Lamel, and G. Adda, "Partitioning and transcription of broadcast news data," pp. 1335–1338, 1998. 25

[55] I. Shafran and R. Rose, "Robust speech detection and segmentation for real-time ASR applications," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, vol. 1, pp. I–432, IEEE, 2003. 25, 63

[56] T. Ng, B. Zhang, L. Nguyen, S. Matsoukas, X. Zhou, N. Mesgarani, K. Veselỳ, and P. Matejka, "Developing a Speech Activity Detection System for the DARPA RATS Program.," in *Interspeech*, 2012.

[57] G. Saon, S. Thomas, H. Soltau, S. Ganapathy, and B. Kingsbury, "The IBM speech activity detection system for the DARPA RATS program.," in *Interspeech*, pp. 3497–3501, 2013. 25, 63

[58] N. Ryant, M. Liberman, and J. Yuan, "Speech activity detection on youtube using deep neural networks.," in *Interspeech*, pp. 728–731, 2013. 25, 63

[59] T. Drugman, Y. Stylianou, Y. Kida, and M. Akamine, "Voice activity detection : Merging source and filter-based information," *IEEE Signal Processing Letters*, vol. 23, no. 2, pp. 252–256, 2016. 25, 63

[60] T. Hughes and K. Mierle, "Recurrent neural networks for voice activity detection," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 7378–7382, IEEE, 2013. 25, 63

[61] F. Eyben, F. Weninger, S. Squartini, and B. Schuller, "Real-life voice activity detection with lstm recurrent neural networks and an application to hollywood movies," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 483–487, IEEE, 2013.

[62] G. Gelly and J.-L. Gauvain, "Minimum word error training of rnn-based voice activity detection," *Interspeech*, 2015.

[63] S. Tong, H. Gu, and K. Yu, "A comparative study of robustness of deep learning approaches for vad," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5695–5699, IEEE, 2016. 25, 63

[64] R. Lileikyte, L. Lamel, and J.-L. Gauvain, "Conversational telephone speech recognition for lithuanian," in *International Conference on Statistical Language and Speech Processing*, pp. 164–172, Springer, 2015. 27

[65] T. Fraga-Silva, J.-L. Gauvain, L. Lamel, A. Laurent, V. B. Le, and A. Messaoudi, "Active learning based data selection for limited resource stt and kws.," in *Interspeech*, pp. 3159–3163, 2015. 27

[66] V.-B. Le, L. Lamel, A. Messaoudi, W. Hartmann, J.-L. Gauvain, C. Woehrling, J. Despres, and A. Roy, "Developing STT and KWS systems using limited language resources," in *Interspeech*, 2014. 27

[67] J. Kahn, O. Galibert, L. Quintard, M. Carré, A. Giraudel, and P. Joly, "A presentation of the repere challenge," in *Content-Based Multimedia Indexing (CBMI), 2012 10th International Workshop on*, pp. 1–6, IEEE, 2012. 29

[68] P. Gay, "Segmentation et identification audiovisuelle de personnes dans des journaux télévisés," *Université du Maine*, 2015. 29

[69] A. Roy, C. Guinaudeau, H. Bredin, and C. Barras, "Tvd : A reproducible and multiply aligned tv series dataset.," in *LREC*, pp. 418–425, 2014. 30

[70] A. Waibel, P. Geutner, L. M. Tomokiyo, T. Schultz, and M. Woszczyna, "Multilinguality in speech and spoken language systems," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1297–1313, 2000. 31

[71] T. Schultz and A. Waibel, "Language-independent and language-adaptive acoustic modeling for speech recognition," *Speech Communication*, vol. 35, no. 1, pp. 31–51, 2001. 31

[72] C. Chelba, T. J. Hazen, and M. Saraclar, "Retrieval and browsing of spoken content," *IEEE Signal Processing Magazine*, vol. 25, no. 3, 2008. 31

[73] M. P. Lewis, G. F. Simons, and C. D. Fennig, *Ethnologue : Languages of the world*, vol. 16. SIL international Dallas, TX, 2009. 31

[74] H. Li, B. Ma, and K. A. Lee, "Spoken language recognition : from fundamentals to practice," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1136–1159, 2013. 31

[75] J. Zhao, H. Shu, L. Zhang, X. Wang, Q. Gong, and P. Li, "Cortical competition during language discrimination," *NeuroImage*, vol. 43, no. 3, pp. 624–633, 2008. 31

[76] F. Ramus and J. Mehler, "Language identification with suprasegmental cues : A study based on speech resynthesis," *The Journal of the Acoustical Society of America*, vol. 105, no. 1, pp. 512–521, 1999. 31

[77] Y. K. Muthusamy, E. Barnard, and R. A. Cole, "Reviewing automatic language identification," *IEEE Signal Processing Magazine*, vol. 11, no. 4, pp. 33–41, 1994. 31

[78] M. A. Zissman *et al.*, "Comparison of four approaches to automatic language identification of telephone speech," *IEEE Transactions on speech and audio processing*, vol. 4, no. 1, p. 31, 1996. 31, 34

[79] D. Cimarusti and R. Ives, "Development of an automatic identification system of spoken languages : Phase i," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'82.*, vol. 7, pp. 1661–1663, IEEE, 1982. 32

[80] J. Foil, "Language identification using noisy speech," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'86.*, vol. 11, pp. 861–864, IEEE, 1986.

[81] F. J. Goodman, A. F. Martin, and R. E. Wohlford, "Improved automatic language identification in noisy speech," in *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 international conference on*, pp. 528–531, IEEE, 1989.

[82] M. Sugiyama, "Automatic language recognition using acoustic features," in *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pp. 813–816, IEEE, 1991.

[83] S. Nakagawa, Y. Ueda, and T. Seino, "Speaker-independent, text-independent language identification by hmm.," in *ICSLP*, vol. 92, pp. 1011–1014, 1992.

[84] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-Carrasquillo, "Support vector machines for speaker and language recognition," *Computer Speech & Language*, vol. 20, no. 2, pp. 210–229, 2006.

[85] W. Shen and D. Reynolds, "Improved gmm-based language recognition using constrained mllr transforms," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pp. 4149–4152, IEEE, 2008.

[86] W. M. Campbell, "A covariance kernel for svm language recognition," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pp. 4141–4144, IEEE, 2008.

[87] D. Zhu, B. Ma, and H. Li, "Soft margin estimation of gaussian mixture model parameters for spoken language recognition," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 4990–4993, IEEE, 2010. 32

[88] P. Kenny, "Bayesian speaker verification with heavy-tailed priors.," in *Odyssey*, p. 14, 2010. 32

[89] D. Najim, K. Patrick, D. Reda, D. Pierre, and O. Pierre, "Front-End Factor Analysis for Speaker Verification," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 19, no. 4, pp. 788–798, 2011.

[90] D. Garcia-Romero and C. Y. Espy-Wilson, "Analysis of i-vector length normalization in speaker recognition systems.," in *Interspeech*, pp. 249–252, 2011. 32

[91] D. Martınez, O. Plchot, L. Burget, O. Glembek, and P. Matejka, "Language recognition in ivectors space," *Proceedings of Interspeech, Firenze, Italy*, pp. 861–864, 2011. 32, 33

[92] N. Dehak, P. A. Torres-Carrasquillo, D. A. Reynolds, and R. Dehak, "Language recognition via i-vectors and dimensionality reduction.," in *Interspeech*, pp. 857–860, Citeseer, 2011. 32

[93] S. Ioffe, "Probabilistic linear discriminant analysis," *Computer Vision–ECCV 2006*, pp. 531–542, 2006. 32

[94] S. J. Prince and J. H. Elder, "Probabilistic linear discriminant analysis for inferences about identity," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, IEEE, 2007. 32

[95] J. Bridle and M. Brown, "An experimental automatic word recognition system," *JSRU Report*, vol. 1003, no. 5, 1974. 33

[96] P. A. Torres-Carrasquillo, E. Singer, M. A. Kohler, R. J. Greene, D. A. Reynolds, and J. R. Deller Jr, "Approaches to language identification using gaussian mixture models and shifted delta cepstral features.," in *Interspeech*, 2002. 33

[97] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977. 33

[98] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, *et al.*, "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. EPFL-CONF-192584, IEEE Signal Processing Society, 2011. 33

[99] H. Wang, C.-C. Leung, T. Lee, B. Ma, and H. Li, "Shifted-delta mlp features for spoken language recognition," *IEEE Signal Processing Letters*, vol. 20, no. 1, pp. 15–18, 2013. 33

[100] P. Matejka, L. Zhang, T. Ng, H. S. Mallidi, O. Glembek, J. Ma, and B. Zhang, "Neural network bottleneck features for language identification," *Proc. IEEE Odyssey*, pp. 299–304, 2014. 33

[101] D. Imseng, H. Bourlard, *et al.*, "Hierarchical multilayer perceptron based language identification," in *Proceedings of Interspeech*, no. EPFL-CONF-150597, 2010. 33

[102] J. Gonzalez-Dominguez, I. Lopez-Moreno, H. Sak, J. Gonzalez-Rodriguez, and P. J. Moreno, "Automatic language identification using long short-term memory recurrent neural networks.," in *Interspeech*, 2014. 33

[103] G. Gelly, J.-L. Gauvain, V. Le, and A. Messaoudi, "A divide-and-conquer approach for language identification based on recurrent neural networks," *Interspeech 2016*, pp. 3231–3235, 2016. 33

[104] J.-L. Gauvain, A. Messaoudi, and H. Schwenk, "Language recognition using phone latices.," in *Interspeech*, 2004. 33, 35

[105] A. Lawson, M. McLaren, Y. Lei, V. Mitra, N. Scheffer, L. Ferrer, and M. Graciarena, "Improving language identification robustness to highly channel-degraded speech through multiple system fusion.," in *Interspeech*, pp. 1507–1510, 2013. 33

[106] R. G. Leonard and G. R. Doddington, "Automatic language identification.," tech. rep., DTIC Document, 1974. 33

[107] K. M. Berkling, T. Arai, and E. Barnard, "Analysis of phoneme-based features for language identification," in *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, vol. 1, pp. I–289, IEEE, 1994.

[108] T. J. Hazen and V. W. Zue, "Recent improvements in an approach to segment-based automatic language identification.," in *ICSLP*, vol. 94, pp. 1883–1886, Citeseer, 1994.

[109] L. F. Lamel and J.-L. Gauvain, "Language identification using phone-based acoustic likelihoods," in *ICASSP*, 1994. 34

[110] R. C. Tucker, M. J. Carey, and E. S. Parris, "Automatic language identification using sub-word models," in *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, vol. 1, pp. I–301, IEEE, 1994.

[111] M. A. Zissman, "Automatic language identification using gaussian mixture and hidden markov models," in *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, vol. 2, pp. 399–402, IEEE, 1993. 33

[112] L. Lamel and J.-L. Gauvain, "Identifying non-linguistic speech features.," in *Eurospeech*, 1993. 34

[113] L. F. Lamel and J.-L. Gauvain, "A phone-based approach to non-linguistic speech feature identification," *Computer Speech & Language*, vol. 9, no. 1, pp. 87–103, 1995. 34

[114] M. F. BenZeghiba, J. Gauvain, and L. Lamel, "Phonotactic language recognition using MLP features," in *Interspeech 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pp. 2041–2044, 2012. 34

[115] M. F. BenZeghiba, J.-L. Gauvain, and L. Lamel, "Fusing language information from diverse data sources for phonotactic language recognition.," in *Odyssey*, pp. 346–352, 2012. 34

[116] M. J. Gales, "Maximum likelihood linear transformations for hmm-based speech recognition," *Computer speech & language*, vol. 12, no. 2, pp. 75–98, 1998. 34

[117] D. Zhu and M. Adda-Decker, "Language identification using lattice-based phonotactic and syllabotactic approaches," in *Speaker and Language Recognition Workshop, 2006. IEEE Odyssey 2006 : The*, pp. 1–4, IEEE, 2006. 35

[118] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the em algorithm," *Neural computation*, vol. 6, no. 2, pp. 181–214, 1994. 35

[119] J. Kittler, "Combining classifiers : A theoretical framework," *Pattern analysis and Applications*, vol. 1, no. 1, pp. 18–27, 1998. 35

[120] G. E. Hinton, "Products of experts," *IET*, 1999. 35

[121] NIST, "The 2015 nist language recognition evaluation plan (lre15)," 2015. `http://www.itl.nist.gov/iad/mig//tests/lre/`. 36

[122] NIST, "Nist lre-2007 evaluation plan," 2007. `http://www.itl.nist.gov/iad/mig/tests/lre/2007/LRE07EvalPlan-v8b.pdf`. 36

[123] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994. 41

[124] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets : the difficulty of learning long-term dependencies," *A field guide to dynamical recurrent neural networks. IEEE Press*, 2001. 41

[125] T. Lin, B. G. Horne, P. Tino, and C. L. Giles, "Learning long-term dependencies in narx recurrent neural networks," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1329–1338, 1996. 41

[126] J. Schmidhuber, "Learning complex, extended sequences using the principle of history compression," *Neural Computation*, vol. 4, no. 2, pp. 234–242, 1992. 41

[127] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 41

[128] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002. 41

[129] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet : A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015. 95

[130] H. Bredin, "TristouNet : Triplet Loss for Speaker Turn Embedding," in *ICASSP 2017, IEEE International Conference on Acoustics, Speech, and Signal Processing*, (New Orleans, USA), March 2017. 95

[131] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008. 99, 104

[132] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964. 108

[133] D. C. Plaut *et al.*, "Experiments on learning by back propagation.," *ERIC*, 1986. 108

[134] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983. 108

[135] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning.," *ICML (3)*, vol. 28, pp. 1139–1147, 2013. 108

[136] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning : The rprop algorithm," in *Neural Networks, 1993., IEEE International Conference On*, pp. 586–591, IEEE, 1993. 108

[137] C. Igel and M. Hüsken, "Improving the rprop learning algorithm," in *Proceedings of the second international ICSC symposium on neural computation (NC 2000)*, vol. 2000, pp. 115–121, Citeseer, 2000. 108

[138] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995. 108

[139] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude," *COURSERA : Neural Networks for Machine Learning*, vol. 4, p. 2, 2012. 108

[140] D. Kingma and J. Ba, "Adam : A method for stochastic optimization," *arXiv preprint arXiv :1412.6980*, 2014. 108

[141] J. Sohl-Dickstein, B. Poole, and S. Ganguli, "An adaptive low dimensional quasi-newton sum of functions optimizer," *CoRR*, vol. abs/1311.2115, 2013. 108

[142] S. Funk, "Rmsprop loses to smorms3," 2015. `http://sifter.org/~simon/journal/20150420.html`. 108

[143] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates.," *ICML (3)*, vol. 28, pp. 343–351, 2013. 108

[144] G. Gelly and P. Vernis, "Neural Networks as a Guidance Solution for Soft-Landing and Aerocapture," in *AIAA Guidance, Navigation, and Control Conference Chicago, Illinois*, 2009. 111

[145] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in *Congress on Evolutionary Computation*, 2004. 111

[146] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 1, pp. 39–43, New York, NY, 1995. 111

[147] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, 2002. 111

[148] J. Sun, X. Wu, V. Palade, W. Fang, C.-H. Lai, and W. Xu, "Convergence analysis and improvements of quantum-behaved particle swarm optimization," *Information Sciences*, vol. 193, pp. 81–103, 2012. 111

[149] V. N. Vapnik, "An overview of statistical learning theory," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999. 113

[150] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks : Tricks of the trade*, pp. 9–48, Springer, 2012. 115

[151] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks.," in *Aistats*, vol. 9, pp. 249–256, 2010. 115

[152] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers : Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015. 116

[153] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning : Generalization gap and sharp minima," *arXiv preprint arXiv :1609.04836*, 2016. 116

[154] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429–1451, 2003. 116

[155] Y. Bengio, I. J. Goodfellow, and A. Courville, "Deep learning," *An MIT Press book in preparation. Draft chapters available at http ://www. iro. umontreal. ca/ bengioy/dlbook*, 2015. 116

[156] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition.," in *Interspeech*, pp. 3586–3589, 2015. 117

[157] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, *et al.*, "Deep speech : Scaling up end-to-end speech recognition," *arXiv preprint arXiv :1412.5567*, 2014. 117

[158] N. Jaitly and G. E. Hinton, "Vocal tract length perturbation (vtlp) improves speech recognition," in *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language*, 2013. 117

[159] X. Cui, V. Goel, and B. Kingsbury, "Data augmentation for deep neural network acoustic modeling," *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 23, no. 9, pp. 1469–1477, 2015. 117

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Réseaux de neurones récurrents pour le traitement automatique de la parole

Mots clefs : réseaux de neurones récurrents, LSTM, reconnaissance de la parole

Résumé : Le domaine du traitement automatique de la parole regroupe un très grand nombre de tâches parmi lesquelles on trouve la reconnaissance de la parole, l'identification de la langue ou l'identification du locuteur. Ce domaine de recherche fait l'objet d'études depuis le milieu du vingtième siècle mais la dernière rupture technologique marquante est relativement récente et date du début des années 2010. C'est en effet à ce moment qu'apparaissent des systèmes hybrides utilisant des réseaux de neurones profonds (DNN) qui améliorent très notablement l'état de l'art.
Inspirés par le gain de performance apporté par les DNN et par les travaux d'Alex Graves sur les réseaux de neurones récurrents (RNN), nous souhaitions explorer les capacités de ces derniers. En effet, les RNN nous semblaient plus adaptés que les DNN pour traiter au mieux les séquences temporelles du signal de parole. Dans cette thèse, nous nous intéressons tout particulièrement aux RNN à mémoire court-terme persistante (Long Short Term Memory (LSTM)) qui permettent de s'affranchir d'un certain nombre de difficultés rencontrées avec des RNN standards. Nous augmentons ce modèle et nous proposons des processus d'optimisation permettant d'améliorer les performances obtenues en segmentation parole/non-parole et en identification de la langue. En particulier, nous introduisons des fonctions de coût dédiées à chacune des deux tâches: un simili-WER pour la segmentation parole/non-parole dans le but de diminuer le taux d'erreur d'un système de reconnaissance de la parole et une fonction de coût dite de proximité angulaire pour les problèmes de classification multi-classes tels que l'identification de la langue parlée.

Title : Speech processing using recurrent neural networks

Keywords : recurrent neural networks, long short term memory, speech recognition

Abstract : Automatic speech processing is an active field of research since the 1950s. Within this field the main area of research is automatic speech recognition but simpler tasks such as speech activity detection, language identification or speaker identification are also of great interest to the community.
The most recent breakthrough in speech processing appeared around 2010 when speech recognition systems using deep neural networks drastically improved the state-of-the-art.
Inspired by this gains and the work of Alex Graves on recurrent neural networks (RNN), we decided to explore the possibilities brought by these models on realistic data for two different tasks: speech activity detection and spoken language identification.
In this work, we closely look at a specific model for the RNNs: the Long Short Term Memory (LSTM) which mitigates a lot of the difficulties that can arise when training an RNN. We augment this model and introduce optimization methods that lead to significant performance gains for speech activity detection and language identification. More specifically, we introduce a WER-like loss function to train a speech activity detection system so as to minimize the word error rate of a downstream speech recognition system. We also introduce two different methods to successfully train a multiclass classifier based on neural networks for tasks such as LID. The first one is based on a divide-and-conquer approach and the second one is based on an angular proximity loss function. Both yield performance gains but also speed up the training process.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France