Exploitation de transcriptions bruitées pour la reconnaissance automatique de la parole

Adrien Dufraux

▶ **To cite this version:**

Adrien Dufraux. Exploitation de transcriptions bruitées pour la reconnaissance automatique de la parole. Informatique [cs]. Université de Lorraine, 2022. Français. NNT : 2022LORR0032 . tel-03669875

HAL Id: tel-03669875
https://hal.univ-lorraine.fr/tel-03669875

Submitted on 17 May 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

UNIVERSITÉ DE LORRAINE

Thèse de Doctorat

Adrien Dufraux

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Lorraine
Mention Informatique

École doctorale : IAEM

Unité de recherche : Laboratoire Lorrain de Recherche en Informatique et ses Applications UMR 7503

Soutenue le 14 avril 2022

Exploitation de transcriptions bruitées pour la reconnaissance automatique de la parole

JURY

Présidente :	**Lori Lamel**, Directrice de Recherche, CNRS - LISN
Rapporteur :	**Yannick Estève**, Professeur, Avignon Université - LIA
Rapporteur :	**Anthony Larcher**, Professeur, Le Mans Université - LIUM
Examinateur :	**Matthijs Douze**, Chercheur, Meta AI
Directeur de thèse :	**Emmanuel Vincent**, Directeur de Recherche, Inria Nancy - Grand Est
Directeur de thèse :	**Armelle Brun**, Maître de Conférences, Université de Lorraine - LORIA

Résumé

Les méthodes usuelles pour la conception d'un système de reconnaissance automatique de la parole nécessitent des jeux de données de parole transcrite de bonne qualité. Ceux-ci sont composés du signal acoustique produit par un locuteur ainsi que de la transcription mot à mot de ce qui a été dit. Pour construire un bon modèle de reconnaissance automatique il faut plusieurs milliers d'heures de parole transcrite. Le jeu de données doit être créé à partir d'un panel de locuteurs et de situations différentes pour couvrir la variabilité de la parole et de la langue. Pour créer un tel jeu de données, on demande généralement à des annotateurs humains d'écouter les signaux acoustiques et d'écrire le texte correspondant. Ce procédé coûte cher et est source d'erreurs car ce qui est dit lors d'un enregistrement en conditions réelles n'est pas toujours facilement intelligible. Des signaux mal transcrits impliquent une baisse de performance du modèle acoustique. Pour améliorer la qualité des transcriptions, plusieurs personnes peuvent annoter le même signal acoustique, mais alors le procédé coûte encore plus cher.

Cette thèse prend le contre-pied de cette démarche et propose de concevoir des algorithmes permettant d'utiliser des jeux de données dont les transcriptions sont « bruitées », c'est-à-dire qu'elles contiennent des erreurs. Le but principal est donc de réduire les coûts pour construire un système de reconnaissance automatique de la parole en limitant la perte de qualité du système induite par ces erreurs.

Dans un premier temps, nous présentons l'algorithme Lead2Gold. Lead2Gold est basé sur une fonction de coût qui permet d'utiliser des jeux de données dont les transcriptions contiennent des erreurs. Nous modélisons ces erreurs par un modèle de bruit simple basé au niveau des lettres. Pour une transcription présente dans le jeu de données, l'algorithme cherche un ensemble de transcriptions probablement meilleures. Nous utilisons pour cela une recherche en faisceau dans le graphe. Une telle technique de recherche n'est habituellement pas utilisée pour la formulation d'une fonction de coût. Nous montrons qu'il est possible d'ajouter explicitement de nouveaux éléments, ici un modèle de bruit, pour créer des fonctions de coût complexes.

Ensuite nous améliorons la formulation de Lead2Gold pour que la fonction de coût soit modulable. Pour cela, nous utilisons des wFST. Les wFST sont des graphes dont les arcs sont pondérés et représentent des symboles. Nous pouvons composer différents graphes pour construire des fonctions de coût de façon flexible. Avec notre proposition, il devient plus facile d'ajouter de nouveaux éléments, comme un lexique, pour mieux caractériser les bonnes transcriptions. Nous montrons que l'utilisation des wFST est une

bonne alternative à l'utilisation explicite de la recherche en faisceau de Lead2Gold. La formulation modulaire nous permet de proposer une nouvelle gamme de fonctions de coût modélisant les erreurs de transcription.

Enfin nous procédons à une expérience de collecte de données en conditions réelles. Nous observons les différents profils d'annotateurs. Les annotateurs n'ont pas la même perception des signaux acoustiques et les erreurs qu'ils commettent peuvent être de natures différentes. Le but explicite de cette expérience est d'obtenir des transcriptions erronées et de prouver l'utilité de modéliser ces erreurs.

Abstract

Usual methods to design automatic speech recognition systems require speech datasets with high quality transcriptions. These datasets are composed of the acoustic signals uttered by speakers and the corresponding word-level transcripts representing what is being said. It takes several thousand hours of transcribed speech to build a good speech recognition model. The dataset must include a variety of speakers recorded in different situations in order to cover the wide variability of speech and language. To create such a system, human annotators are asked to listen to audio tracks and to write down the corresponding text. This process is costly and can lead to errors. What is being said in realistic settings is indeed not always easy to understand. Poorly transcribed signals cause a drop of performance of the acoustic model. To improve the quality of the transcripts, the same utterances may be transcribed by several people, but this leads to an even more expensive process.

This thesis takes the opposite view. We design algorithms which can exploit datasets with "noisy" transcriptions i.e., which contain errors. The main goal of this thesis is to reduce the costs of building an automatic speech recognition system by limiting the performance drop induced by these errors.

We first introduce the Lead2Gold algorithm. Lead2Gold is based on a cost function that is tolerant to datasets with noisy transcriptions. We model transcription errors at the letter level with a noise model. For each transcript in the dataset, the algorithm searches for a set of likely better transcripts relying on a beam search in a graph. This technique is usually not used to design cost functions. We show that it is possible to explicitly add new elements (here a noise model) to design complex cost functions.

We then express the Lead2Gold loss in the wFST formalism. wFSTs are graphs whose edges are weighted and represent symbols. To build flexible cost functions we can compose several graphs. With our proposal, it becomes easier to add new elements, such as a lexicon, to better characterize good transcriptions. We show that using wFSTs is a good alternative to using Lead2Gold's explicit beam search. The modular formulation allows us to design a new variety of cost functions that model transcription errors.

Finally, we conduct a data collection experiment in real conditions. We observe different types of annotator profiles. Annotators do not have the same perception of acoustic signals and hence can produce different types of errors. The explicit goal of this expe-

riment is to collect transcripts with errors and to prove the usefulness of modeling these errors.

Remerciements

Je tiens en premier lieu à remercier le jury de thèse. D'abord Mme la présidente du jury, Lori Lamel, pour avoir mené à bien la déroulement de la soutenance ainsi que pour ses questions et retours efficaces concernant l'état de l'art de la thèse. Ensuite, je remercie chaleureusement les rapporteurs M. Yannick Estève et M. Anthony Larcher pour leur lecture du manuscrit et pour leurs questions pertinentes lors de la soutenance. La bonne humeur et la bienveillance du jury m'ont particulièrement marquées. Grâce à eux, j'ai vécu une soutenance détendue et scientifiquement stimulante.

Ensuite, je veux remercier mes encadrants de thèse. J'ai effectué la moitié de ma thèse depuis chez moi à cause de la pandémie. La solitude et le choc psychologique liés à ces conditions étaient particulièrement difficiles à supporter au quotidien. Grâce au suivi régulier de mes encadrants, je suis tout de même allé au bout de la thèse. Pour cela, je vous remercie infiniment.

Merci Emmanuel Vincent pour l'expertise du domaine que tu m'as transmise. Grâce à toi, j'ai acquis de nombreuses connaissances précises sur la reconnaissance de la parole. C'est cette précision que je retiens de toi, et qui va de paire avec une rigueur mathématique que j'apprécie particulièrement.

Merci Armelle Brun pour ton encadrement. Grâce à toi, j'ai pu prendre du recul lors des moments les plus difficiles. Tu poses les bonnes questions, ce qui m'a souvent permis d'éviter de me perdre. Ton expertise scientifique était parfaitement complémentaire avec les autres encadrants, notamment avec le cinquième chapitre, me permettant de produire un manuscrit cohérent jusqu'au bout.

Merci Matthijs Douze pour ton encadrement et ton accueil au sein de l'entreprise Meta à Paris. Bien que tu viennes d'un domaine différent, ton expertise et ton expérience m'ont été très utiles pour ma formation scientifique.

Merci Awni Hannun pour ton encadrement en début de thèse et ton accueil à Meta New-York. Tu m'as donné l'envie d'utiliser les graphes, sur lesquels se base le chapitre quatre. Je retiens nos discussions stimulantes et je te remercie pour cela.

Je remercie l'entreprise Meta et l'Inria de Nancy pour m'avoir donné l'opportunité d'effectuer cette thèse. Je ne garde que des bons souvenirs des rencontres que j'ai faites, que ce soit à Meta ou au sein du laboratoire Multispeech de Nancy.

Enfin je remercie particulièrement ma famille et mes amis pour leur soutien sans faille et indispensable. Je remercie ma compagne Blandine pour tout le soutien moral ainsi que pour sa patience et sa relecture du manuscrit.

Table des matières

Liste des figures xiii

Liste des tableaux xv

Liste des algorithmes xvii

Liste des acronymes xix

1. Introduction 1
- 1.1. Motivation et cadre . . . 1
- 1.2. Problématique . . . 3
- 1.3. Les outils utilisés . . . 5
- 1.4. Contributions et plan du document . . . 7
- 1.5. Publication associée à cette thèse . . . 9

2. État de l'art 11
- 2.1. Du signal au texte . . . 11
  - 2.1.1. Représentation d'une donnée séquentielle . . . 11
  - 2.1.2. Transformation de l'observation dans le cas d'un signal acoustique 12
  - 2.1.3. Formulation générale du problème d'étiquetage de séquences . . . 14
  - 2.1.4. Du signal à la transcription : les étapes de la reconnaissance automatique de la parole . . . 15
- 2.2. Le transducteur pondéré à états finis . . . 16
  - 2.2.1. Définitions . . . 16
  - 2.2.2. Opérations sur les wFSA et les wFST . . . 20
  - 2.2.3. Algorithmes sur les wFSA et les wFST . . . 21
  - 2.2.4. Les wFST différentiables . . . 25
  - 2.2.5. Implémentation . . . 27
  - 2.2.6. Utilisation pour la RAP . . . 27
  - 2.2.7. L'apprentissage de wFST . . . 28
- 2.3. Le modèle acoustique . . . 29
  - 2.3.1. Historique . . . 29
  - 2.3.2. Architectures de réseaux de neurones profonds . . . 31
    - 2.3.2.1. Architecture *Gated ConvNet* . . . 32
    - 2.3.2.2. Architecture Transformer . . . 33
  - 2.3.3. Formulation de la fonction de coût . . . 34
    - 2.3.3.1. La fonction de coût CTC . . . 35

2.3.3.2. La fonction de coût ASG . . . 38
2.3.4. Algorithmes d'apprentissage . . . 39
2.4. Décodage du modèle acoustique . . . 42
2.4.1. Décodeurs . . . 42
2.4.2. Métrique . . . 43
2.5. Apprentissage à partir d'étiquettes bruitées . . . 43
2.5.1. Étiquettes bruitées pour la reconnaissance d'image . . . 44
2.5.2. Étiquettes bruitées pour la RAP . . . 45
2.5.3. Conclusion . . . 46

3. L'algorithme Lead2gold **49**
3.1. Modèle de bruit proposé . . . 49
3.2. Algorithme d'apprentissage proposé . . . 52
3.2.1. Fonction de coût prenant en compte le bruit . . . 52
3.2.2. Calcul de Z, S_{ASG} et S_{CTC} . . . 54
3.2.2.1. Formulation générale de l'algorithme dynamique . . . 56
3.2.2.2. Cas $G = G_{\text{full}}$ . . . 56
3.2.2.3. Cas $G = G_{\text{ASG}}$ . . . 58
3.2.2.4. Cas $G = G_{\text{CTC}}$ . . . 58
3.2.3. Calcul exact de S_{L2G} . . . 59
3.2.4. Approximation de S_{L2G} par une recherche en faisceau . . . 62
3.2.5. Recherche en faisceau différentiable . . . 66
3.3. Mise en place des expériences et observations préliminaires . . . 71
3.3.1. Introduction . . . 71
3.3.2. Obtention du modèle de bruit de transcription . . . 73
3.3.3. Création des jeux de données bruitées . . . 75
3.3.4. Évaluation . . . 75
3.3.5. Exemple qualitatif de résultat de Lead2Gold . . . 76
3.4. Expériences . . . 77
3.4.1. Protocole expérimental . . . 77
3.4.2. Résultats . . . 79
3.4.3. Interprétation . . . 81
3.4.4. Conclusion . . . 83

4. Extension modulaire de Lead2Gold basée sur les wFST **85**
4.1. Formulation de la fonction de coût . . . 85
4.1.1. Lien entre Lead2Gold et les wFST . . . 85
4.1.2. Construction de G_{full}, G_{CTC} et G_{ASG} avec des wFST . . . 86
4.1.3. Construction de G_{L2G} avec des wFST . . . 93
4.2. Nouveaux modules . . . 97
4.2.1. Utilisation de symboles différents pour le modèle de bruit . . . 97
4.2.2. Restrictions des hypothèses Y^* avec un lexique . . . 98
4.2.3. Intégration des modules . . . 98

4.3. Le modèle de bruit de transcription N . . . 100
4.3.1. Structures du modèle de bruit N . . . 100
4.3.2. Apprentissage des poids du modèle de bruit N . . . 102
4.4. Expériences . . . 103
4.4.1. Mise en place . . . 104
4.4.2. Évaluation des fonctionnalités . . . 106
4.4.3. Apprentissage du modèle de bruit . . . 108
4.4.4. Évaluation du coût de calcul . . . 109
4.4.5. Conclusion . . . 111

5. Collecte de transcriptions **113**
5.1. Mise en place de l'expérience . . . 113
5.1.1. Outils utilisés . . . 113
5.1.2. Objectifs et contraintes . . . 114
5.1.3. Le jeu de données à transcrire . . . 114
5.1.4. L'interface Web . . . 116
5.2. La collecte des données . . . 118
5.2.1. Calibrations et tests . . . 118
5.2.2. Lancement de la véritable expérience . . . 120
5.3. Exploitation des données collectées . . . 121
5.3.1. Analyse descriptive . . . 121
5.3.2. Apprentissage de modèles acoustiques . . . 126
5.3.3. Conclusion . . . 128

6. Conclusion et perspectives **131**
6.1. Résumé . . . 131
6.2. Conclusion . . . 132
6.3. Perspectives . . . 133
6.3.1. La collecte des transcriptions est-elle vraiment une bonne approche ? 133
6.3.2. Les transcriptions sont-elles vraiment nécessaires ? . . . 134
6.3.3. Le mot de la fin . . . 135

A. La fonction logadd **137**

B. Exemple d'utilisation des wFST différentiables **139**

C. Observation de $R(t)$ **143**

Bibliographie **145**

Liste des figures

1.1. Fonctionnement d'un assistant vocal . . . 1
1.2. Performances de modèles à l'état de l'art depuis 2015 . . . 3
1.3. Composition d'un jeu de données . . . 4
1.4. Exemple minimal de wFST . . . 7

2.1. Fenêtre de Hamming . . . 12
2.2. Représentation d'un signal de parole, de son spectrogramme ainsi que les coefficients *log mel-filterbanks* et MFCC . . . 13
2.3. Étapes de calcul des *log mel-filterbanks* et des MFCC. . . . 14
2.4. Cycle complet de la Reconnaissance Automatique de la Parole (RAP). . . . 17
2.5. Exemple de wFSA . . . 18
2.6. Exemple de wFST . . . 18
2.7. Exemple de wFST avec des transitions epsilon. . . . 19
2.8. Union de deux wFSA . . . 20
2.9. Fermeture de Kleene . . . 21
2.10. Intersection de deux wFSA . . . 22
2.11. Composition de deux wFST . . . 23
2.12. Minimisation d'un FSA . . . 24
2.13. Calcul de la distance totale d'un wFSA . . . 25
2.14. Fonctionnement général d'un réseau de neurones profond . . . 32
2.15. Composition d'une couche d'un réseau *Gated ConvNet*. . . . 33
2.16. Composition d'une couche Transformer. . . . 34
2.17. Système d'alignement utilisé pour CTC. . . . 36
2.18. Système d'alignement utilisé pour ASG. . . . 38

3.1. Exemples d'alignements entre transcriptions. . . . 51
3.2. Comparaison visuelle de l'efficacité de l'algorithme dynamique . . . 57
3.3. Alignement des termes du modèle de bruit sur les trames . . . 61
3.4. Exemple d'application de Lead2Gold . . . 66
3.5. Évaluation du LER au cours de l'apprentissage . . . 79
3.6. Évolution de $R(t)$ . . . 82

4.1. Exemple des wFSA A_{ASG} et A_{CTC} . . . 87
4.2. Exemple d'un wFSA B . . . 87
4.3. Exemple d'un wFSA G_{full} . . . 89
4.4. Exemple des wFSA Y_{R1} et Y . . . 89
4.5. Les wFST MAP_{ASG} et MAP_{CTC} . . . 90

4.6. Les wFST $\text{MAP}_{\text{ASG}} \circ Y_{\text{R1}}$ et $\text{MAP}_{\text{CTC}} \circ Y$ . . . 91
4.7. Les wFST G_{ASG} et G_{CTC} . . . 92
4.8. Exemple de gradient sur CTC . . . 92
4.9. Exemple de modèle de bruit de transcription N . . . 93
4.10. Exemple d'utilisation du modèle de bruit de transcription . . . 93
4.11. Modélisation du symbole **R1** avec le wFST R . . . 94
4.12. Exemple d'utilisation du wFST R . . . 94
4.13. Alignement des hypothèses Y^* . . . 95
4.14. Exemple de wFST $G_{\text{L2G}}^{\text{ASG}}$ . . . 96
4.15. Calcul des gradients de $L2G_{\text{ASG}}$ . . . 96
4.16. Exemple de wFST $G_{\text{L2G}}^{\text{CTC}}$ . . . 96
4.17. Exemple de wFST $D_{A \leftrightarrow N}$ . . . 97
4.18. Les deux étapes de construction du lexique L . . . 99
4.19. Les différentes structures de wFST N . . . 101

5.1. Distribution des durées des fichiers audio . . . 115
5.2. Capture d'écran de la tâche de test et des consignes . . . 117
5.3. Captures d'écran de la tâche de transcription . . . 118
5.4. Distribution du nombre d'heures de parole transcrite . . . 121
5.5. Distribution du LER pour l'ensemble des transcriptions collectées . . . 122
5.6. Distribution du LER par travailleur . . . 123
5.7. LER atteint par le modèle acoustique pour des jeux de données de différentes qualité. . . . 128
5.8. LER atteint par le modèle acoustique en filtrant les travailleurs. . . . 129

B.1. Exemple de calcul de gradient sur des wFST . . . 141

C.1. Observation de $R(t)$ . . . 143

Liste des tableaux

2.1. Exemples de demi-anneaux courants 19

3.1. Le modèle de bruit de transcription « M20 f1 » 74
3.2. Exemple d'utilisation des modèles de bruit 76
3.3. Exemple des 10 meilleures hypothèses Y^* obtenues 77
3.4. WER atteint par Lead2Gold en activant uniquement les substitutions . . 80
3.5. WER atteint par Lead2Gold dans le cas général 80

4.1. WER obtenu pour la création des jeux de données bruitées. 104
4.2. Exemple de transcription obtenue après décodage. 105
4.3. WER obtenus pour la première expérience 107
4.4. WER obtenus pour la seconde expérience 107
4.5. Résultats d'apprentissage de modèles de bruit pour différentes tailles de dictionnaires. 109
4.6. Exemple des tailles des wFST communs 110
4.7. Exemple des tailles des wFST après les compositions 110
4.8. Coût de calcul des opérations de composition avec GTN et K2 111

5.1. Erreurs les plus communes des transcriptions collectées 125
5.2. Exemples de transcriptions fournies par les travailleurs 127

Liste des algorithmes

1. Algorithme dynamique pour le calcul de Z, S_{ASG} ou S_{CTC} 57
2. Construction du graphe $\hat{G}_{\text{L2G}}$. 63
3. Algorithme Lead2Gold complet pour le calcul de $\hat{S}_{\text{L2G}}$. 65
4. Calcul des dérivées partielles pour Lead2Gold 72

Liste des acronymes

RAP Reconnaissance Automatique de la Parole - Transforme la parole d'un locuteur en texte.

IA Intelligence Artificielle - Regroupe l'ensemble des algorithmes qui ont pour but de simuler les compétences humaines.

WER *Word Error Rate* - Taux d'erreur de mots en français. C'est une mesure de performance des systèmes de RAP.

LER *Letter Error Rate* - Taux d'erreur de lettres en français. C'est une mesure de performance des systèmes de RAP.

wFST *weighted Finite State Transducer* - Transducteur pondéré à états finis en français. C'est un outil qui permet de transformer une suite de caractères en une autre suite de caractères.

wFSA *weighted Finite State Acceptor* - Accepteur pondéré à états finis en français. Ce modèle indique si une série de caractères est reconnue.

DNN *Deep Neural Network* - Réseau de neurones profond en français. Fonction mathématique composée de fonctions élémentaires paramétriques (neurones) organisées par couches. L'estimation de ces paramètres à partir d'un ensemble de données est appelée l'étape d'apprentissage.

GPU *Graphics Processing Unit* - Processeur graphique en français. Permet d'effectuer un grand nombre d'opérations à la seconde. Inventé à la base pour les jeux-vidéos, c'est un élément indispensable à l'apprentissage rapide des réseaux de neurones profonds.

CTC *Connectionist temporal classification* - Fonction de coût pour l'apprentissage du modèle acoustique. Elle est adaptée aux données séquentielles et ne nécessite pas d'alignement préalable. Chaque probabilité est normalisée au niveau d'une trame.

ASG *Auto Segmentation Criterion* - Fonction de coût similaire à CTC. Les probabilités sont normalisées au niveau de la séquence.

AMT *Amazon Mechanical Turk* - Plateforme de *crowdsourcing* qui permet de proposer des tâches rémunérées qui sont effectuées par des humains. Elle est souvent utilisée pour l'annotation de données.

MFCC *Mel Frequency Cepstral Coefficient* - Méthode d'extraction de caractéristiques d'un signal audio.

HMM *Hidden Markov Model* - Modèle de Markov caché.

GMM *Gaussian Mixture Model* - Modèle de mélange gaussien.

ANN *Artificial neural network* - Réseau de neurones artificiel.

TDNN *Time-delay neural networks* - Réseau de neurones artificiel pour la modélisation de séquences.

RNN *Recurrent neural networks* - Réseau de neurones récurrent.

GLU *Gated Linear Unit* - Fonction d'activation utilisée dans certaines couches de réseaux de neurones profonds.

ReLU *Rectified Linear Unit* - Fonction d'activation utilisée dans certaines couches de réseaux de neurones profonds.

SGD *Stochastic Gradient Descent* - Algorithme d'optimisation permettant d'obtenir une approximation du gradient de la fonction de coût en un temps de calcul réduit.

1. Introduction

1.1. Motivation et cadre

Applications de la reconnaissance vocale. Parler est l'une des premières compétences que l'on apprend et c'est le moyen le plus naturel de transmettre une information à quelqu'un d'autre. Cependant, les premières machines ont été conçues pour interpréter du code informatique et non du texte et encore moins de la parole. Alan Turing a proposé en 1950 le test de Turing [Turing, 2009] pour déterminer si une machine était capable d'imiter une conversation humaine. Il a prédit qu'un jour elles pourraient si bien le faire qu'il serait impossible de deviner si l'on communique avec une machine ou un autre humain. En sommes-nous à ce stade ? En 2011, l'entreprise Apple présentait au monde le premier assistant vocal conçu pour être utilisé par le grand public. Le fonctionnement d'un tel assistant est schématisé dans la Fig. 1.1 [Young, 2021]. La voix de l'utilisateur est captée par un microphone puis l'outil de RAP intégré transforme cette onde sonore en texte. Ce texte est alors analysé par un modèle de compréhension et une réponse est donnée à l'utilisateur. Cette dernière étape d'interprétation du texte n'a cessé d'être améliorée ces 10 dernières années avec des modèles toujours plus évolués de compréhension du langage naturel. C'est une sous-branche de l'Intelligence Artificielle (IA) qui a pour but de modéliser la communication humaine.

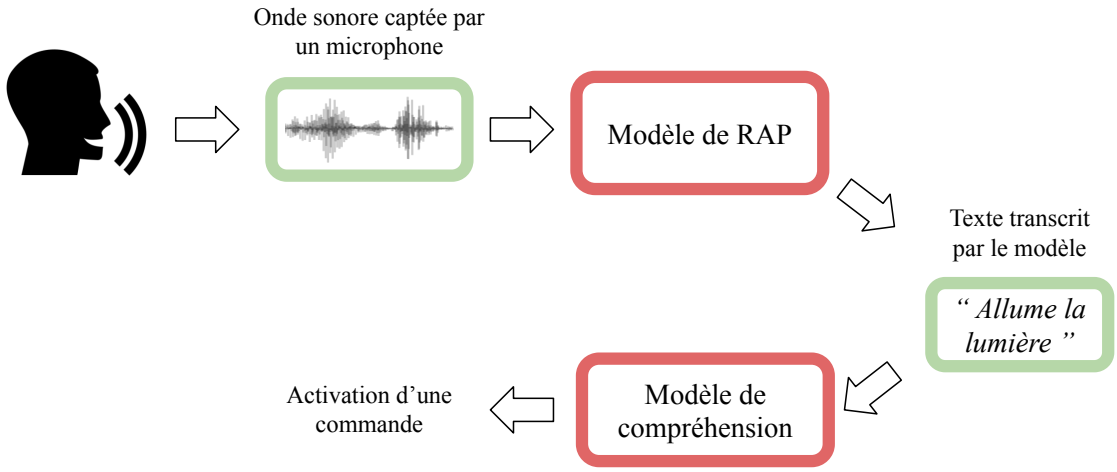


FIGURE 1.1. – Fonctionnement d'un assistant vocal. En rouge, les différents modèles qui utilisent les modalités en vert.

Qu'en est-il du premier bloc technologique, celui de RAP ? Nous avons également pu observer des progrès exemplaires dans ce domaine. Comme nous venons de le voir avec l'assistant vocal, la RAP permet des applications concrètes. On peut notamment citer la dictée vocale pour écrire des documents par la parole. Les plateformes vidéos comme YouTube ou Facebook ont également de grands intérêts pour le développement de la RAP. Le sous-titrage automatique de vidéos est utilisé pour regarder un contenu sans le son dans un environnement où il faut être silencieux, comme dans les transports en commun. Ces entreprises augmentent ainsi le temps passé par utilisateur sur la plateforme ce qui a un impact direct sur leurs revenus publicitaires. De plus, avec un nombre toujours croissant d'utilisateurs, et donc du nombre de critiques à leur encontre, ces plateformes se doivent d'être exemplaires et ne proposer que du contenu approprié à ses utilisateurs. C'est donc dans le domaine de la censure que l'on retrouve aussi cette technologie. Le son de chaque vidéo est analysé et s'il est jugé inapproprié, la vidéo doit être retirée très rapidement. Avec toutes ces applications, la RAP est de plus en plus utilisée et donc l'effort consacré à la recherche dans ce domaine est devenu plus important.

Des records de performance. Ces cinq dernières années les performances obtenues sur les jeux de données traditionnels de recherche se sont considérablement améliorées, voir Fig. 1.2. Mais alors, le problème de la RAP est-il résolu ? Que reste-t-il à faire ? La machine est-elle capable de transcrire la parole comme un humain ? Si l'on regarde uniquement les métriques obtenues sur les jeux de données classiques en anglais [Xiong et al., 2018], on pourrait penser que oui. Si l'on se fie exclusivement à cette communication tapageuse de 2014 où l'Université anglaise de Reading affirme que le test de Turing est résolu [Warwick and Shah, 2016], on pourrait se persuader que c'est le cas. Hélas, la réalité est bien plus contrastée. Concernant le test de Turing, on peut émettre une objection : il n'a jamais constitué une bonne mesure de performance des modèles d'IA.

Enfin, lors de l'évaluation des modèles de RAP il ne faut pas oublier que ces jeux de données ne représentent pas les conditions réelles, et que donc les performances reportées doivent être interprétées avec précaution. Le jeu de données Librispeech [Panayotov et al., 2015] est par exemple composé de textes lus en anglais. Il y a peu de bruits parasites et les sons sont bien articulés. Szymański et al. [2020] argumentent également que les systèmes modernes de RAP sont incapables de transcrire de façon satisfaisante une conversation spontanée entre deux personnes.

La reconnaissance vocale en conditions réelles. Pour qu'un modèle de RAP fonctionne bien, il faut qu'il réunisse plusieurs critères qui ne sont souvent pas satisfaits lors de son utilisation en conditions réelles. Il faut éviter les bruits parasites comme le vent ou l'écho, avoir un micro de bonne qualité et il faut que les données utilisées lors de l'apprentissage du modèle soient similaires aux conditions réelles d'utilisation. Ce dernier point pose beaucoup de problèmes. Il suffit qu'un accent soit un peu prononcé et les performances chutent immédiatement [Ghorbani and Hansen, 2018]. Un modèle appris sur un anglais sans accent fonctionnera moins bien avec un utilisateur ayant un accent indien [Vergyri

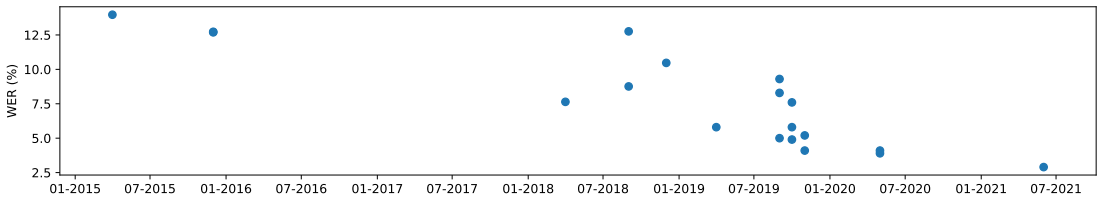


FIGURE 1.2. – Performance de modèles à l'état de l'art depuis 2015. Nous reportons ici le taux d'erreur sur les mots *Word Error Rate* (WER) de ces modèles sur le jeu de données Librispeech *test-other* (plus celui-ci est faible, meilleur est le modèle). Les données sont agrégées par Synnaeve [2021].

et al., 2010]. De même si l'on veut retranscrire de façon fidèle une conversation complexe, il faudra que le jeu de données contienne des annotations de haute qualité : tous les détails qui créent une conversation animée comme l'hésitation, le rire, couper des mots ou interrompre l'autre locuteur doivent être indiqués dans le jeu de données.

Pour que tout le monde ait accès à un bon modèle de RAP, il faut idéalement en concevoir un différent pour chaque accent. Mais alors le problème est-il résolu ? Non, pour qu'un modèle soit performant il faut posséder une grande quantité de données de qualité. La limitation est alors beaucoup plus pragmatique : créer ces jeux de données coûte très cher. L'accès à cette technologie est donc inégal. Ce sont surtout les grandes entreprises vendant ce service qui vont supporter les coûts de création des jeux de données. Actuellement il existe de bons modèles pour les langues les plus parlées telles que l'anglais, l'espagnol ou même le français. Le service de RAP de Google propose une centaine de langues [1], mais cela reste très minoritaire par rapport aux 7 000 langues parlées actuellement dans le monde [Eberhard et al., 2021].

1.2. Problématique

Les paradigmes d'apprentissage. Les progrès observés ces dernières années en RAP sont en partie dus à la maturité des technologies d'apprentissage automatique [Hannun et al., 2014, Xiong et al., 2017a], à l'augmentation des ressources informatiques, mais aussi à la disponibilité de très grands jeux de données annotés. L'annotation consiste à retranscrire chaque phrase par une séquence de mots, ce qui prend beaucoup de temps et qui est très coûteux. L'apprentissage supervisé permet d'utiliser efficacement de telles annotations. D'autres approches, illustrées dans la Fig. 1.3, permettent de changer de paradigme. L'apprentissage semi-supervisé utilise à la fois des données annotées et non annotées, ce qui peut apporter un gain de performance sans effort d'annotation supplémentaire. Ces approches considèrent qu'il n'y a aucune erreur dans les annotations, ce qui est faux en pratique. La tendance récente est de toujours utiliser plus de données, si

1. https://cloud.google.com/blog/products/ai-machine-learning/new-features-models-and-languages-for-speech-to-text

bien que dans la plupart des travaux, le postulat est que les annotations sont exactes, et que même si elles ne le sont pas, les erreurs sont noyées dans la masse de données. Elles n'auront alors que peu d'influence sur les performances générales du modèle.

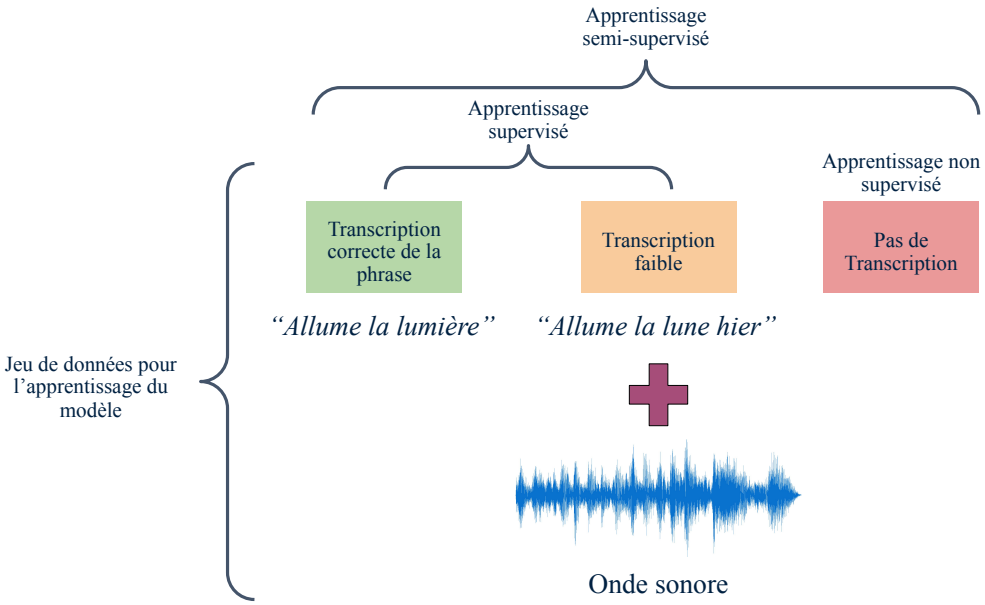


FIGURE 1.3. – Composition d'un jeu de données pour l'apprentissage d'un modèle de RAP. Selon la composition des annotations, différents paradigmes d'apprentissage sont possibles.

Les annotations faibles. Dans cette thèse, nous souhaitons aller à contre-pied de la tendance qui consiste à utiliser toujours plus de données. Nous traitons le problème des annotations faibles, c'est-à-dire qui contiennent des erreurs ou qui ne fournissent qu'une information partielle. L'apprentissage faiblement supervisé exploite ces annotations faibles. Les annotations faibles peuvent être de natures différentes. Nous listons ci-dessous les différentes natures ainsi que les contextes dans lesquels elles sont obtenues :

1. les annotations erronées – les annotateurs doivent travailler vite et peuvent faire des erreurs sur certains mots ;
2. les annotations générées automatiquement – par un modèle de RAP existant ;
3. les annotations incomplètes – les annotateurs peuvent refuser d'annoter une portion de la phrase car celle-ci est peut-être inintelligible ;
4. les annotations au niveau de la phrase – les annotateurs peuvent être amenés à vérifier des transcriptions automatiques et indiquer celles qui sont globalement correctes sans être une transcription exacte ;

5. les annotations implicites – le comportement des utilisateurs peut informer indirectement si une transcription proposée par un système de RAP d'un assistant vocal est bonne ou non : si l'utilisateur demande une nouvelle transcription, c'est que la première était probablement erronée ;
6. les annotations transformées – les sous-titres de vidéos ne constituent pas une transcription exacte des mots prononcés, mais une version résumée qui peut être utilisée sans coût supplémentaire pour créer un modèle de RAP.

Ce sont surtout les deux premiers points que nous traiterons. Le but principal de cette thèse est de réduire les coûts pour construire un système de RAP. Plus précisément, il s'agit d'explorer des approches d'apprentissage faiblement supervisé qui exploitent plus d'informations que l'apprentissage supervisé ou semi-supervisé, et cela sans coût supplémentaire d'annotation. Ces informations supplémentaires viennent des erreurs de transcription que nous prenons en compte. Nous travaillons toujours avec des annotations, mais celles-ci ne sont pas parfaites. Nous voulons réduire les pertes de performances liées à ces imperfections. En contrepartie, il y a un coût de calcul supplémentaire. Le défi est alors de bien concevoir la méthode pour que ce coût soit acceptable.

En prenant en compte les erreurs de transcriptions, nous nous autorisons ainsi à collecter des données de moins bonne qualité et qui sont donc en moyenne moins chères à produire. Elles pourront être obtenues de manière détournée, ce qui ne nécessitera pas de faire annoter des milliers d'heures de parole par des professionnels :

— Nous créons des jeux de données synthétiques qui reproduisent les erreurs d'un modèle de RAP pour évaluer nos algorithmes.

— Avec une plateforme de *crowdsourcing*, nous nous autorisons à collecter des annotations de plus faible qualité via des non-professionnels.

Modéliser l'annotateur. La nature et la quantité d'erreurs dans les annotations ainsi que les raisons pour lesquelles il y a des erreurs dépendent de chaque annotateur. Certains sont peut-être meilleurs que d'autres, mais même à niveau égal les annotateurs peuvent ne pas faire les mêmes erreurs. Pouvons-nous caractériser ces comportements pour mieux comprendre les erreurs qu'il peut y avoir ? En fonction de l'annotateur, nous pourrions accorder des niveaux de confiance différents aux annotations produites. Les erreurs systématiques faites par un annotateur peuvent alors être automatiquement corrigées et ainsi nous permettre de faire un apprentissage plus fin des modèles de RAP.

1.3. Les outils utilisés

La thèse se focalise sur la reconnaissance vocale et sur la modélisation d'erreurs de transcription. Nous proposons de nouvelles méthodes qui combinent l'apprentissage profond avec des algorithmes de recherche en faisceau (*beam search*). La recherche en faisceau

sera implémentée de différentes manières, une d'entre elles utilisant un transducteur pondéré à états finis.

L'apprentissage automatique est un ensemble d'algorithmes qui permettent de résoudre une tâche à partir de l'observation d'un jeu de données. Pour cela, nous utilisons un modèle composé de paramètres. La phase d'observation est appelée phase d'apprentissage ou d'entraînement et elle consiste à estimer tous les paramètres du modèle. Nous pouvons ensuite utiliser le modèle pour résoudre la tâche sur des données jamais vues auparavant, par exemple un discours que l'on doit retranscrire ou un objet que l'on doit reconnaître sur une image. Parmi ces algorithmes, on retrouve la régression logistique [Berkson, 1944], les forêts aléatoires [Breiman, 2001] ou encore les machines à vecteurs de support [Vapnik, 2013].

L'apprentissage profond est une classe d'algorithmes d'apprentissage utilisant un réseau de neurones profonds ou *Deep Neural Network* (DNN) [Deng and Yu, 2014, Goodfellow et al., 2016] comme modèle. La différence avec les modèles classiques est que les DNN sont composés d'un très grand nombre de paramètres (ou neurones), jusqu'à plusieurs millions voire plusieurs milliards [Fedus et al., 2021]. Le principe fondamental de ces algorithmes existe depuis longtemps [Rosenblatt, 1958] mais ce n'est que récemment que cette technologie a pu montrer son plein potentiel [Krizhevsky et al., 2012], grâce notamment aux grandes quantités de données disponibles et à la montée en puissance des ordinateurs et surtout des processeurs graphiques *Graphics Processing Unit* (GPU). Les neurones sont souvent organisés en couches successives et un réseau sera d'autant plus profond qu'il y aura de couches. La complexité de ces nouveaux modèles ainsi que les algorithmes d'apprentissage associés permettent une modélisation beaucoup plus fine de la tâche à résoudre, mais cela se fait au détriment de l'interprétabilité des paramètres. L'apprentissage du modèle se fait sur le principe de la descente du gradient et les gradients se calculent avec l'algorithme de rétro-propagation du gradient. Les modèles à l'état de l'art de nombreux domaines utilisent de tels algorithmes. Ils sont aussi bien utilisés pour la classification d'images [Touvron et al., 2020], la détection d'objets [Bochkovskiy et al., 2020] et la RAP [Hsu et al., 2021] que pour la recommandation [Ma et al., 2020b] ou encore l'analyse de textes [Devlin et al., 2019].

Le transducteur pondéré à états finis ou *weighted Finite State Transducer* (wFST) est un modèle qui indique comment transformer une série de symboles en entrée en une autre série de symboles en sortie. À chaque étape de transition d'un symbole à un autre est associée un poids. Un exemple est donné dans la Fig. 1.4. Les wFST peuvent être utilisés pour représenter un modèle probabiliste. En reconnaissance vocale les wFST ont longtemps servi pour effectuer la recherche en faisceau afin de décoder un signal d'entrée. L'outil Kaldi [Povey et al., 2011] implémente cette technique. Usuellement, les wFST ne peuvent pas être utilisés dans le cadre d'un apprentissage de bout en bout car ils ne sont pas différentiables. Mais de nouveaux outils comme k2 [Povey et al., 2020] ou GTN [Hannun et al., 2020] permettent de calculer le gradient des poids et de les propager au

DNN à apprendre. L'avantage d'utiliser les wFST est que l'on a alors accès à toute une gamme d'algorithmes très utiles pour les manipuler. On peut composer, concaténer ou encore minimiser les wFST. De plus, certaines implémentations utilisent les ressources du GPU ce qui accélère grandement les calculs.

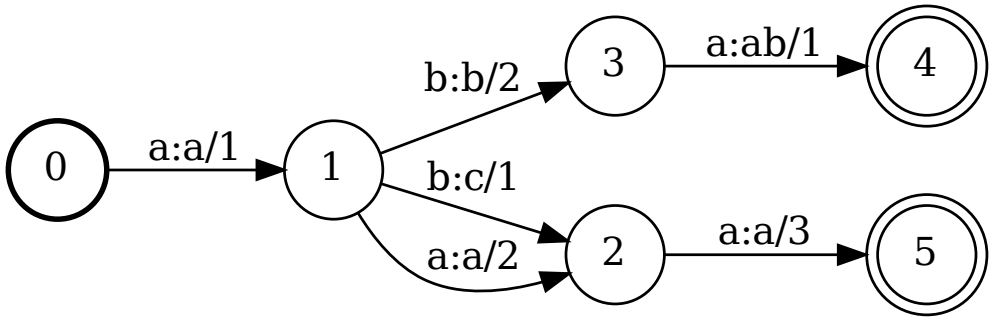


FIGURE 1.4. – Exemple minimal de wFST. Il y a l'état initial **0** à partir duquel part l'ensemble des séquences. Ces séquences se finissent aux états finaux **4** et **5**. La transition $b : c/1$ indique que le symbole **b** est transformé en **c** avec un poids de 1. Ici la séquence $[a, a, a]$ est reconnue en entrée en suivant les états [**0**, **1**, **2**, **5**] et reste identique en sortie avec un poids cumulé de 6. La séquence $[a, b, a]$ est aussi reconnue et peut être transformée en $[a, b, ab]$ en suivant les états [**0**, **1**, **3**, **4**] avec un poids cumulé de 4 ou en $[a, c, a]$ en suivant les états [**0**, **1**, **2**, **5**] avec un poids cumulé de 5.

Une plateforme de *crowdsourcing* permet de mettre en relation des travailleurs avec des personnes proposant des tâches à effectuer. En particulier nous utilisons *Amazon Mechanical Turk* (AMT) qui a été créée par Amazon en 2005. Ces tâches n'exigent souvent pas de qualifications particulières et consistent la plupart du temps à annoter des données. Le travailleur pourra être amené à devoir traduire du texte, modérer du contenu ou, dans le cas qui nous intéresse, transcrire des fichiers audio. Avec cette méthode, il est possible de construire des jeux de données utilisables pour l'apprentissage automatique. La qualité moyenne du travail effectué est moins bonne qu'en utilisant les services d'une entreprise spécialisée car les travailleurs ne sont pas spécialement qualifiés pour la transcription.

1.4. Contributions et plan du document

Nous apportons trois contributions à cette thèse. La première est l'incorporation d'une recherche en faisceau dans un apprentissage profond de bout en bout. Cette recherche en faisceau parcourt la transcription disponible, qui peut comporter des erreurs, et sélectionne les corrections les plus probables. La seconde est une évolution de la première méthode. Elle modélise la recherche en faisceau à l'aide de wFST. Le résultat est beaucoup plus modulaire car il accepte d'autres sources d'information et des types de corrections

de transcription moins contraignantes. La dernière est la prise en compte des spécificités de chaque annotateur. Pour ce dernier point nous avons besoin de données particulières, pour cela nous mettons en place un processus de collecte de données avec une plateforme de *crowdsourcing*.

Le chapitre 2 présente les méthodes d'apprentissage de réseaux de neurones profonds. Plus particulièrement, il décrit les spécificités liées à la RAP. Nous y retrouvons notamment l'étape de décodage du signal d'entrée avec une recherche en faisceau. La recherche en faisceau utilise le modèle de RAP et ajoute d'autres informations annexes en utilisant un lexique et un modèle de langage. Ce chapitre expose certaines des structures des DNN mais explique de façon plus détaillée la phase d'apprentissage de bout en bout, notamment les étapes *forward* et *backward*. Il définit les wFST et les algorithmes associés utiles à la RAP et à notre approche. Ce chapitre présente également les méthodes liées à la prise en compte des erreurs dans les données. Il met en évidence la difficulté à prendre en compte les erreurs dans le cas de données séquentielles comme la parole. Enfin il caractérise le processus de collecte de données.

Le chapitre 3 présente notre première contribution, l'algorithme Lead2Gold [Dufraux et al., 2019]. Le but est d'utiliser un algorithme de recherche en faisceau pour intégrer la modélisation des erreurs de transcription. Cet algorithme se distingue de l'usage habituel de la recherche en faisceau dans la phase de décodage. Contrairement à son utilisation classique, notre méthode incorpore une recherche en faisceau directement pour l'apprentissage du modèle de RAP. Comme information annexe, nous n'utilisons pas un modèle de langage mais un modèle d'erreurs de transcription. La combinaison des probabilités acoustiques et des probabilités d'erreurs permet alors d'effectuer une recherche en faisceau pour sélectionner les transcriptions candidates les plus probables. Ce chapitre ne considère que des transformations simples. Nous prenons seulement en compte la possibilité qu'une lettre soit confondue avec une autre. Les données utilisées y sont également synthétiques, c'est-à-dire que nous ajoutons volontairement des erreurs dans un jeu de données. Cela permet de prendre un recul nécessaire pour évaluer la méthode. Cette méthode adapte les fonctions de coûts classiques *Connectionist temporal classification* (CTC) [Graves et al., 2006] et *Auto Segmentation Criterion* (ASG) [Collobert et al., 2016]. Ce chapitre démontre aussi la validité mathématique de cette méthode qui intègre un modèle d'erreur de transcription.

Le chapitre 4 adapte l'algorithme Lead2gold pour le rendre plus modulaire. Pour cela l'étape de recherche en faisceau est entièrement implémentée à l'aide de wFST. Le modèle d'erreur de transcription est construit avec l'aide de cet outil. Celui-ci peut aussi être plus général et accepter d'autres types de transformation. Ce chapitre explique comment l'algorithme de composition de wFST permet de reproduire la fonction de coût de Lead2Gold. Avec peu d'efforts, nous pouvons aussi ajouter d'autres sources d'informations complémentaires pour caractériser les transcriptions possibles. Ainsi, prendre en

compte un lexique ou un modèle de langage est possible simplement en ajoutant une opération de composition de wFST.

Le chapitre 5 est consacré à la collecte de données en conditions réelles ainsi qu'à la prise en compte des spécificités de chaque annotateur. Nous utilisons la plateforme AMT pour collecter des transcriptions de qualité non-professionnelle. Nous plaçons les annotateurs dans des conditions difficiles en limitant le temps alloué pour produire chaque transcription et en limitant le nombre d'écoutes possibles. Ainsi nous pouvons reproduire une collecte de données en condition réelle avec une qualité non-professionnelle. Le but est de démontrer la faisabilité de construire un modèle de RAP dans une nouvelle langue avec peu de ressources disponibles. Nous observons également l'impact des erreurs dans les transcriptions sur les performances d'un modèle de RAP.

Le chapitre 6 apporte une conclusion et résume les différents aspects du manuscrit. Nous y proposons aussi des pistes possibles pour continuer le travail de recherche.

1.5. Publication associée à cette thèse

— Dufraux, A., Vincent, E., Hannun, A., Brun, A., and Douze, M. (2019). Lead2gold : Towards exploiting the full potential of noisy transcriptions for speech recognition. In 2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), pages 78-85.

2. État de l'art

La transformation de la parole en texte se fait en plusieurs étapes : l'acquisition et la transformation du signal, l'apprentissage du modèle acoustique, la création d'un lexique et d'un modèle de langage et enfin le décodage final du signal permettant de combiner tous ces éléments. Dans ce chapitre nous détaillons le fonctionnement et l'état de l'art pour toutes ces étapes. Nous détaillerons le fonctionnement des *weighted Finite State Transducer* (wFST) ainsi que les algorithmes utilisant cet outil qui sont utiles pour notre méthode et plus généralement pour la RAP. Nous expliquerons également les méthodes permettant d'utiliser les étiquettes bruitées.

2.1. Du signal au texte

Le signal acoustique et le texte sont des données séquentielles. Dans cette partie nous définissons ce qu'est une donnée séquentielle et nous formulons le problème général de l'annotation de séquences. Nous prenons le point de vue de la RAP et nous expliquons comment les différentes étapes doivent se combiner pour transformer la parole en texte.

2.1.1. Représentation d'une donnée séquentielle

Une donnée séquentielle est composée d'observations ordonnées. Ce nombre d'observations est variable et est noté T. La position d'une l'observation dans la séquence est indexée par la variable t. La séquence se note

$$O = (o_1\ o_2\ ...\ o_t\ ...\ o_{T-1}\ o_T). \tag{2.1}$$

Les observations discrètes. La séquence d'observations peut être discrète et dans ce cas, les observations appartiennent à un ensemble fini de symboles : $o_t \in \{v_1, v_2, ..., v_M\}$. Par exemple l'observation peut être une phrase composée d'un ensemble de mots, les mots appartenant à un lexique de taille finie M.

Les observations continues. Lorsqu'elles sont continues, les observations sont composées de N mesures à valeur réelle. On a donc $o_t \in \mathbb{R}^N$ et

$$O = \begin{pmatrix} o_{11} & & o_{t1} & & o_{T1} \\ \vdots & \dots & \vdots & \dots & \vdots \\ o_{1N} & & o_{tN} & & o_{TN} \end{pmatrix} \in \mathbb{R}^{N \times T}. \tag{2.2}$$

Nous pouvons par exemple représenter un signal acoustique brut de l secondes dans le domaine temporel en faisant 16 000 mesures par seconde de l'intensité (fréquence de 16 kHz) avec $N = 1$ et $T = 16000 \times l$.

2.1.2. Transformation de l'observation dans le cas d'un signal acoustique

La représentation temporelle brute d'un signal acoustique est peu pratique à manipuler au vu du grand nombre de mesures nécessaires pour décrire le signal et du peu d'informations contenues dans les mesures prises individuellement. Certains travaux l'utilisent cependant directement [Jaitly and Hinton, 2011, Tjandra et al., 2017, Zeghidour, 2019]. Mais la méthode usuelle est de calculer des descripteurs qui contiennent seulement les informations les plus utiles pour l'analyse du signal. Ces descripteurs sont souvent obtenus à partir du spectre du signal, mais incluent parfois aussi la hauteur ou le voisement [Ghahremani et al., 2014, Zolnay et al., 2003].

Pour obtenir le spectre, il faut d'abord appliquer une fenêtre glissante sur le signal d'origine. Cette fenêtre est définie par sa forme, sa taille et son pas de déplacement. Pour les expériences de ce manuscrit nous faisons le choix classique d'utiliser une fenêtre de Hamming de 25 millisecondes avec un pas de 10 millisecondes. Celle-ci permet d'éviter des effets de bords néfastes en appliquant un masque aux échantillons du signal selon leur position dans la fenêtre. La fenêtre de Hamming est représentée Fig. 2.1 et elle a pour expression :

$$\text{Hamming}(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N_\text{f} - 1}\right), \tag{2.3}$$

avec N_f la taille de la fenêtre et n une position particulière dans cette fenêtre.

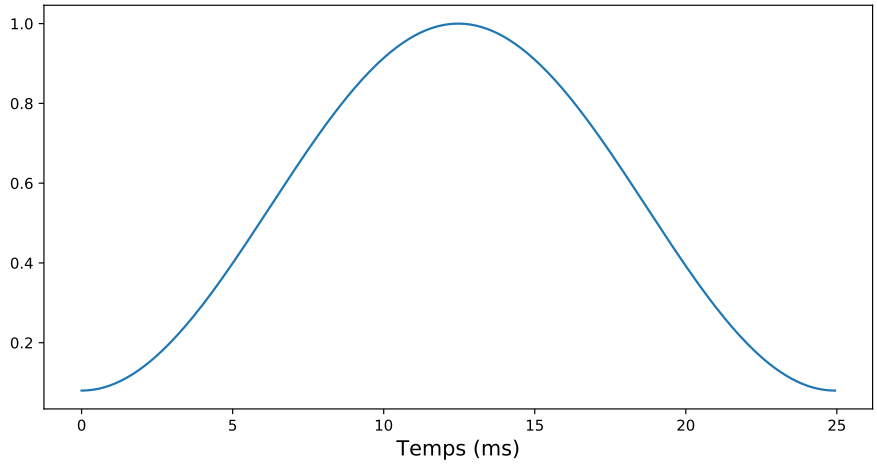


FIGURE 2.1. – Fenêtre de Hamming utilisée dans ce manuscrit.

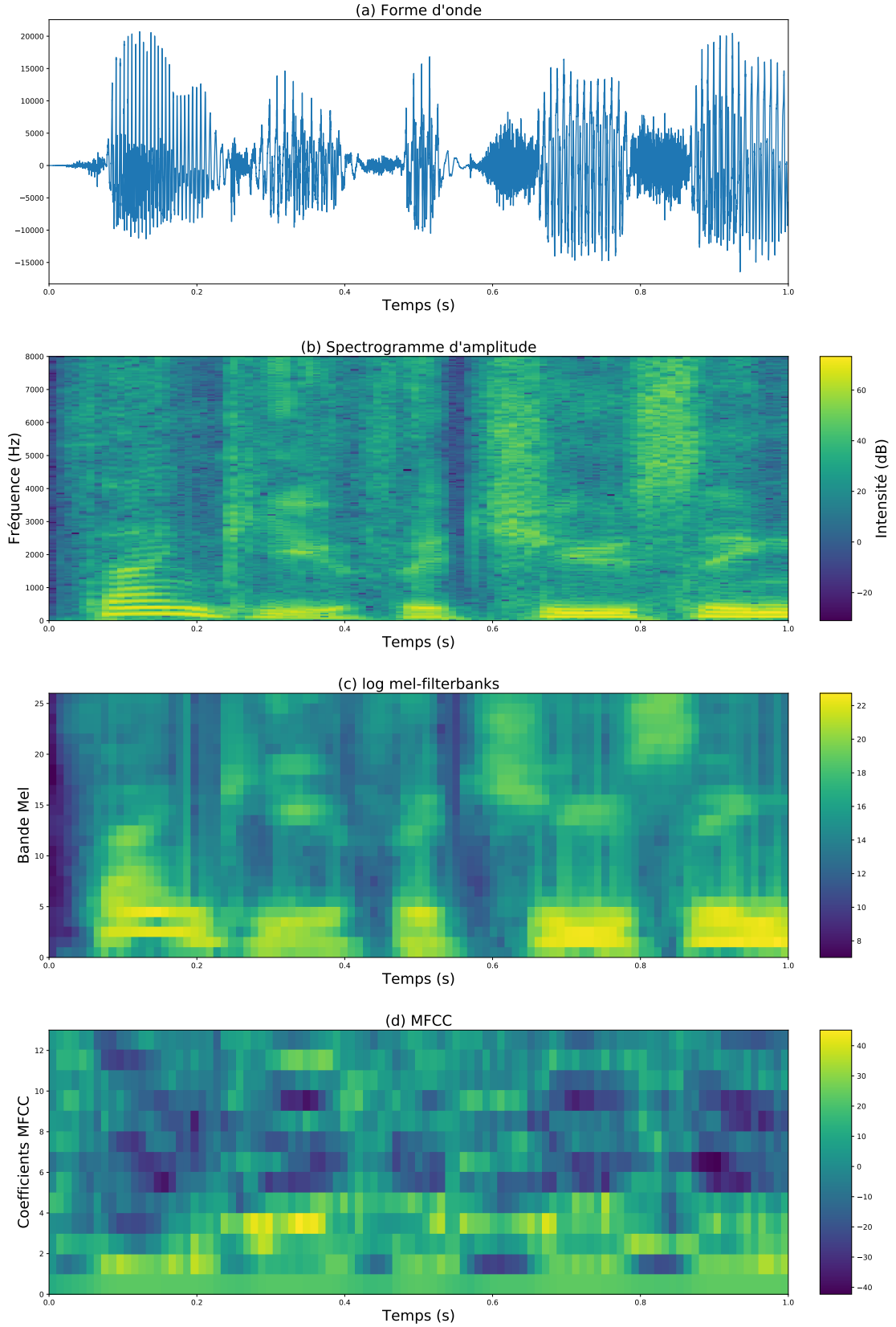


FIGURE 2.2. – Représentation d'un signal de parole (a), de son spectrogramme d'amplitude (b) ainsi que les 26 premiers coefficients *log mel-filterbanks* (c) et les 13 premiers coefficients MFCC (d). La phrase «*Thank you for choosing*» est prononcée.

Chaque segment obtenu après application de la fenêtre est appelé « trame ». Il faut ensuite calculer la transformée de Fourier discrète du signal fenêtré avec l'algorithme de la transformée de Fourier rapide (FFT, *Fast Fourier transform*). Pour chaque trame, nous obtenons une représentation temps-fréquence appelée spectre. L'ensemble des spectres forme le spectrogramme du signal. Cette méthode produit un spectrogramme à valeurs complexes. On ne garde que le module de chaque coefficient pour obtenir le spectrogramme d'amplitude. La Fig. 2.2 illustre un signal brut (a) et son spectrogramme d'amplitude (b).

Pour la RAP, les descripteurs les plus utilisés sont les spectres de log-amplitude en échelle Mel (*log mel-filterbanks*) ou alors les *Mel Frequency Cepstral Coefficient* (MFCC) [Davis and Mermelstein, 1980]. Même si ces méthodes ont été développées il y a plus de 30 ans, elles sont toujours utilisées aujourd'hui [Gulati et al., 2020].

Le calcul de ces descripteurs s'effectue en plusieurs étapes décrites dans la Fig. 2.3. Le but de ces étapes est de simuler la perception de la hauteur et de la puissance sonore par l'oreille humaine en les modélisant par l'échelle fréquentielle Mel et par une compression logarithmique. Pour obtenir les *log mel-filterbanks*, il faut appliquer un banc de M filtres en échelle Mel au spectrogramme d'amplitude du signal et calculer le logarithme du résultat. Pour calculer les MFCC, il faut ajouter une transformée en cosinus discrète (DCT, *Discrete Cosine Transform*). Pour les expériences de ce manuscrit nous choisissons d'utiliser directement les *log mel-filterbanks* avec 80 filtres. On a $O \in \mathbb{R}^{80 \times T}$, avec T le nombre de trames obtenues après fenêtrage du signal.

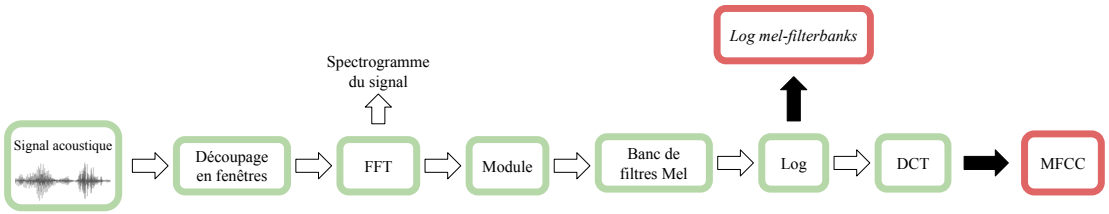


FIGURE 2.3. – Étapes de calcul des *log mel-filterbanks* et des MFCC.

2.1.3. Formulation générale du problème d'étiquetage de séquences

Soit une observation $O \in \mathbb{R}^{N \times T}$, nous voulons trouver la meilleure séquence de symboles $W^* = (w_1 \ ... \ w_L)$ qui peut être associée à cette observation :

$$W^* = \underset{W}{\operatorname{argmax}}\, P(W|O). \tag{2.4}$$

La séquence de symboles W n'a pas forcément la même longueur que la séquence d'observations O. Dans le cas de la RAP, W peut être une séquence de mots, de phonèmes

ou encore de lettres. Il faut trouver la séquence la plus probable selon l'observation acoustique O. Si l'on applique le théorème de Bayes à l'équation (2.4), on obtient :

$$W^* = \underset{W}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} = \underset{W}{\operatorname{argmax}}\, P(O|W)P(W). \tag{2.5}$$

Le terme $P(O)$ peut être enlevé de l'équation (2.5) car il ne dépend pas de W.

— $P(O|W)$ est le modèle d'attache aux données. Pour la RAP c'est le modèle acoustique.

— $P(W)$ décrit comment sont formées les séquences W. Pour la RAP c'est le modèle de langage. Il détermine la probabilité d'existence d'une phrase. Ce modèle peut être vu comme un filtre qui ne va sélectionner que des phrases qui sont vraisemblables.

La solution de l'équation (2.5) s'obtient en explorant un certain nombre de solutions avec un algorithme de Viterbi [Forney, 1973] ou de recherche en faisceau [Lowerre, 1976]. L'algorithme de Viterbi permet d'obtenir la solution optimale du problème en explorant toutes les séquences possibles. Le problème est décomposé en la recherche de sous séquences optimales et le principe de programmation dynamique peut être appliqué [Bellman, 1966]. La complexité est de $\mathcal{O}(TN^2)$ avec N le nombre d'états que l'on peut atteindre à partir d'une sous-séquence. N dépend du modèle de langage et peut devenir très grand, ce qui ne permet pas d'utiliser cet algorithme en pratique. L'algorithme de recherche en faisceau réduit le nombre de possibilités que l'on explore en ne sélectionnant que les K meilleurs états atteignables, ce qui nous donne une complexité de $\mathcal{O}(TK^2)$. On peut alors choisir K pour que le calcul soit faisable.

2.1.4. Du signal à la transcription : les étapes de la reconnaissance automatique de la parole

Comme l'indique la Fig 2.4, pour transcrire la parole d'un locuteur il y a deux étapes principales. Il faut d'abord obtenir le modèle acoustique $P(O|W)$ et le modèle de langage $P(W)$ pendant l'étape d'apprentissage. Cet apprentissage statistique nécessite l'utilisation de jeux de données afin d'estimer les paramètres des modèles. Un lexique doit aussi être fourni avec la liste des mots autorisés. Ensuite vient la phase de décodage. Les trois composants sont utilisés afin de fournir la meilleure transcription possible d'un signal acoustique qui n'a jamais été vu auparavant. Le décodeur est une recherche en faisceau qui va parcourir un grand nombre de transcriptions possibles. Ces transcriptions sont uniquement composées de mots présents dans le lexique. La meilleure transcription est alors choisie selon l'association des probabilités du modèle acoustique et du modèle de langage.

Le modèle acoustique prend en entrée l'observation O et fournit en sortie la probabilité $P(O|W)$ de présence des symboles. Il faut au préalable faire un choix sur ces symboles. Chaque symbole peut représenter un phonème, une lettre ou un morceau de mot. Si

les phonèmes sont utilisés, il faut indiquer dans le lexique l'ensemble des prononciations possibles de chaque mot. Bien que classique, l'utilisation des phonèmes complexifie le problème puisqu'il faut fournir ces prononciations. Dans ce manuscrit nous utilisons comme symboles soit directement des lettres (ou graphèmes) [Killer et al., 2003] soit des morceaux de mots (*word pieces*) [Rao et al., 2017]. Pour l'apprentissage du modèle acoustique le jeu de données doit être annoté : pour chaque phrase prononcée nous avons besoin de la transcription textuelle correspondante.

Pour créer le modèle de langage nous n'avons besoin que de données textuelles. La présence du lexique force le décodeur à ne proposer que des séquences de symboles qui représentent des mots qui existent. Le modèle de langage fournit la probabilité $p(W)$ qu'une phrase soit effectivement prononcée. Il indique par exemple que la phrase « Éteins la lumière » a beaucoup plus de chances d'être prononcée que la phrase « Éteins le livre ».

Dans les parties 2.3 et 2.4 nous détaillons chaque étape de l'apprentissage et du décodage d'un modèle de RAP. La prochaine partie introduit les wFST ainsi que les algorithmes utiles à notre problème.

2.2. Le transducteur pondéré à états finis

Le transducteur pondéré à états finis (wFST) est un modèle permettant de transformer une série de symboles en une autre série de symboles. Les wFST sont utilisés dans différents domaines du traitement du langage [Knight and May, 2009] et pour la RAP [Mohri et al., 2008]. Ils permettent de manipuler des données textuelles avec une représentation commune aux différents composants.

2.2.1. Définitions

Pour bien définir ce qu'est un wFST, il faut d'abord définir ce qu'est un accepteur.

Définition d'un accepteur Un accepteur ou un reconnaisseur pondéré à états finis *weighted Finite State Acceptor* (wFSA) est un modèle qui indique si une entrée est acceptée ou non. Cette entrée n'est pas transformée, le modèle renvoie une sortie binaire égale à 1 ou 0 selon que l'entrée est reconnue ou non. Si l'accepteur est pondéré, on a également accès à un score lorsqu'une entrée est reconnue. La Fig. 2.5 donne un exemple. Formellement un wFSA est un 5-uplet $(\Sigma, Q, Q_\text{i}, Q_\text{f}, \pi)$ composé :

— d'un alphabet Σ. C'est l'ensemble des symboles pouvant être utilisés. Les symboles « pomme » et « rouge » de la Fig. 2.5 sont dans Σ.

— d'un ensemble d'états Q. Les 5 états de la Fig. 2.5 sont numérotés de 0 à 4. Ce sont les nœuds du graphe représentant le modèle.

— d'un ensemble d'états initiaux $Q_\text{i} \subseteq Q$. $Q_\text{i} = \{0, 4\}$ pour la Fig. 2.5.

— d'un ensemble d'états finaux $Q_\text{f} \subseteq Q$. $Q_\text{f} = \{0\}$ pour la Fig. 2.5.

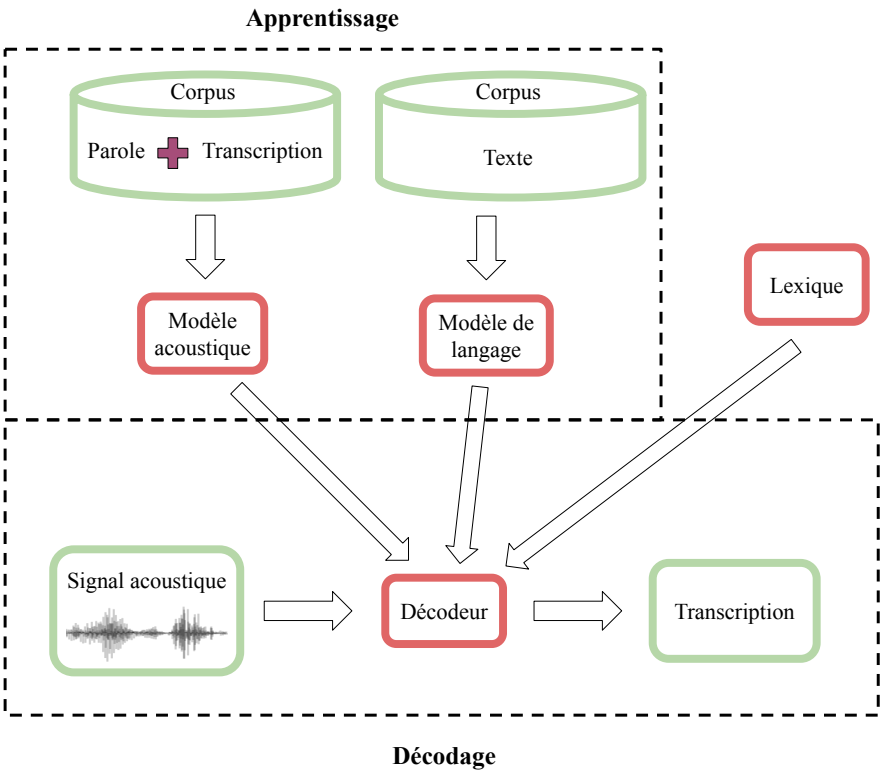


FIGURE 2.4. – Cycle complet de la RAP. Il faut d'abord obtenir le modèle acoustique et le modèle de langage pendant la phase d'apprentissage. Un lexique doit aussi être fourni. On utilise ensuite ces composants pour transcrire un signal pendant la phase de décodage.

— d'une fonction de transition $\pi(q, a_\mathrm{i}, w)$ qui associe des éléments de $Q \times \Sigma \times \mathbb{K}$ à des éléments de Q. Ce sont les arcs du graphe représentant le modèle. Une transition relie deux états q et p de Q par un symbole a_i de Σ et y associe un score $w \in \mathbb{K}$.

Définition d'un transducteur Un wFST est similaire à un wFSA à l'exception qu'en plus de reconnaître les séquences en entrée, celles-ci sont transformées en d'autres séquences. Chaque transition reconnaît un symbole et transforme celui-ci en un autre. Les ensembles de symboles en entrée et en sortie peuvent être différents. La Fig. 2.6 donne un exemple.
Ainsi un wFST est un 6-uplet $(\Sigma, \Delta, Q, Q_\mathrm{i}, Q_\mathrm{f}, \pi)$. Les seules différences avec un wFSA sont :

— qu'il y a un alphabet Δ pour les symboles en sortie.

— que la fonction de transition $\pi(q, a_\mathrm{i}, a_\mathrm{f}, w)$ associe des éléments de $Q \times \Sigma \times \Delta \times \mathbb{K}$ à des éléments de Q. Une transition relie deux états q et p de Q, prend un symbole a_i de Σ en entrée, donne un symbole a_f de Δ en sortie et y associe un score $w \in \mathbb{K}$.

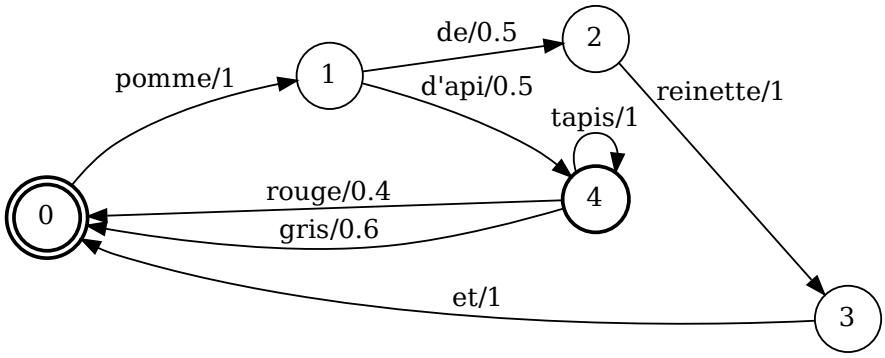


FIGURE 2.5. – Exemple de wFSA. Les états 0 et 4 sont les états initiaux. L'état 0 est le seul état final. Par exemple, ce wFSA reconnaît les séquences [**pomme**, **de**, **reinette**, **et**] (score total de $1 \times 0.5 \times 1 \times 1 = 0.5$), [**pomme**, **d'api**, **tapis**, **tapis**, **rouge**] (score total de $1 \times 0.5 \times 1 \times 1 \times 0.4 = 0.2$), [**gris**] (score de 0.6). Pour obtenir le score total d'une séquence nous avons ici multiplié les scores successifs de chaque transition.

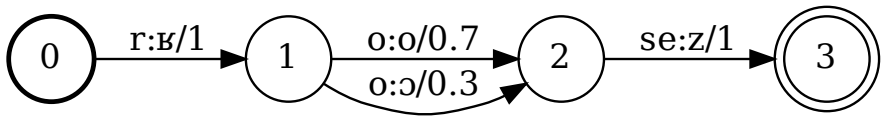


FIGURE 2.6. – Exemple de wFST. L'état 0 est le seul état initial. L'état 3 est le seul état final. Ce wFST n'accepte qu'une seule entrée [**r**, **o**, **se**] qui encode le mot « rose ». L'alphabet d'entrée contient des graphèmes. En sortie, nous avons les deux prononciations possibles de ce mot [**ʁ**, **o**, **z**] ou [**ʁ**, **ɔ**, **z**] avec respectivement des scores de 0.7 et 0.3. L'alphabet de sortie contient des phonèmes.

Opérations sur les scores Dans les exemples des Fig. 2.5 et 2.6, nous avons multiplié les scores des transitions pour obtenir les scores totaux. Mais il n'y a pas qu'une seule façon de définir cette opération de multiplication $\otimes$. Dans l'exemple de la Fig. 2.6, nous avons obtenu 2 séquences en sortie. Il nous faut également définir l'opération $\oplus$ si l'on veut combiner les scores. Ainsi afin de bien définir les wFSA et les wFST il faut munir l'espace des scores d'un demi-anneau $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ [Kuich and Salomaa, 2012]. Un tel demi-anneau a les propriétés suivantes :

— Les scores sont définis sur $\mathbb{K}$.

— L'opération $\oplus$ est commutative, associative et a pour élément neutre $\bar{0}$, c.-à-d. que $a \oplus \bar{0} = a$.

— L'opération $\otimes$ est associative. Elle a pour élément neutre $\bar{1}$, c.-à-d. que $a \otimes \bar{1} = a$.

— L'opération $\otimes$ est distributive par rapport à $\oplus$.

— $\bar{0}$ est absorbant pour $\otimes$, c.-à-d. que $a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$.

Ce sont en fait les propriétés d'un anneau à l'exception qu'il n'y a pas forcément d'inverse pour l'addition $\oplus$. Le Tab. 2.1 montre des exemples de demi-anneaux.

Demi-anneau	$\mathbb{K}$	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Booléen	$\{0,1\}$	$\vee$	$\wedge$	0	1
Probabilité	$\mathbb{R}^+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	logadd	$+$	$-\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	max	$+$	$-\infty$	0

TABLEAU 2.1. – Exemples de demi-anneaux courants. L'opération logadd est définie par $\text{logadd}(a, b) = \log(\exp(a) + \exp(b))$. L'annexe A définit cette fonction plus généralement et explique comment l'implémenter de façon à être numériquement stable.

Pour la RAP, nous manipulons couramment des probabilités. Le demi-anneau naturel est donc le demi-anneau des probabilités $(\mathbb{R}^+, +, \times, 0, 1)$. C'est avec cet anneau que nous avons manipulé les scores des Fig. 2.5 et 2.6. Mais pour des questions de stabilité numérique, nous utilisons en pratique des log-probabilités. C'est donc le demi-anneau Log, qui est l'image du demi-anneau des probabilités par la fonction log, que nous utiliserons par la suite. Lorsque nous voulons additionner des probabilités, il faudra à la place utiliser l'opération logadd sur les log-probabilités respectives. De même si nous voulons multiplier des probabilités, il faut à la place additionner les log-probabilités.

Il faut également mentionner le demi-anneau Tropical qui, au lieu de combiner les scores de deux chemins avec logadd, sélectionne celui qui a le meilleur score. Il permet de trouver le meilleur chemin parmi un ensemble de chemins possibles.

Les transitions epsilon Pour relier deux états sans produire ou consommer de nouveaux symboles, il peut être utile d'utiliser une transition epsilon. Pour se faire, les wFST utilisent le symbole ϵ qui est ajouté aux alphabets Σ et Δ. Avec un wFST, nous ne sommes donc pas contraint de transformer une séquence de symboles en une autre de même longueur. La Fig. 2.7 propose un exemple d'une telle transition.

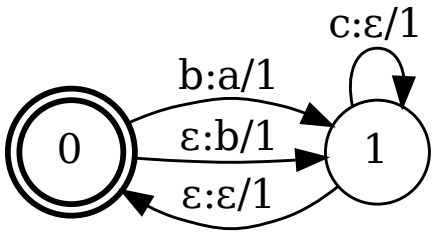
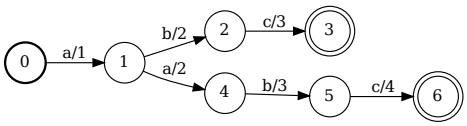


FIGURE 2.7. – Exemple de wFST avec des transitions epsilon. L'état 0 est l'état intial et final. Avec ce wFST, on peut produire autant de nouveaux symboles **b** que l'on veut. On peut aussi consommer n'importe quel nombre de symboles **c**. La séquence d'entrée [**c**, **b**, **c**] peut ainsi être transformée en [**b**, **a**] ou encore en [**b**, **b**, **b**, **a**, **b**, **b**].

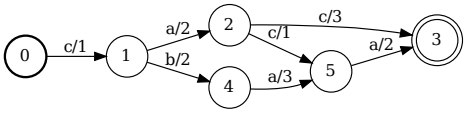
2.2.2. Opérations sur les wFSA et les wFST

Nous présentons dans cette partie quelques opérations basiques sur les wFSA et les wFST.

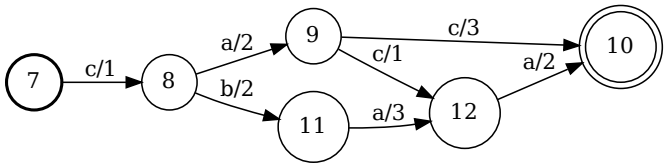
Union et concaténation L'union de deux wFSA ou wFST $B = A_1 \cup A_2$ contient tous les chemins de A_1 et de A_2. La concaténation $C = [A_1, A_2]$ contient les chemins pouvant être formés à partir d'un chemin de A_1 puis d'un chemin de A_2. Il suffit d'ajouter des transitions epsilon entre les états finaux de A_1 et les états initiaux de A_2. La Fig. 2.8 donne un exemple.



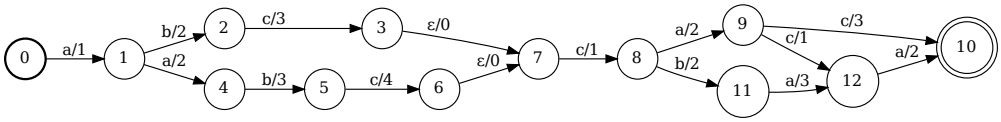
(a) A_1



(b) A_2



(c) $B = A_1 \cup A_2$



(d) $C = [A_1, A_2]$

FIGURE 2.8. – Représentation de A_1 (a), A_2 (b), B (c) et C (d). B est l'union de A_1 et A_2. C est la concaténation de A_1 et A_2.

Fermeture de Kleene Soit A un wFSA ou un wFST. La fermeture de Kleene ou l'étoile de Kleene $B = A^*$ permet à B d'accepter les chemins de A répétés un nombre arbitraire de fois. B accepte également le chemin vide. Il suffit de créer un nouvel état q qui soit initial et final. On ajoute des transitions epsilon qui partent de q pour aller aux états initiaux de A, et des transitions qui arrivent vers q et qui partent des états finaux de A. La Fig. 2.9 représente la fermeture de Kleene du wFSA A_1 de la Fig. 2.8a.

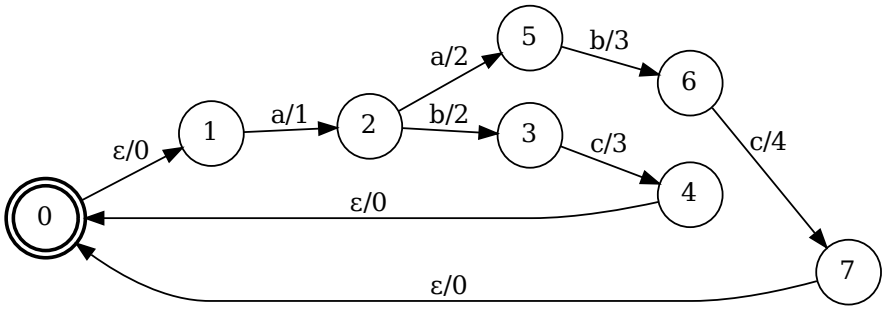


FIGURE 2.9. – Fermeture de Kleene $B = A_1^*$. Le wFSA A_1 est représenté dans la Fig. 2.8a.

2.2.3. Algorithmes sur les wFSA et les wFST

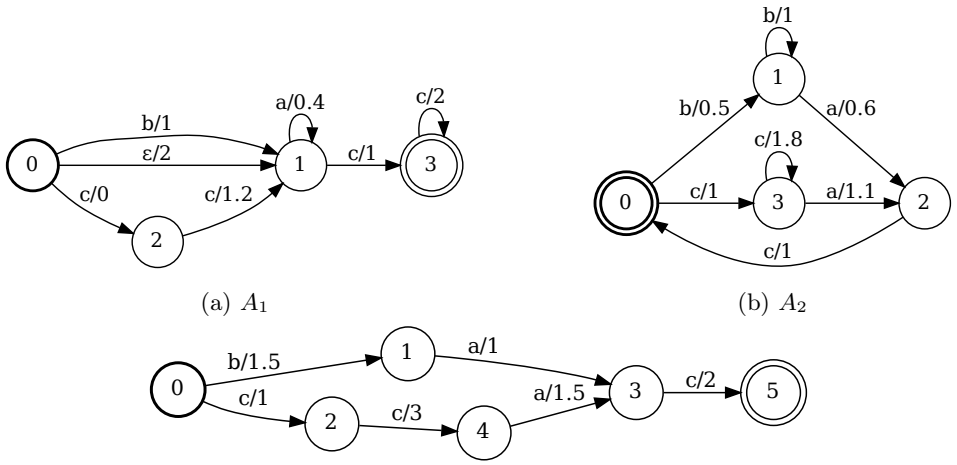
L'intérêt des wFST est de pouvoir utiliser un panel d'algorithmes bien étudiés et efficaces et cela pour différentes situations. Nous présentons dans cette partie les algorithmes que nous utilisons dans cette thèse. La bibliothèque openFST [Allauzen et al., 2007] propose des implémentations de tous ces algorithmes sur CPU. Les bibliothèques k2 [Povey et al., 2020] et GTN [Hannun et al., 2020] sont plus récentes et permettent de combiner l'utilisation des wFST avec des méthodes d'apprentissage profond. Tous les algorithmes ne sont pas présents dans ces bibliothèques. Pour les besoins de la thèse nous utiliserons GTN sauf pour la partie 4.4.4 où nous comparons différentes implémentations.

Intersection Nous notons $B = A_1 \circ A_2$ l'intersection de deux wFSA. B contient alors tous les chemins qui sont acceptés par A_1 et par A_2. Le score d'un chemin de B est obtenu en agrégeant les scores de A_1 et de A_2 de ce même chemin par l'opération $\otimes$ (c'est-à-dire l'addition dans le demi-anneau Log). La Fig. 2.10 illustre un exemple d'intersection de deux wFSA.

Composition Nous notons également $B = A_1 \circ A_2$ la composition de deux wFST. Si A_1 peut transformer le chemin π_1 en π_2 et si A_2 peut transformer le chemin π_2 en π_3 alors $B = A_1 \circ A_2$ peut transformer π_1 en π_3. La Fig. 2.11 illustre un exemple de composition de deux wFST. La composition de deux wFST ou l'intersection de deux wFSA utilisent en fait le même algorithme car un accepteur peut être vu comme un transducteur avec les mêmes symboles d'entrée et de sortie. L'intersection de deux accepteurs est donc équivalente à la composition de deux transducteurs.

La composition de deux wFST $B = A_1 \circ A_2$ [Kuich and Salomaa, 2012, Mohri, 2009, Mohri et al., 2005] est une généralisation du cas non pondéré [Eilenberg, 1974, Hopcroft et al., 2001]. Voici l'idée générale de l'algorithme :

— Il faut d'abord créer les états de B. Il y en a un pour chaque paire $q = (q_1, q_2)$ d'états de A_1 et A_2. La propriété d'état final ou initial est attribuée à q si q_1 et q_2 possèdent cette propriété.



(a) A_1 (b) A_2

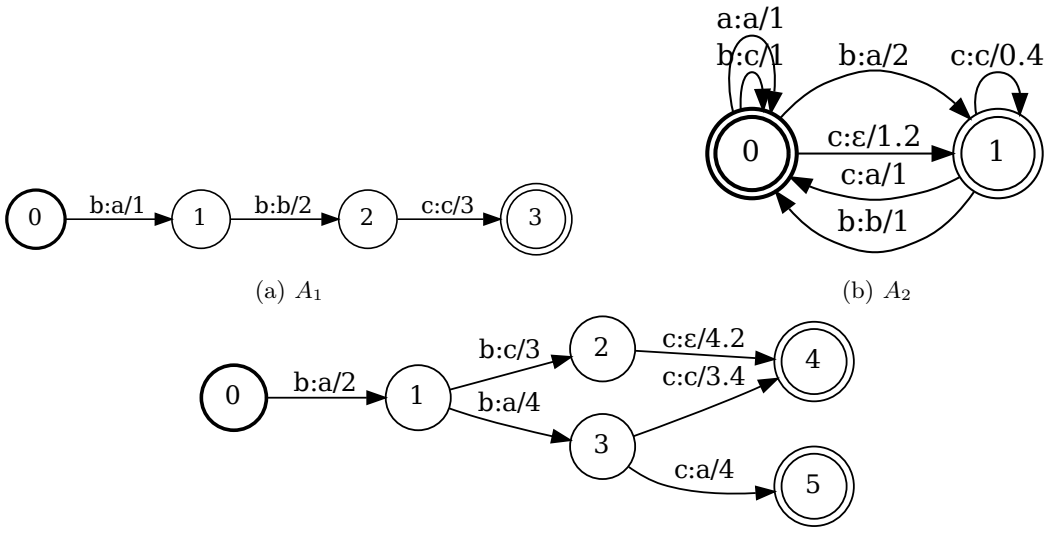
(c) $B = A_1 \circ A_2$

FIGURE 2.10. – Représentation des trois wFSA A_1 (a), A_2 (b) et B (c). B est le résultat de l'intersection de A_1 et A_2. B contient deux chemins. Le premier [**b**, **a**, **c**] a un poids total de 4.5. Les contributions à ce score de A_1 et de A_2 sont respectivement de 2.4 et 2.1. Le deuxième chemin [**c**, **c**, **a**, **c**] a un poids total de 7.5. Les contributions à ce score de A_1 et de A_2 sont respectivement de 2.6 et 4.9.

— Ensuite il faut créer les transitions de B.
  — Soient t_1 une transition de A_1 allant de q_1 à q_1' et t_2 une transition de A_2 allant de q_2 à q_2'.
  — Pour chaque paire $t = (t_1, t_2)$ on crée une transition dans B entre l'état $q = (q_1, q_2)$ et $q' = (q_1', q_2')$ si le symbole de sortie de t_1 est le même que le symbole d'entrée de t_2. Le symbole d'entrée de t est alors celui de t_1 et le symbole de sortie est celui de t_2. Son poids est le produit $\otimes$ des poids de t_1 et t_2.

L'algorithme ci-dessus est imparfait. Il crée de nombreux états et transitions qui ne sont pas joignables et il n'est pas efficace lorsque le nombre de transitions est trop grand. Il faut aussi adapter l'algorithme lorsque les transitions epsilon sont autorisées dans le cas pondéré [Mohri et al., 2005]. L'algorithme de composition est en fait un algorithme d'appariement entre les transitions t_1 de A_1 et t_2 de A_2 qui cherche les transitions $t = (t_1, t_2)$ valides. Il est alors utile de trier au préalable les transitions de A_1 et de A_2 pour rendre cette recherche plus efficace.

Soit $|A| = |A|_Q + |A|_E$, où $|A|_Q$ est le nombre d'états et $|A|_E$ le nombre de transitions du wFST A. Puisque chaque transition de A_1 partant d'un état q_1 peut être appariée à toutes les transitions de A_2 partant d'un état q_2, l'algorithme a une complexité de $\mathcal{O}(|A_1||A_2|)$ dans le pire des cas [Mohri, 2009]. Cette complexité dépend fortement de la structure des transducteurs utilisés. Le degré extérieur d'un état est le nombre de transitions partant



(a) A_1 (b) A_2

(c) $B = A_1 \circ A_2$

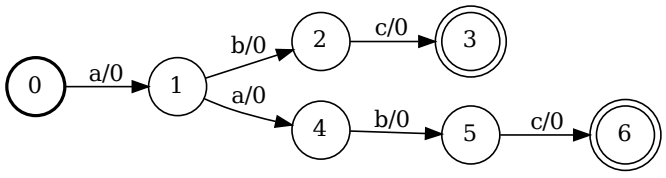
FIGURE 2.11. – Représentation des trois wFST A_1 (a), A_2 (b) et B (c). B est le résultat de la composition de A_1 et A_2. A_1 peut transformer [**b**, **b**, **c**] en [**a**, **b**, **c**] avec un score total de 6. A_2 peut transformer [**a**, **b**, **c**] de trois manières. D'abord en [**a**, **c**] avec un score de 3.2, en [**a**, **a**, **c**] avec un score de 3.4 et en [**a**, **a**, **a**] avec un score de 4. Dans le résultat de la composition B, nous retrouvons bien les trois chemins avec des scores de 9.2, 9.4 et 10.

de cet état. Soit $|A|_\mathrm{d}$ le maximum des degrés extérieurs des états du transducteur A et $|A|_\mathrm{m}$ la multiplicité maximum de ses états (définition dans le paragraphe suivant). La librairie openFST [Allauzen et al., 2009, 2007] propose un algorithme de composition de complexité $\mathcal{O}(|A_1|_Q|A_2|_Q|A_1|_\mathrm{d}(\log|A_2|_\mathrm{d} + |A_2|_\mathrm{m}))$.

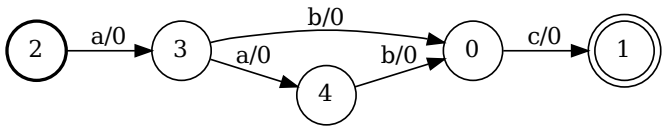
Déterminisation La multiplicité m d'un état est le nombre maximal de transitions partant de cet état qui partagent le même symbole d'entrée. Un wFST est déterministe s'il possède un seul état initial et si la multiplicité maximale de ses états $|A|_\mathrm{m} = 1$. On voit bien l'intérêt d'avoir un wFST déterministe pour l'utilisation de l'algorithme de composition car sa complexité dépend de $|A|_\mathrm{m}$. La déterminisation consiste à rendre déterministe un wFST qui ne l'est pas. L'algorithme de déterminisation [Mohri, 1997] a une complexité exponentielle mais en pratique il peut tout de même être utilisé. Tous les wFST ne sont pas déterminisables. En pratique, on essaiera de construire directement des wFST déterministes. L'algorithme est bien présent dans openFST [Allauzen et al., 2007] et dans l'outil k2 [Povey et al., 2020] mais pas dans GTN [Hannun et al., 2020] qui est l'outil que nous utilisons.

Minimisation Deux wFST sont équivalents si, pour chaque séquence d'entrée, ils associent la même séquence de sortie avec le même poids total. L'algorithme de minimisa-

tion permet de construire un wFST A' équivalent à A qui possède le plus petit nombre d'états et de transitions possibles. Tous les wFST déterministes peuvent être minimisés. Un exemple est donné dans la Fig. 2.12. L'algorithme a une complexité de $\mathcal{O}(|A|)$ dans le cas acyclique et $\mathcal{O}(|A|_E \log |A|_Q)$ dans le cas général [Mohri, 1997]. L'algorithme est bien présent dans openFST mais pas dans k2 ni dans GTN. Pour les besoins de la thèse, nous avons implémenté dans GTN l'algorithme de minimisation pour les FSA dans le cas acyclique, non pondéré et sans transition epsilon.



(a) Le FSA initial A



(b) Le FSA minimisé A'

FIGURE 2.12. – Illustration de l'algorithme de minimisation. Le FSA A (a), qui possède deux chemins [**a**, **b**, **c**] et [**a**, **a**, **b**, **c**], est construit comme un arbre dont le préfixe **a** est commun aux deux chemins. L'application de l'algorithme de minimisation (b) permet de mettre en commun le suffixe **c**.

Calcul de distance On utilise un algorithme de parcours de graphe en largeur pour calculer la distance entre deux états d'un wFST. La complexité est donc de $O(|A|_Q+|A|_E)$ si le graphe est acyclique, c'est-à-dire qu'il ne possède pas de chemins de longueur infinie. On restera dans ce cas lorsque l'on fera un calcul de distance. L'algorithme consiste à cumuler les poids sur les transitions pendant le parcours du graphe. Ces poids peuvent être cumulés selon le demi-anneau Log ou Tropical. Si l'on utilise le demi-anneau Tropical on aura alors le poids cumulé du meilleur chemin. Avec le demi-anneau Log on aura le poids agrégé par la fonction logadd de tous les chemins reliant les deux états. Ce qui nous intéresse particulièrement est la distance totale entre les états initiaux et finaux, c'est-à-dire les chemins acceptés par le wFST.

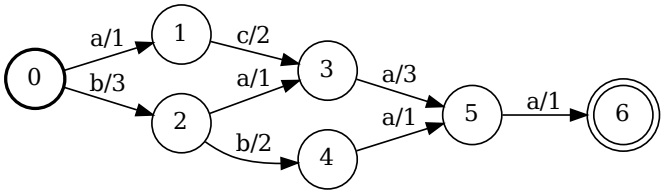
On note Score(.) la fonction qui calcule cette distance dans le demi-anneau Log et ScoreViterbi(.) ce même calcul dans le demi-anneau Tropical. La Fig. 2.13 présente un exemple de calcul de cette distance en appliquant l'algorithme de parcours de graphe. On note $\pi = [\pi_1, \ldots, \pi_n]$ un chemin d'un graphe dont les poids sont $[w_1, \ldots, w_n]$, $\pi \in A$ tous les chemins acceptés du wFST A et $s(\pi)$ le score total d'un chemin. Les équations

(2.6), (2.7) et (2.8) explicitent ce que l'algorithme de distance calcule :

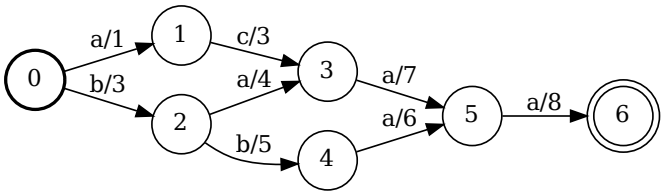
$$s(\pi) = \sum_{i=1}^{n} w_i \tag{2.6}$$

$$\text{ScoreViterbi}(A) = \max_{\pi \in A} s(\pi) \tag{2.7}$$

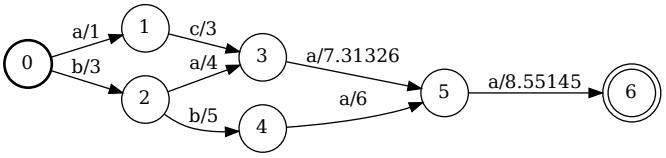
$$\text{Score}(A) = \underset{\pi \in A}{\text{logadd}}\ s(\pi). \tag{2.8}$$



(a) Le wFSA initial A



(b) Les poids sont cumulés en utilisant la fonction max.



(c) Les poids sont cumulés en utilisant la fonction logadd.

FIGURE 2.13. – Illustration de l'algorithme de calcul de distance entre l'état initial 0 et l'état final 6 du wFSA A (a). (b) et (c) représentent le même wFSA mais dont les poids sont cumulés avec la fonction max ou logadd. On a donc $\text{ScoreViterbi}(A) = 8$ et $\text{Score}(A) \approx 8.55$.

2.2.4. Les wFST différentiables

Dans l'outil GTN, Hannun et al. [2020] proposent pour la première fois de calculer automatiquement le gradient d'une fonction scalaire lorsque celle-ci provient d'une succession d'opérations de wFST. L'outil k2 [Povey et al., 2020] permet aussi de faire cela.

Définitions Soient $X_1, \ldots, X_k$ des wFST. Soit $w = [w_1, w_2, \ldots, w_n]$ l'ensemble des poids de ces k wFST. Nous ne fixons pas ces poids, ce sont des variables. On considère $F_1, F_2, ..., F_L$ un ensemble de fonctions opérant sur des wFST et Y le wFST défini

par

$$Y = F_1 \circ F_2 \circ \cdots \circ F_L(X_1, \ldots, X_k). \tag{2.9}$$

Notons $y \in \mathbb{R}^{n_1}$ les poids de Y. Ces poids sont le résultat d'une succession d'opérations opérant sur l'ensemble des poids w. Ces opérations sont définies par

$$\begin{aligned} &f_l : \mathbb{R}^{n_{l+1}} \to \mathbb{R}^{n_l}, n_L = n \\ &y = f_1 \circ f_2 \circ \cdots \circ f_L(w). \end{aligned} \tag{2.10}$$

Le but est de calculer le gradient d'un des poids de Y par rapport aux poids w. On veut aussi évaluer ce gradient pour des wFST $X_1, \ldots, X_k$ fixés avec $A_1, \ldots, A_k$ dont l'ensemble de tous les poids est $a = [a_1, a_2, \ldots, a_n]$. Pour le poids y_i nous avons

$$\nabla y_i(a) = \begin{bmatrix} \frac{\partial y_i}{\partial w_1}(a) \\ \frac{\partial y_i}{\partial w_2}(a) \\ \vdots \\ \frac{\partial y_i}{\partial w_n}(a) \end{bmatrix}. \tag{2.11}$$

Pour calculer cela notons J_y la matrice jacobienne de y :

$$J_y = \begin{bmatrix} \frac{\partial y_1}{\partial w_1} & \cdots & \frac{\partial y_1}{\partial w_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_{n_L}}{\partial w_1} & \cdots & \frac{\partial y_{n_L}}{\partial w_n} \end{bmatrix}. \tag{2.12}$$

Le gradient que l'on cherche est la i-ème ligne de cette matrice jacobienne transposée et évaluée en a. Pour le calcul, on applique le théorème de dérivation des fonctions composées :

$$\begin{aligned} J_y(a) &= J_{f_1 \circ f_2 \circ \cdots \circ f_L}(a) \\ &= J_{f_1}[f_2 \circ \cdots \circ f_L(a)] \times \cdots \times J_{f_l}[f_{l+1} \circ \cdots \circ f_L(a)] \times \cdots \times J_{f_L}(a). \end{aligned} \tag{2.13}$$

GTN et k2 permettent de calculer ces matrices jacobiennes automatiquement. Pour ce faire, ces outils construisent un graphe des opérations. Ainsi on peut retrouver comment a été construit un poids y_i lors du calcul des dérivées partielles. Il y a deux étapes : la propagation avant et la propagation arrière. La propagation avant construit les wFST selon une suite d'opérations, la propagation arrière calcule les dérivées partielles.

Prenons une opération en particulier $V = F_l(U)$ opérant sur des wFST $U = U_1, \ldots, U_k$. Pendant la propagation avant, on opère une modification de l'ensemble des poids u par la fonction f_l pour obtenir les poids de V qui sont $v = f_l(u)$. f_l est bien définie lors de la construction de V. Mais on définit aussi J_{f_l} lors de cette étape. La matrice jacobienne sera utilisée pendant la propagation arrière lors du calcul du gradient de l'équation (2.11).

L'annexe B présente un exemple complet de propagation avant et arrière. Les calculs des matrices Jacobiennes pour l'opération de composition et pour la fonction Score y sont notamment présentés.

2.2.5. Implémentation

Pour les opérations et algorithmes sur les wFST, GTN s'inspire largement des bibliothèques existantes comme openFST [Allauzen et al., 2007] et de son prédécesseur FSM [Mohri et al., 2000]. Le calcul des gradients s'inspire des bibliothèques de différentiation automatique et d'apprentissage profond [Paszke et al., 2019, Pratap et al., 2019]. Le but principal de k2 et GTN est de pouvoir s'interfacer facilement avec des bibliothèques d'apprentissage profond car celles-ci peuvent utiliser les gradients calculés avec les wFST.

k2 et GTN sont programmés en langage C++ pour la majeure partie. Les deux bibliothèques proposent une interface en Python. GTN peut être utilisé directement dans un autre programme en C++ mais k2 est conçu pour être utilisé uniquement avec l'interface Python. Durant la thèse nous avons utilisé la bibliothèque Flashlight (anciennement Wav2letter++) directement en C++ pour créer des algorithmes d'apprentissage profond pour la RAP. La bibliothèque GTN est donc adaptée pour inclure des wFST dans nos algorithmes d'apprentissage.

Les bibliothèques d'apprentissage profond ont su utiliser ces dernières années le potentiel des GPU. Ces composants sont particulièrement efficaces pour effectuer beaucoup de calculs en parallèle. Mais historiquement les algorithmes sont souvent conçus pour être exécutés sur CPU. L'utilisation des GPU nécessite d'adapter les algorithmes et de les implémenter dans le langage natif CUDA intégrable dans un programme C++. Les bibliothèques d'apprentissage automatique ont déjà accumulé beaucoup d'expérience dans l'utilisation des GPU, c'est même la norme de les utiliser. Mais les bibliothèques k2 et GTN sont beaucoup plus récentes. Elles tentent tout de même d'implémenter des versions GPU pour certains algorithmes. On peut noter que k2 a une certaine avance sur GTN pour l'implémentation des versions GPU, mais tous les algorithmes ne sont pas encore disponibles. Mais GTN fait aussi des efforts dans ce sens en implémentant récemment un algorithme de composition sur GPU [Sengupta et al., 2021]. Dans le cadre de cette thèse nous utilisons essentiellement les algorithmes sur CPU de GTN. Cela nous permet d'ajouter plutôt rapidement nos propres algorithmes sur CPU.

2.2.6. Utilisation pour la RAP

L'utilisation de wFST pour la RAP est bien documentée [Mohri, 1997, Mohri et al., 2002, 2005, 2008]. L'outil peut servir à implémenter un modèle *Hidden Markov Model* (HMM), un lexique ou encore un modèle de langage. Mais dans ces travaux il n'est pas fait mention de wFST différentiable. Les wFST sont définis et utilisés pendant la phase de décodage, mais pas pendant la création du système lui-même. Kaldi [Povey et al., 2011] qui est un outil pour la RAP utilise beaucoup les wFST lors de la phase de décodage (voir

partie 2.4). Les wFST peuvent aussi être utilisés pour incorporer des « treillis » fixés à l'avance dans une fonction de coût pour l'apprentissage d'un modèle [Veselý et al., 2013].

L'utilisation des wFST différentiables permet une nouvelle utilisation. On peut directement incorporer cet outil dans un algorithme d'apprentissage qui se base sur la descente de gradient (voir la partie 2.3.3) [Hannun et al., 2020]. L'outil nous permet d'incorporer facilement une source d'information externe à un algorithme d'apprentissage (voir chapitre 4). Sans cet outil nous devons incorporer à la main ces informations. Cela ajoute une couche de complexité et rend l'algorithme inutilisable si le type d'information à ajouter change (voir chapitre 3).

2.2.7. L'apprentissage de wFST

Dans cette thèse nous utilisons les wFST pour implémenter les fonctions de coût CTC et ASG. Nous le verrons dans le chapitre 4, il faut pour cela créer des wFSA avec une structure particulière modélisant les règles de construction de ces fonctions de coût. Les poids viennent du modèle acoustique que l'on cherche à apprendre. Pour cette utilisation, la structure et les poids des wFSA utilisés sont bien définis.

Méthodes spectrales Si l'on ne connaît pas les poids ni la structure du wFSA, il est possible de les trouver avec les méthodes spectrales [Balle et al., 2013, Balle and Mohri, 2015, 2018]. Le wFSA est obtenu à partir de l'approximation d'une matrice de Hankel. Il faut pour cela fournir un jeu de données comprenant une liste de séquences que le wFSA doit reconnaître. Il faut aussi indiquer le nombre maximal d'état du wFSA. De plus, pour une séquence x reconnue par le wFSA, il est souvent utile d'avoir une interprétation probabiliste du score de la séquence. Il faut pour cela adapter un peu la méthode [Balle et al., 2013, Hsu et al., 2012]. Il existe une bibliothèque Python qui implémente ces algorithmes [Arrivault et al., 2017].

Pour faire la même chose avec les wFST il faut prendre en considération plusieurs problèmes. D'abord il faut un jeu de données contenant des paires de séquences (x_1, x_2) pour l'entrée et la sortie du wFST. Balle et al. [2011] proposent une solution mais il faut un alignement entre x_1 et x_2. Bailly et al. [2013] présentent une extension pour les paires non-alignées. Ces méthodes consistent en fait à créer des paires de symboles d'entrée et de sortie et à considérer chaque paire comme un symbole unique. Si l'on veut alors interpréter les scores de façon probabiliste nous obtenons $p(x_1, x_2)$. Mais en pratique nous voulons $p(x_2|x_1)$. Il faut alors normaliser les probabilités pour obtenir ce que l'on veut [Eisner, 2002].

Méthodes par descente de gradients Il est aussi possible d'exploiter la propriété différentiable des wFST que nous utilisons. Eisner [2002] fait mention de cette possibilité mais globalement le sujet n'est pas encore étudié puisque la propriété différentiable des

wFST est récente. Dans le chapitre 4 nous proposons une méthode d'apprentissage des poids d'un wFST à partir d'une structure de graphe fixée à l'avance.

2.3. Le modèle acoustique

2.3.1. Historique

Les débuts de la RAP. Le rôle du modèle acoustique est d'estimer la probabilité $P(O|W)$ de l'équation (2.5). Les premiers essais datent des années 1950 [Furui, 2005]. Les laboratoires Bell développent en 1952 un système capable de reconnaître les 10 chiffres en estimant les fréquences d'apparition de formants dans les régions des voyelles de chaque chiffre [Davis et al., 1952]. Ce n'est que dans les années 1980 que l'on commence à avoir les premiers systèmes de RAP capables de reconnaître plusieurs milliers de mots. Cela a été possible grâce à un changement de paradigme en passant d'approches de reconnaissance de motifs dans le domaine spectral à des approches statistiques. La plus grande avancée fut le développement du modèle statistique des chaînes de Markov cachées (HMM, *Hidden Markov Model*) [Juang and Rabiner, 1991, Rabiner, 1993, 1989].

Un HMM est un modèle génératif qui décrit comment une séquence d'états $Q = q_1...q_T$ peut générer une séquence d'observations acoustiques O de même longueur T. Ce modèle a la propriété de pouvoir modéliser la temporalité d'une séquence sur un assez long terme. Les états q_t encodent la phrase W à reconnaître. Ces états peuvent être des phonèmes, et alors ils sont appelés monophones. Mais cette modélisation ne permet pas de prendre le contexte : en effet un phonème peut être prononcé différemment selon ce qui précède et ce qui suit dans la phrase. C'est pourquoi un état est créé pour chaque triplet de phonèmes. Pour éviter l'explosion du nombre d'états, les états représentant des phonèmes similaires sont regroupés. Les états ainsi obtenus sont appelés triphones à états partagés (*clustered triphones*) [Gales and Young, 2008]. La séquence d'états q correspondant au signal à transcrire n'est pas connue, les états sont donc dits « cachés ». Nous avons seulement accès à l'observation O. Pour ce modèle, on applique l'hypothèse de Markov d'ordre 1 ce qui simplifie la complexité :

$$P(Q) = P(q_1) \prod_{t=2}^{T} P(q_t|q_{t-1}). \tag{2.14}$$

Nous faisons aussi l'hypothèse d'indépendance conditionnelle des observations :

$$P(O|Q) = \prod_{t=1}^{T} P(o_t|q_t). \tag{2.15}$$

Nous pouvons ensuite retrouver $P(O|W)$ en faisant la somme sur toutes les prononciations valides de la phrase W :

$$P(O|W) = \sum_{Q} P(O|Q)P(Q|W). \tag{2.16}$$

Les termes $P(o_t|q_t)$ peuvent être modélisés par des mélanges de gaussiennes, *Gaussian Mixture Model* (GMM). Dans ce cas, le modèle acoustique est un GMM-HMM. Les GMM sont des modèles génératifs qui caractérisent chaque état indépendamment des autres. Mais la capacité de modélisation des GMM est un facteur limitant pour ces modèles.

Utilisation des réseaux de neurones. À partir de la fin des années 1980 beaucoup de travaux ont essayé d'utiliser des réseaux de neurones artificiels - *Artificial neural network* (ANN) - pour la RAP car ceux-ci ont une bonne capacité de modélisation [Cosi et al., 1990, Haffner et al., 1989, Lippmann, 1989, Mori et al., 1989, Robinson, 1992].

Il n'y a pas de différence fondamentale avec la première proposition du perceptron multicouches de Rosenblatt [1958] et les ANN. Mais les résultats étaient décevants car le modèle était en fait incapable d'approximer des fonctions très simples. Ce n'est que dans les années 1970 et 1980 qu'il a été découvert qu'une simple méthode de descente de gradient permettait de trouver de bons paramètres [LeCun, 1985] avec l'algorithme de rétro-propagation du gradient.

Les ANN sont des modèles discriminants qui sont entraînés à séparer des classes entre elles. C'est-à-dire que l'on va directement essayer d'assigner une catégorie à une observation. La RAP n'est pas directement un problème de classification mais cela peut être formulé comme tel comme nous allons le voir avec les modèles hybrides et les DNN. Deux propositions ont particulièrement influencé le domaine : les *Time-delay neural networks* (TDNN) [Lang, 1988, Waibel, 1989, Waibel et al., 1989] et les réseaux de neurones récurrents [Elman, 1990, Mozer, 1993], *Recurrent neural networks* (RNN).

Les modèles hybrides. Durant les années 1990 il y a eu des tentatives pour combiner les HMM avec les ANN pour combler les faiblesses des deux approches. Cela a formé une nouvelle classe de modèle hybride ANN-HMM [Bourlard and Wellekens, 1990, Bourlard and Morgan, 1994, Franzini et al., 1990, Levin, 1990, Morgan and Bourlard, 1990, Trentin and Gori, 2001]. Dans ce modèle hybride les ANN peuvent être utilisés à la place des GMM pour estimer les probabilités a posteriori des états $P(q_t|o_t)$ [Richard and Lippmann, 1991]. Les probabilités $P(o_t|q_t)$ sont ensuite obtenues en calculant :

$$P(o_t|q_t) = \frac{P(q_t|o_t)P(o_t)}{P(q_t)} \propto \frac{P(q_t|o_t)}{P(q_t)}. \tag{2.17}$$

L'évidence $P(o_t)$ peut être omise car elle ne dépend pas de q_t et la probabilité a priori $P(q_t)$ ne dépend pas de t.

Les ANN ont aussi été utilisés comme étape de pré-traitement pour générer de meilleurs descripteurs pour un GMM-HMM. Dans cette catégorie, il y a la technique dit du « goulot d'étranglement » qui consiste à utiliser la représentation extraite d'une couche intermédiaire de faible dimension d'un ANN comme vecteur de descripteurs [Grézl et al., 2007]. Il

y a aussi la méthode Tandem qui utilise les logarithmes des probabilités a posteriori issues de la couche de sortie d'un ANN comme descripteurs [Hermansky et al., 2000, Stolcke et al., 2006, Tóth et al., 2008]. Les ANN ont aussi été utilisés pour transformer l'espace continu des observations acoustiques en un espace discret en utilisant un apprentissage non supervisé [Iwamida et al., 1991]. L'utilisation d'un espace discret pour les HMM évite de modéliser une distribution de probabilité continue avec les GMM. Toutes ces approches nécessitent cependant de séparer l'apprentissage en plusieurs étapes, ce qui rend le processus assez complexe. Aujourd'hui les systèmes de l'état de l'art reposent sur des DNN.

Les réseaux de neurones profonds

Les *Deep Neural Network* (DNN) sont des modèles permettant une modélisation fine des données. Ils peuvent fournir une approximation correcte d'une fonction arbitraire que l'on cherche. Tout comme les ANN, ces modèles sont composés de paramètres organisés en plusieurs couches. Chaque couche détermine un niveau d'abstraction différent des données d'entrée. Les premières couches transforment les données d'entrée et fournissent une représentation de plus haut niveau. Au fur et à mesure que l'on se rapproche de la couche finale nous obtenons une représentation de plus en plus utile pour répondre au problème. La couche finale nous fournit ce qu'on veut finalement modéliser : pour la RAP c'est une probabilité dépendant directement de l'observation et qui sert à construire la probabilité $p(W|O)$ de l'équation (2.4). Les DNN sont utilisés comme modèles discriminants et non comme modèles génératifs comme les HMM. Les ANN se distinguent des DNN par le nombre de paramètres utilisés ainsi que la structure des couches. Toute la difficulté est d'estimer ces paramètres pendant la phase d'apprentissage.

L'apprentissage profond (*deep learning*) [Goodfellow et al., 2016, LeCun et al., 2015] est en fait l'ensemble des techniques nécessaires à la conception et à l'utilisation d'un DNN. Cette pratique s'est fortement développée à partir des années 2010. Le fonctionnement des différentes étapes d'obtention d'un DNN et plus particulièrement les éléments nécessaires à la compréhension des chapitres de ce manuscrit sont détaillés dans les sous-parties suivantes. Nous séparons ces étapes en trois : le choix de l'architecture (2.3.2), celui de la fonction de coût (2.3.3) et de l'algorithme d'apprentissage (2.3.4).

2.3.2. Architectures de réseaux de neurones profonds

Les modèles pour la RAP doivent prendre en compte qu'il n'y a pas d'alignement disponible entre la parole et le texte dans le jeu de données. On a accès au texte prononcé durant toute la durée du fichier audio mais à un instant t on ne sait pas quel est le phonème qui a été prononcé. Cela peut être résolu par l'utilisation de modèles séquence-à-séquence [Sutskever et al., 2014] qui modélisent un alignement implicite par l'intermédiaire du mécanisme d'attention présent dans les couches du modèle [Bahdanau et al., 2015, Chorowski et al., 2015]. Nous utiliserons plutôt des modèles moins structurés

avec une fonction de coût comme CTC [Graves et al., 2006] ou ASG [Collobert et al., 2016] pour modéliser l'alignement. L'avantage est que toutes les trames peuvent être traitées ensemble et non pas une par une comme avec les modèles séquence-à-séquence. Ce processus est schématisé sur la Fig. 2.14.

Nous utilisons deux architectures dans ce manuscrit. L'une se base sur des couches convolutionnelles et l'autre des couches « Transformer ». La dernière couche du modèle est toujours une couche linéaire qui projette la dimension de la séquence finale vers le nombre de symboles utilisés M. L'architecture Transformer permet d'obtenir un meilleur modèle acoustique mais l'utilisation de celle-ci n'était pas encore démocratisée au début de la thèse.

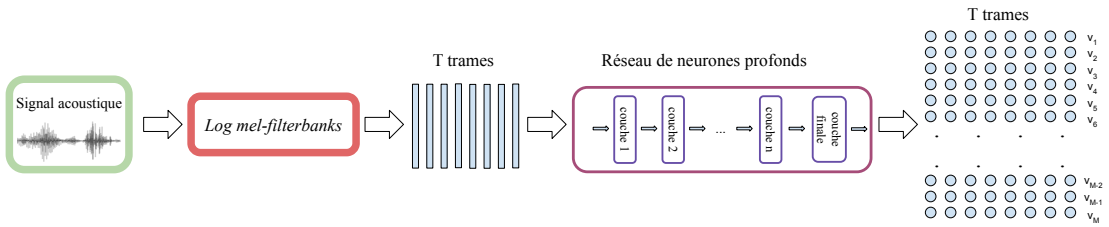


FIGURE 2.14. – Un réseau de neurones profond produit les scores $s(v_i^t|X)$ pour chaque trame t et pour chaque symbole v_i du dictionnaire de symboles de taille M.

2.3.2.1. Architecture *Gated ConvNet*

Gated ConvNet [Dauphin et al., 2017, Liptchinsky et al., 2017] est une architecture qui se base sur l'utilisation de couches convolutionnelles. C'est une des architectures par défaut présente dans l'outil Wav2Letter [Collobert et al., 2016]. Elle est composée d'une succession de couches de convolution 1D *Gated*. C'est-à-dire qu'à chaque couche une fonction d'activation *Gated Linear Unit* (GLU) est utilisée. La i-ème couche prend en entrée une matrice $X^i \in \mathbb{R}^{T^i \times d^i}$ comportant T^i trames de dimension d^i. Les poids de cette matrice sont d'abord normalisés pour qu'elle soit de moyenne nulle et de variance égale à 1 [Salimans and Kingma, 2016]. Ensuite l'opération suivante est effectuée :

$$X^{i+1} = (X^i * (W^i, b^i)) \otimes \sigma(X * (V^i, c^i)) \in \mathbb{R}^{T^{i+1} \times d^{i+1}} \tag{2.18}$$

avec W^i, $V^i \in \mathbb{R}^{d^{i+1} \times d^i \times \mathrm{kw}^i}$ et b^i, $c^i \in \mathbb{R}^{d^{i+1}}$ les paramètres à apprendre. d^{i+1} est le nombre de noyaux de convolution utilisés pour chaque couche d^i, c'est aussi la dimension en sortie de X^{i+1}. kw^i (*kernel width*) est la taille des noyaux de convolution de la couche i. $\sigma(.)$ est la fonction sigmoïde et $\otimes$ est le produit point à point entre matrices. Enfin si l'on pose $Y = X * (Z, b)$, $*$ est l'opérateur de convolution défini par :

$$Y_{j,t} = b_j + \sum_{k=1}^{d^i} \sum_{l=1}^{\mathrm{kw}^i} Z_{j,k,l} X_{k,\mathrm{sw}^i \times (t-1)+l} \quad \forall j \in \{1, \ldots, d^{i+1}\},\ t \in \{1, \ldots, T^{i+1}\}. \tag{2.19}$$

La valeur de T^{i+1} dépend de sw^i (*stride width*) qui indique le pas de déplacement du noyau de convolution. T^{i+1} est également dépendant de la stratégie de gestion des effets de bord avec l'utilisation du *padding*, qui est le remplissage par des valeurs nulles des séquences à gauche et à droite. Le *padding* permet de forcer plusieurs séquences à avoir la même longueur. C'est efficace car alors on peut traiter ces séquences en une seule fois, c'est de l'apprentissage par *batch*. Pendant l'apprentissage nous regroupons ensemble les séquences de longueurs similaires, ainsi le *padding* a peu d'influence sur le nombre de trames obtenu en sortie de la couche. On a approximativement $T^{i+1} \simeq \frac{T^i}{s_w^i}$. Le nombre de trames peut donc varier entre chaque couche et l'utilisation du *stride* permet de réduire le nombre de trames à traiter pour les couches suivantes.

Chaque couche i est également suivie d'une opération de *dropout* [Srivastava et al., 2014]. Avec une probabilité p, chaque valeur en sortie de la couche peut être mise à 0. Cette opération permet au modèle de mieux généraliser en évitant le problème de sur-apprentissage (*overfitting*). La Fig. 2.15 résume la composition ainsi que les hyper-paramètres à choisir pour chaque couche d'un modèle *Gated ConvNet*.

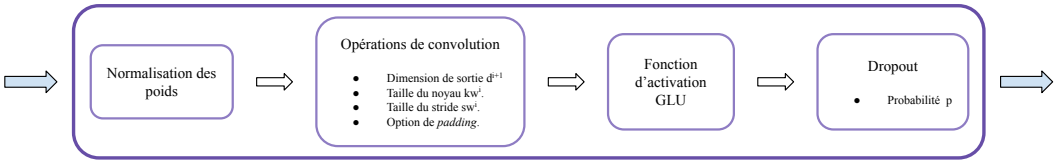


FIGURE 2.15. – Composition de la i-ème couche d'un réseau *Gated ConvNet*.

2.3.2.2. Architecture Transformer

Les couches Transformer ont été introduites par Vaswani et al. [2017] pour un modèle séquence-à-séquence de traduction. Depuis, l'utilisation de cette architecture s'est largement répandue à d'autres domaines dont celui de la RAP [Gulati et al., 2020, Hsu et al., 2021, Synnaeve et al., 2019]. Les couches Transformer se basent sur le mécanisme d'auto-attention à plusieurs têtes. Avec $X^i \in \mathbb{R}^{T^i \times d^i}$ comme séquence d'entrée, la sortie X^{i+1} d'une couche Transformer est obtenue par les opérations suivantes :

$$z(X^i) = \text{Norm}(\text{AttentionMultiTêtes}(X^i) + X^i) \tag{2.20}$$

$$X^{i+1} = \text{Norm}(\text{FFN}(z(X^i)) + z(X^i)) \tag{2.21}$$

où Norm(.) est l'opération de normalisation par couche [Ba et al., 2016]. Le fait de réinjecter la matrice X^i dans (2.20) et $z(X^i)$ dans (2.21) s'appelle une connexion résiduelle [He et al., 2016] et cela facilite l'apprentissage du modèle. L'opération FFN(.) est un bloc *Feed Forward Network* composée de deux couches linéaires et d'une fonction d'activation *Rectified Linear Unit* (ReLU) [Nair and Hinton, 2010] :

$$\text{FFN}(X) = \max(0, XW_1)W_2 \tag{2.22}$$

où $W_1 \in \mathbb{R}^{d \times d_{\text{FFN}}}$ et $W_2 \in \mathbb{R}^{d_{\text{FFN}} \times d}$ sont deux matrices de paramètres à apprendre. La dimension interne d_{FFN} est un hyper-paramètre à choisir. L'opération d'auto-attention à plusieurs têtes est définie par

$$\text{AttentionMultiTêtes}(X) = [\text{Attention}_1(X), \ldots, \text{Attention}_N(X)]W^O \tag{2.23}$$

où N est le nombre de têtes à choisir et $W^O \in \mathbb{R}^{Nd_a \times d}$ une matrice de paramètres à apprendre qui projette le résultat de l'opération d'auto-attention à plusieurs têtes vers la dimension d'origine de X. d_a est la dimension de sortie de l'opération d'attention. La n-ème opération d'attention est définie par

$$K_n = W_{K,n}X \tag{2.24}$$
$$Q_n = W_{Q,n}X \tag{2.25}$$
$$V_n = W_{V,n}X \tag{2.26}$$
$$\text{Attention}_n(X) = \text{Softmax}\left(\frac{Q_n K_n^\top}{\sqrt{d_k}}\right) V_n \tag{2.27}$$

où $W_{Q,n}, W_{K,n} \in \mathbb{R}^{d \times d_k}$ et $W_{V,n} \in \mathbb{R}^{d \times d_a}$ sont des matrices de paramètres à apprendre. On choisit $d_k = d_a = d/N$.

On introduit également le mécanisme de *dropout* de probabilité p après les opérations de normalisation des équations (2.20) et (2.21). La Fig. 2.16 résume les composants et les hyper-paramètres à choisir d'une couche Transformer.

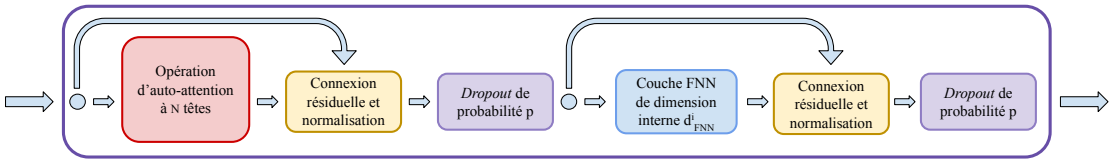


FIGURE 2.16. – Composition d'une couche Transformer.

La structure d'un réseau entier n'utilise pas que des couches Transformer. Pour les premières couches, n_1 couches convolutionelles de l'architecture *Gated ConvNet* sont utilisées. Celles-ci prennent le rôle d'extraction de caractéristiques et elles permettent de diminuer le nombre de trames avec le *stride*. Nous réglons n_1 à 4 ou 6 avec l'utilisation d'un *stride* sw $= 2$ une couche sur deux. Ensuite nous ajoutons n_2 couches Transformer puis nous finissons par une couche linéaire. Pour régler n_2, l'ordre de grandeur est plutôt entre 10 et 30.

2.3.3. Formulation de la fonction de coût

Soit Θ l'ensemble des paramètres du DNN. Le but de l'apprentissage est de régler Θ en minimisant la fonction de coût $L(f_\Theta(X), Y)$, où $f_\Theta(X)$ est la sortie du DNN et

(X, Y) est une paire composée du signal acoustique et de sa transcription de référence. La formulation de cette fonction est donc directement liée à la tâche que doit résoudre notre modèle. Le but de cette thèse est de prendre en compte les erreurs dans les données textuelles. Notre solution se base sur la formulation d'une fonction de coût adéquate.

Il y a plusieurs catégories de fonctions de coût. Selon les configurations, il peut être nécessaire d'obtenir au préalable un alignement entre le signal acoustique et la transcription de référence. C'est-à-dire qu'il faut trouver à quels moments sont prononcées les différentes portions de la transcription. Pour les HMM il faut effectuer cette étape. On peut alors formuler une fonction de coût opérant sur chaque trame individuellement en utilisant le maximum de vraisemblance (*Maximum Likelihood*, ML) [Bahl et al., 1983] ou l'entropie croisée (*Cross-Entropy*, CE). Mais la RAP est fondamentalement un problème de classification de séquences, ainsi cela fonctionne mieux d'utiliser une fonction de coût opérant sur toute la séquence. Dans cette catégorie, nous avons les fonctions de coût MMI [Bahl et al., 1986, Kapadia et al., 1993], *Maximum Mutual Information*, et LF-MMI [Hadian et al., 2018], *Latice-Free Maximum Mutual Information*. Celles-ci ont aussi pour particularités de non seulement maximiser la probabilité de la transcription de référence mais aussi de minimiser les probabilités de toutes les autres transcriptions. Il y a aussi MBR [Gibson and Hain, 2006, Veselý et al., 2013] (*Minimum Bayes Risk*) qui permet d'inclure des informations à différentes granularités de la transcription.

Pour les modèles DNN, nous préférons dans cette thèse de ne pas être dépendant d'une étape d'alignement. Les modèles récurrents n'ont pas besoin d'un tel alignement et la fonction de coût d'entropie croisée peut aussi être utilisée directement. Les modèles *Gated ConvNet* et *Transformer* que nous utilisons n'ont par contre pas cette particularité. Les fonctions de coût CTC [Graves et al., 2006] et ASG [Collobert et al., 2016] qui opèrent sur toute la séquence permettent de résoudre ce problème. Dans cette partie, nous décrivons avec précision les fonctions de coût CTC et ASG utilisées comme bases de notre travail.

2.3.3.1. La fonction de coût CTC

Principe La fonction de coût CTC [Graves et al., 2006] a pour but de maximiser la vraisemblance de la transcription Y sachant le signal acoustique X en sortie du DNN. Le signal acoustique X est de dimension $\mathbb{R}^{T\times M}$, avec T le nombre de trames et M le nombre de symboles utilisés. La transcription $Y = [y_1, ..., y_L]$ est une séquence de symboles de taille L. Au lieu de maximiser on va plutôt minimiser l'inverse et composer par la fonction logarithme. Sur ordinateur on ne stocke qu'une approximation des nombres flottants et l'utilisation de la fonction logarithme permet d'éviter des problèmes de précision lors de la manipulation de la fonction de coût. Ainsi la fonction de coût CTC est définie par :

$$\mathrm{CTC}(Y) = -\log p(Y|X). \tag{2.28}$$

Système d'alignement Nous avons ici un problème : Y est de taille L et X possède T trames. Il n'est donc pas immédiat de calculer $p(Y|X)$. Dans la partie 2.1.2 nous avons

créé des trames de 25 millisecondes toutes les 10 millisecondes. Il y a donc 100 trames par seconde. Ensuite, dans la partie 2.3.2, nous avons précisé que l'on pouvait réduire le nombre de trames par deux plusieurs fois avec le *stride*. Prenons les lettres comme symboles : avec un débit moyen de 200 mots par minute et des mots d'une longueur moyenne de 5 lettres, on a environ 16 lettres par seconde qui sont prononcées. Pour les lettres on peut utiliser le *stride* deux fois afin obtenir 25 trames par seconde $(100/(2*2))$. Ce que l'on veut, c'est associer une ou plusieurs trames avec un symbole. On appelle cela l'alignement entre X et Y. Remarquons qu'il faut supposer que $L \leqslant T$ et donc que CTC n'est pas adapté pour tous les problèmes de séquence à séquence.

Un alignement $\pi = [\pi_1, \ldots, \pi_T]$ est une séquence de symboles de taille T. Cet alignement encode la transcription Y. Pour retrouver Y à partir de π il y a certaines règles pour CTC :

— Si une sous-séquence $\pi_k, \ldots, \pi_{k+l-1}$ de longueur l est composée du même symbole, alors ce symbole est dans Y et il est associé aux l trames correspondantes de X.

— On se rend compte qu'avec seulement cette règle on ne peut pas encoder d'alignement lorsque Y a deux symboles identiques de suite. Pour cela on introduit le symbole blanc β. C'est un symbole qui ne correspond à rien dans la transcription Y, il est simplement enlevé après application de la première règle. Il peut donc servir à séparer des symboles identiques. Il peut également modéliser un moment de silence du signal acoustique. On peut voir un exemple de ces deux règles dans la Fig. 2.17.

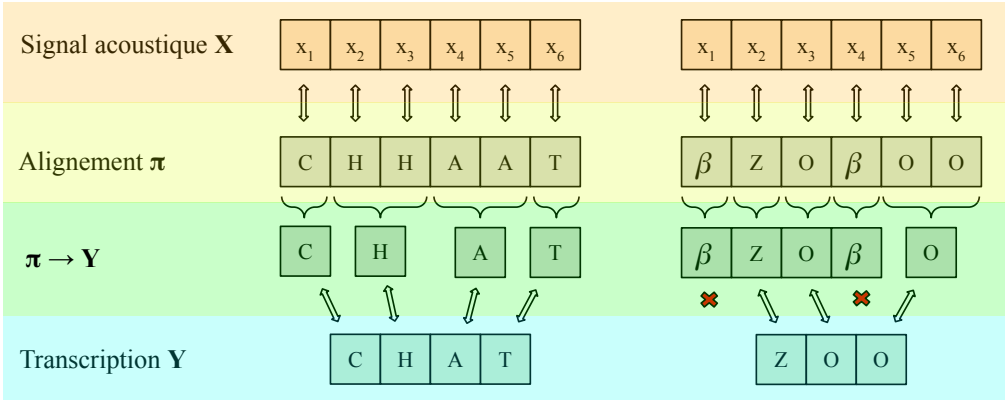


FIGURE 2.17. – Système d'alignement utilisé pour CTC.

Formulation La difficulté est que l'on n'a pas connaissance de cet alignement π. Au lieu d'avoir à chercher un alignement précis entre la parole et la transcription, CTC prend en compte tous les alignements autorisés entre la transcription Y de taille L et les T trames

du signal acoustique X :

$$\mathrm{CTC}(Y) = -\log \sum_{\pi \in G_{\mathrm{ctc}}(Y,T)} p(\pi|X), \tag{2.29}$$

avec G_{ctc} un graphe orienté acyclique dont chaque chemin π représente un alignement autorisé. $p(\pi|X)$ est la vraisemblance d'un tel chemin. CTC utilise l'hypothèse d'indépendance conditionnelle entre les états d'un même chemin. Ainsi pour un chemin π sur les T trames $\pi = [\pi_1, \pi_2, ..., \pi_T]$ on a :

$$p(\pi|X) = p(\pi_1, \pi_2, ..., \pi_T|X) = \prod_{t=1}^{T} p(\pi_t|X) = \prod_{t=1}^{T} e^{\log p(\pi_t|X)}. \tag{2.30}$$

La couche finale d'un DNN fournit les scores $s_{\pi_t}(X)$. Il faut normaliser ces scores avec la fonction softmax pour qu'ils représentent des probabilités. On manipulera plus facilement des log-probabilités et on les note $f_{\pi_t}(X)$:

$$f_{\pi_t}(X) = \log p(\pi_t|X) = \log \frac{e^{s_{\pi_t}(X)}}{\sum_{k=1}^{M} e^{s_{\pi_k}(X)}} = s_{\pi_t}(X) - \log \left(\sum_{k=1}^{M} e^{s_{\pi_k}(X)} \right). \tag{2.31}$$

On peut également introduire la notation $f_{\pi}(X)$:

$$f_{\pi}(X) = \log p(\pi|X) = \log \prod_{t=1}^{T} p(\pi_t|X) = \sum_{t=1}^{T} \log p(\pi_t|X) = \sum_{t=1}^{T} f_{\pi_t}(X). \tag{2.32}$$

Ce qui signifie que la log-probabilité d'un chemin est égale à la somme des log-probabilités de chaque état du chemin. L'équation (2.29) devient :

$$\begin{aligned} \mathrm{CTC}(Y) &= -\log \sum_{\pi \in G_{\mathrm{ctc}}(Y,T)} e^{f_{\pi}(X)} && (2.33) \\ &= - \operatorname*{logadd}_{\pi \in G_{\mathrm{ctc}}(Y,T)} f_{\pi}(X) && (2.34) \\ &= - \operatorname*{logadd}_{\pi \in G_{\mathrm{ctc}}(Y,T)} \sum_{t=1}^{T} f_{\pi_t}(X). && (2.35) \end{aligned}$$

Finalement comme l'indique l'équation (2.34), pour finaliser le calcul de CTC il faut agréger toutes les log-probabilités des chemins autorisés du graphe G_{ctc} par la fonction logadd. Cette dernière est définie précisément dans l'annexe A.

La fonction de coût CTC dépend de la sortie du DNN via les équations (2.35) et (2.31). Minimiser cette fonction de coût permet bien de régler les paramètres Θ. Le détail de cette étape est expliqué dans la partie 2.3.4.

Pour calculer CTC(Y) il y a deux méthodes. Classiquement, la fonction de coût est implémentée avec un algorithme de programmation dynamique. Il sera présenté dans le chapitre 3 dans un cadre plus général. Au vu de la formulation de l'équation (2.35) nous avons une autre implémentation immédiate en utilisant les wFST. En effet, nous pouvons utiliser l'opération Score(.) définie dans l'équation (2.8) sur le graphe $G_{\text{ctc}}(Y, T)$. La création de ce graphe sera expliquée dans le cadre du chapitre 4.

2.3.3.2. La fonction de coût ASG

Principe La fonction de coût ASG [Collobert et al., 2016] est similaire à CTC puisqu'il ne faut pas non plus modéliser d'alignements explicites pour la construire. Il faut noter les différences suivantes :

— La fonction de coût inclut un modèle bi-gramme de transition de symboles au niveau de la trame.

— Le système d'alignement change. On n'utilise pas le symbole blanc β ce qui crée moins de possibilités d'alignement.

— Les scores ne sont pas normalisés au niveau de la trame comme avec CTC (équation (2.31)) mais au niveau de la séquence entière.

Comme pour CTC, on a :

$$\text{ASG}(Y) = -\log p(Y|X). \tag{2.36}$$

Système d'alignement Pour modéliser deux symboles qui se suivent, on utilise le symbole R_1 pour indiquer que l'on répète le symbole précédent une fois. Par exemple, le mot « zoo » encodé avec des lettres devient $Y = [z, o, R_1]$. Un exemple est donné dans la Fig. 2.18. L'article original introduit aussi le symbole R_2 pour répéter deux fois le symbole précédent mais nous ne l'exploitons pas car il est très rarement utilisé.

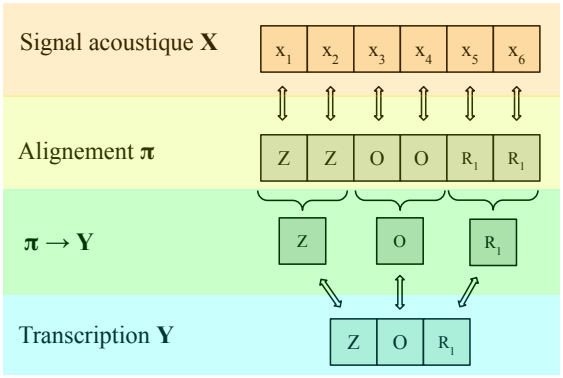


FIGURE 2.18. – Système d'alignement utilisé pour ASG.

Formulation Comme CTC, on prend en compte tous les alignements possibles, cette fois-ci selon les règles d'alignement d'ASG :

$$\mathrm{ASG}(Y) = -\log \sum_{\pi \in G_{\mathrm{asg}}(Y,T)} p(\pi|X). \tag{2.37}$$

Pour ASG, $p(\pi|X)$ est obtenue en normalisant sur toute la séquence et non pas par trame. Soit $z_\pi(X)$ le score non normalisé associé. Pour obtenir $p(\pi|X)$, on applique la fonction softmax mais sur l'ensemble de toutes les séquences de symboles possibles sur T trames :

$$p(\pi|X) = \frac{e^{z_\pi(X)}}{\sum_{\dot{\pi} \in G_{\mathrm{full}}(T)} e^{z_{\dot{\pi}}(X)}}, \tag{2.38}$$

avec $G_{\mathrm{full}}(T)$ un graphe orienté acyclique dont les chemins sont l'ensemble de toutes les séquences de symboles de longueurs T. On obtient alors :

$$\begin{aligned}
\mathrm{ASG}(Y) &= -\log \sum_{\pi \in G_{\mathrm{asg}}(Y,T)} \frac{e^{z_\pi(X)}}{\sum_{\dot{\pi} \in G_{\mathrm{full}}(T)} e^{z_{\dot{\pi}}(X)}} && (2.39)\\
&= -\log \frac{\sum_{\pi \in G_{\mathrm{asg}}(Y,T)} e^{z_\pi(X)}}{\sum_{\dot{\pi} \in G_{\mathrm{full}}(T)} e^{z_{\dot{\pi}}(X)}} && (2.40)\\
&= \log \sum_{\pi \in G_{\mathrm{full}}(T)} e^{z_\pi(X)} - \log \sum_{\pi \in G_{\mathrm{asg}}(Y,T)} e^{z_\pi(X)} && (2.41)\\
&= \operatorname*{logadd}_{\pi \in G_{\mathrm{full}}(T)} z_\pi(X) - \operatorname*{logadd}_{\pi \in G_{\mathrm{asg}}(Y,T)} z_\pi(X). && (2.42)
\end{aligned}$$

Formulons maintenant $z_\pi(X)$, le score non-normalisé d'une séquence $\pi = [\pi_1, \ldots, \pi_T]$. Ce score est défini par :

$$z_\pi(X) = \sum_{t=1}^{T} s_{\pi_t}(X) + g_{\pi_{t-1},\pi_t}, \tag{2.43}$$

avec $s_{\pi_t}(X)$ les scores des symboles fournis par le DNN et $g_{i,j}$ un score de transition du symbole i au symbole j. C'est le modèle bi-gramme de transition. L'ensemble de ces scores $g_{i,j}$ sont des paramètres que nous regroupons dans l'ensemble Θ_{ASG}. Ce sont des nouveaux paramètres qu'il faut conjointement régler avec Θ en minimisant la fonction de coût pendant la phase d'apprentissage.

Comme pour CTC, la fonction de coût ASG peut être implémentée avec un algorithme dynamique ou avec des wFST. Cela sera détaillé dans les chapitres 3 et 4.

2.3.4. Algorithmes d'apprentissage

Utilisation d'un jeu de données Soit $L(f_\Theta(X), Y)$ une fonction de coût pour une paire de signal acoustique X et sa transcription de référence Y. La paire (X, Y) est tirée d'une distribution de probabilité inconnue p_{data}. Idéalement, on veut minimiser

$$J(\Theta) = \mathbb{E}_{(X,Y)\sim p_{\mathrm{data}}} L(f_\Theta(X), Y). \tag{2.44}$$

Mais la distribution p_{data} est inconnue. On a par contre un jeu de données composé de m paires (X_i, Y_i) qui forment une distribution empirique $\hat{p}_{\text{data}}$. On peut alors minimiser

$$\hat{J}(\Theta) = \mathbb{E}_{(X,Y)\sim\hat{p}_{\text{data}}} L(f_\Theta(X), Y) = \frac{1}{m}\sum_{i=1}^{m} L(f_\Theta(X_i), Y_i). \tag{2.45}$$

La descente de gradient Pour minimiser $\hat{J}(\Theta)$, l'algorithme le plus simple est celui de la descente de gradient. Il consiste à répéter l'opération suivante un certain nombre de fois jusqu'à ce qu'un critère soit rempli :

$$\Theta \leftarrow \Theta - \alpha\nabla_\Theta \hat{J}(\Theta), \tag{2.46}$$

où α est un hyper-paramètre appelé « pas d'apprentissage » ou *learning rate*. L'équation (2.46) signifie que l'on prend la direction de la plus grande pente de la surface $\hat{J}(.)$ au point Θ, et que l'on déplace ce point Θ dans la direction inverse d'un pas α.

Calcul du gradient Le calcul automatique du gradient est l'atout principal des bibliothèques de *deep learning*. Le principe est exactement le même que pour les wFST différentiables. Lorsque l'architecture est définie, la bibliothèque indique comment chaque poids est obtenu et comment calculer son gradient par rapport aux poids de la couche précédente. Pendant la phase de propagation avant, on calcule la sortie du DNN et la fonction de coût. Puis pendant la phase de propagation arrière, on rétro-propage le gradient jusqu'à la première couche du DNN pour obtenir le gradient de la fonction de coût par rapport à tous les poids du DNN, c'est l'équivalent de l'application de l'équation (2.13) des wFST.

Critère d'arrêt de l'algorithme Pour savoir quand arrêter les itérations de l'équation (2.46) on observe l'évolution de la fonction de coût au fur et à mesure des itérations. Il faut faire attention au problème de sur-apprentissage (*overfitting*) : on peut arriver facilement dans la situation où l'on donne au modèle la capacité de modéliser parfaitement le jeu de données utilisé pour l'apprentissage mais que face à une nouvelle donnée jamais vue auparavant il soit mauvais. Pour éviter cela, on utilise un jeu de données de développement qui n'est pas utilisé pour l'apprentissage. On calcule la valeur que prend la fonction de coût sur ce jeu de données et on arrête l'apprentissage lorsque cette valeur est au plus bas.

La descente de gradient stochastique par paquet Le problème de l'algorithme de descente de gradient classique est qu'il est très lent. Il faut en effet calculer $\hat{J}(\Theta)$ à chaque itération sur le jeu de données de taille m. À la place, on va utiliser un paquet (ou *batch*) de b exemples pour calculer $\hat{J}(\Theta)$ à chaque itération. C'est l'algorithme *Stochastic Gradient Descent* (SGD). b est un hyper-paramètre qu'il faut choisir de façon à pouvoir effectuer le calcul de la fonction de coût sur les b exemples en parallèle. Avec SGD on obtient un gradient approché du vrai gradient pour un temps de calcul beaucoup plus faible.

Algorithmes avancés La surface $\hat{J}(.)$ n'est pas convexe. Il y a un risque que l'algorithme se bloque sur un plateau ou un minimum local. Pour éviter ce phénomène, beaucoup d'algorithmes d'optimisation pour les DNN incluent une notion de « moment » paramétré par β_1. De plus, dans la version classique de SGD le paramètre α est constant pour toutes les directions de Θ. Il est préférable que celui-ci s'adapte pour chaque dimension. On doit toujours choisir un α initial mais le paramètre devient moins critique pour l'apprentissage. Parmi les algorithmes qui incluent ces améliorations il y a Adagrad [Duchi et al., 2011], RMSProp [Hinton et al., 2012], AdaDelta [Zeiler, 2012]. L'algorithme Adam [Kingma and Ba, 2014] utilise en plus un moment d'ordre 2 paramétré par β_2. Dans ce manuscrit nous utiliserons SGD et Adam.

Astuces d'apprentissage L'apprentissage d'un modèle DNN peut être délicat. Il faut éviter les problèmes de sur-apprentissage. Il y a les hyper-paramètres de l'architecture du modèle à choisir et ceux de l'algorithme d'apprentissage. Le *deep learning* est l'ensemble des pratiques et algorithmes permettant à ce qu'un modèle DNN fonctionne. Par exemple, dans l'architecture elle-même, nous avons utilisé le *dropout* qui a un effet de régularisation pour éviter ce sur-apprentissage. On a aussi des connexions résiduelles et la normalisation des poids pour permettre à la rétro-propagation du gradient d'être efficace sur des architectures comprenant beaucoup de couches.

En plus des astuces ci-dessus, nous normalisons le gradient pour qu'il ne puisse pas être trop grand [Liptchinsky et al., 2017] :

$$\text{si } \left\|\nabla\hat{J}\right\| \geq c \text{ alors } \nabla\hat{J} \leftarrow c \times \frac{\nabla\hat{J}}{\left\|\nabla\hat{J}\right\|}, \tag{2.47}$$

avec c un hyper-paramètre à choisir. Pour utiliser l'architecture Transformer nous faisons comme Synnaeve et al. [2019]. Le pas d'apprentissage augmente linéairement de 0 à α pendant les $n_{\text{warm-up}}$ premières itérations. Aussi lorsque le critère d'arrêt est atteint, α est divisé par deux puis la phase d'apprentissage reprend. Nous faisons cette étape une ou deux fois. Enfin, pour l'architecture Transformer nous incluons parfois une couche SpecAugment [Park et al., 2019] comme première couche de l'architecture. Cette dernière a un effet de régularisation.

Implémentation L'autre intérêt des bibliothèques de *deep learning* est que toutes ces étapes s'effectuent sur un ou plusieurs GPU. Nous utilisons pour cela la bibliothèque Flashlight en C++. Utiliser plusieurs GPU permet de choisir une taille de *batch* plus grande. L'approximation de gradient est alors meilleure ce qui permet d'augmenter le pas d'apprentissage. Ainsi le critère d'arrêt est atteint plus rapidement. Les GPU sont répartis par nœud ; il y a plusieurs GPU sur un nœud. La bibliothèque permet d'utiliser des GPU répartis sur plusieurs nœuds pour un seul apprentissage.

2.4. Décodage du modèle acoustique

L'étape du décodage (ou inférence) s'effectue après l'apprentissage du modèle. On considère à cette étape que l'on a un modèle DNN modélisant $p(Y|X)$. Pour utiliser notre modèle de RAP sur un signal acoustique nouveau X, il faut reprendre l'équation (2.4) pour trouver Y :

$$Y^* = \underset{Y}{\operatorname{argmax}}\, P(Y|X). \tag{2.48}$$

2.4.1. Décodeurs

Méthode rapide Pour CTC on peut utiliser une heuristique simple qui consiste à chercher l'alignement π avec le plus grand score :

$$\pi^* = \underset{\pi \in G_{\text{full}}}{\operatorname{argmax}}\, f_\pi(X) = \underset{\pi \in G_{\text{full}}}{\operatorname{argmax}} \sum_{t=1}^{T} f_{\pi_t}(X). \tag{2.49}$$

L'alignement est alors obtenu en prenant l'état π_t qui a le plus grand score pour chaque instant t. On décode ensuite cet alignement pour obtenir la transcription Y associée. Cette méthode est rapide à mettre en place, elle permet d'évaluer le modèle pendant l'apprentissage sur la qualité des transcriptions obtenues et non pas sur la simple valeur de la fonction de coût. Elle est aussi adaptable pour ASG en intégrant les scores de transition. Il faut alors utiliser l'algorithme de Viterbi pour trouver le meilleur alignement.

La recherche en faisceau Pour améliorer la qualité de la transcription, on inclut d'autres sources d'informations pendant cette étape de décodage. On ajoute un modèle de langage $P(Y)$ qui indique la probabilité qu'une phrase soit effectivement prononcée. On utilise aussi un lexique qui force la transcription à n'être composée que de mots appartenant à ce lexique. Celui-ci est composé de 200 000 mots et il est fourni avec le jeu de données Librispeech [Panayotov et al., 2015]. Comme Collobert et al. [2016], nous cherchons à résoudre :

$$Y^* = \underset{Y \in \text{Lexique}}{\operatorname{argmax}}\, P(Y|X) \cdot P(Y)^{\alpha_{\text{LM}}} \cdot |Y|^{\alpha_{\text{mot}}}, \tag{2.50}$$

où $|Y|$ est le nombre de mots dans la transcription Y, α_{LM} un paramètre qui règle la contribution du modèle de langage et α_{mot} le paramètre qui règle le bonus d'insertion de mots. Pour obtenir une bonne solution approchée, on utilise un algorithme de recherche en faisceau. L'algorithme fonctionne en faisant des itérations sur les T trames. La probabilité du modèle de langage n'est pas inclue à chaque trame mais uniquement lorsque l'on peut former un mot. Cela a pour conséquence de favoriser les mots courts. L'ajout du terme $|Y|^{\alpha_{\text{mot}}}$ permet d'éviter ce phénomène. Un algorithme similaire est présenté dans le chapitre 3.

La bibliothèque Kaldi [Povey et al., 2011] fait le choix d'implémenter cette étape en utilisant des wFST [Povey et al., 2012]. Le décodeur de Flashlight est directement programmé en C++.

Le modèle de langage Comme modèle de langage, nous utilisons un n-gramme d'ordre 4 au niveau du mot. C'est-à-dire que pour tous les quadruplets de mots (w_1, w_2, w_3, w_4), nous avons accès à la probabilité $p(w_1|w_2, w_3, w_4)$, et cela pour les 200 000 mots du lexique. Ce modèle est implémenté par la bibliothèque KenLM [Heafield, 2011, Heafield et al., 2013] qui utilise la méthode de lissage de Kneser-Ney [Kneser and Ney, 1995]. Le modèle est appris avec un jeu de données textuelles de 14 500 livres fournis avec Librispeech.

2.4.2. Métrique

Pour évaluer le modèle acoustique nous faisons l'étape de décodage sur les ensembles de développement ou de test. Ensuite nous mesurons le *Word Error Rate* (WER) et le *Letter Error Rate* (LER). Ces métriques représentent le pourcentage de mots ou de lettres à modifier pour passer d'une transcription de référence Y à la transcription obtenue après le décodage Y^*. C'est en fait la distance de Levenshtein [Levenshtein et al., 1966] au niveau du mot ou de la lettre. Elle s'obtient en appliquant un algorithme de programmation dynamique [Navarro, 2001] et vaut :

$$\text{Levenshtein}(Y, Y^*) = \frac{n_{\text{del}}(Y, Y^*) + n_{\text{ins}}(Y, Y^*) + n_{\text{sub}}(Y, Y^*)}{|Y|} \times 100, \tag{2.51}$$

où $|Y|$ est le nombre de mots ou de lettres dans Y et $n_{\text{del}}(Y, Y^*)$, $n_{\text{ins}}(Y, Y^*)$ et $n_{\text{sub}}(Y, Y^*)$ sont le nombre minimal de délétions, d'insertions et de substitutions de mots ou de lettres pour transformer la transcription source Y en Y^*.

2.5. Apprentissage à partir d'étiquettes bruitées

Pour utiliser un algorithme d'apprentissage classique, il faut un jeu de données qui apparie une donnée d'entrée et un étiquetage : pour la RAP ce sont respectivement le signal acoustique X et la transcription Y. C'est de l'apprentissage supervisé. Pour faire de la classification d'image, le signal d'entrée est une image et l'étiquette est une classe qui indique ce qui est représenté sur cette image. Construire un tel jeu de données n'est pas évident. Obtenir l'étiquetage requiert souvent un effort humain important ou alors utilise une méthode automatique sujette à erreurs. Dans tous les cas, ce processus est sujet à des erreurs et implique que les données sont parfois mal étiquetées. On dira que les étiquettes sont bruitées. Cela a des conséquences négatives directes sur les performances des modèles appris sur de telles données [Frénay and Verleysen, 2013, Sukhbaatar et al., 2014]. Dans cette partie, nous répertorions quels axes de recherche de la littérature visent à résoudre ce problème. Nous commençons par le domaine de la vision car il y a beaucoup plus de références dans celui-ci sur les étiquettes bruitées par rapport à la RAP. Nous pouvons ainsi nous inspirer des méthodes de ce domaine pour les adapter à la RAP.

2.5.1. Étiquettes bruitées pour la reconnaissance d'image

Beaucoup de travaux existent pour résoudre ce problème dans le domaine de la vision [Song et al., 2020]. Les méthodes se divisent globalement en trois catégories : la modification de la fonction de coût, la modification de la stratégie d'apprentissage ou l'utilisation de méthodes de correction des étiquettes.

Modification de la fonction de coût La plupart des fonctions de coût prenant en compte les étiquettes bruitées modélisent ce bruit avec une matrice de confusion des classes [Kaneko et al., 2019, Lee et al., 2018, Sukhbaatar et al., 2014]. Cette dernière, que l'on appellera aussi modèle de bruit, est parfois apprise conjointement avec le modèle qui résout la tâche [Yi and Wu, 2019]. Avec c^* la vraie étiquette (ou classe) et $\tilde{c}$ la classe incorrectement attribuée à une donnée d'entrée, le modèle de bruit est défini par $p(\tilde{c} = j|c^* = i)$ pour chaque paire (i, j) d'indices de classe. Ainsi on peut écrire :

$$p(\tilde{c} = j|X) = \sum_i p(\tilde{c} = j|c^* = i)p(c^* = i|X). \tag{2.52}$$

Cette équation sert à créer une fonction de coût à partir d'une donnée appariée $(X, \tilde{c})$ incorrectement étiquetée. Le modèle de la tâche à résoudre $p(c^* = i|X)$ est alors conçu pour classifier X avec la vraie classe c^*. Il existe des résultats théoriques de modèles appris avec de telles fonctions de coûts [Ghosh et al., 2017, Ma et al., 2018, Patrini et al., 2017]. Ici, le modèle de bruit ne dépend pas de X. Vahdat [2017] étudie les dépendances possibles du modèle de bruit et Khetan et al. [2018] font dépendre le modèle de bruit de l'annotateur qui a étiqueté la donnée. Ainsi on peut attribuer des niveaux de confiance différents en fonction de l'annotateur. À noter que l'on peut utiliser des données bruitées et des données considérées comme bonnes [Hendrycks et al., 2018, Lee et al., 2018].

Il existe d'autres alternatives. Ma et al. [2020a] combinent plusieurs fonctions de coût pour en obtenir une nouvelle fonction robuste au bruit. On peut aussi paramétrer une fonction de coût afin d'obtenir un résultat similaire [Wang et al., 2019, Zhang and Sabuncu, 2018].

Modification de la stratégie d'apprentissage Certains travaux s'adaptent à la présence d'étiquettes bruitées en modifiant la façon dont elles sont utilisées. Rolnick et al. [2017] utilisent simplement plus de données pour réduire l'effet des étiquettes bruitées. Li et al. [2019] modifient l'étape de descente du gradient en évitant l'effet de surapprentissage du bruit des étiquettes. Xie et al. [2016] proposent une méthode de régularisation consistant à permuter aléatoirement certaines étiquettes du jeu de données.

Des travaux utilisent conjointement plusieurs modèles pour affiner la stratégie d'apprentissage. Jiang et al. [2018] utilisent un DNN qui doit informer au classifieur d'images quelles sont les étiquettes qui sont probablement correctes. Han et al. [2018] introduisent le co-apprentissage de deux DNN qui ont la même tâche mais s'informent l'un l'autre de la présence des étiquettes bruitées.

Méthodes de correction des étiquettes Veit et al. [2017] utilisent un jeu de données d'étiquettes bruitées et un jeu de données considéré sans erreurs pour entraîner un DNN dont le but est de corriger les erreurs. Tanaka et al. [2018] utilisent le classifieur non seulement pour estimer les probabilités a posteriori des classes mais aussi pour extraire des informations concernant les erreurs des étiquettes et ainsi les corriger. D'autres travaux utilisent un modèle externe qui n'est pas un DNN pour corriger les erreurs, tel qu'un modèle graphique [Xiao et al., 2015], un graphe de connaissance [Li et al., 2017] ou encore un CRF (*Conditional Random Field*) [Vahdat, 2017].

2.5.2. Étiquettes bruitées pour la RAP

Contrairement au domaine de la vision, il y a peu de travaux prenant en compte les étiquettes bruitées (les transcriptions) pour la RAP. Nous pouvons tout de même séparer les différentes méthodes selon les mêmes catégories que pour la vision.

Sources d'erreurs de transcription Les erreurs peuvent provenir d'erreurs humaines lors de la phase de transcription d'un jeu de données. Un des objectifs possibles de la RAP est d'atteindre le niveau humain de reconnaissance de la parole. Des travaux mesurent ainsi le WER atteint par les humains afin de le comparer à un modèle de RAP. Des travaux préliminaires estiment ce WER à 4% [Lippmann, 1997]. Ce taux est sous-estimé et d'autres travaux l'estiment plutôt entre 5 et 7 % [Saon et al., 2017, Stolcke and Droppo, 2017, Xiong et al., 2017b]. Pour des transcriptions faites « rapidement », ce taux augmente à 9,6% [Glenn et al., 2010].

Nous pouvons aussi les retrouver dans d'autres situations. Les sous-titres de contenus vidéo sont adaptés pour que la lecture soit fluide et ne correspondent pas toujours à ce qui est dit. Nous pouvons aussi considérer les transcriptions produites par un modèle de RAP comme étant bruitées pour les utiliser à nouveau pendant une phase d'apprentissage. C'est une forme d'apprentissage semi-supervisé (auto-apprentissage) qui consiste à créer des « pseudo-transcriptions » d'un jeu de données initialement non transcrit [Kahn et al., 2020, Likhomanenko et al., 2020, Xu et al., 2020].

L'approche de Hasegawa-Johnson et al. [2016] est originale. Ils considèrent deux autres sources d'erreurs possibles. Pour la première, ils demandent à des travailleurs de transcrire de la parole qui est prononcée dans une langue qu'ils ne connaissent pas. Pour la seconde, les travailleurs doivent écouter de la parole dans une langue qu'ils ne connaissent pas non plus. Des capteurs enregistrent les ondes cérébrales (EEG) émises pendant l'exercice et celles-ci sont utilisées à la place des transcriptions. Les transcriptions générées dans ce travail ne sont pas des suites de mots mais des distributions de probabilités de mots.

Modification de la fonction de coût Nous n'avons pas connaissance de travaux modélisant directement les erreurs de transcriptions dans une fonction de coût. Pour la RAP, les étiquettes ne sont pas des classes d'image mais des transcriptions composées d'une

succession de mots. En reconnaissance d'image, il y a un nombre réduit de classes possibles. Mais pour la RAP, il y a autant de transcriptions possibles qu'il y a de façons d'ordonner les mots et cela pour un nombre variable de mots. Nous ne pouvons donc pas utiliser un modèle de bruit de transcription, tel que représenté par l'équation (2.52), pour toutes les transcriptions possibles. Les chapitres 3 et 4 de cette thèse explorent cet axe de recherche.

Modification de la stratégie d'apprentissage Ling et al. [2021] proposent une stratégie de modification du gradient pour l'apprentissage de modèles de RAP sur des pseudo-transcriptions. Le domaine du traitement du langage naturel (*Natural Language Processing*, NLP) est un problème de séquence comme la RAP. Les performances sont aussi impactées par les erreurs dans le texte en entrée des modèles [Moradi and Samwald, 2021]. Piktus et al. [2019] présentent une méthode de projection des mots robuste aux erreurs. Namysl et al. [2020] proposent une méthode d'augmentation des données pour améliorer la robustesse de leur modèle aux erreurs dans le texte d'entrée.

Méthodes de correction des transcriptions Des méthodes existent pour corriger les transcriptions produites par un modèle de RAP [Errattahi et al., 2018]. Le problème n'est que peu étudié en amont pour corriger les transcriptions du jeu de données d'apprentissage. Guo et al. [2019] utilisent la distribution de probabilité d'un modèle de RAP pour créer, à l'aide de données textuelles, un modèle de correction d'erreurs. Sarma and Palmer [2004] utilisent une méthode non-supervisée pour corriger les erreurs par une analyse de co-occurrence des mots fournis par le modèle de RAP par rapport à un grand corpus de texte. Bassil and Semaan [2012] présentent une méthode pour d'abord détecter les erreurs avec un modèle N-gramme, puis ensuite proposer des meilleurs candidats à partir d'un algorithme de détection d'erreurs prenant en compte le contexte.

2.5.3. Conclusion

Si les modèles DNN sont largement utilisés aujourd'hui, c'est grâce à leur grande capacité de modélisation. Mais ces modèles arrivent tellement bien à modéliser les données qu'ils modélisent aussi les erreurs d'étiquettes. On observe ainsi un effet de sur-apprentissage sur les erreurs qui n'est pas désiré et cela a un réel impact sur la qualité des modèles DNN. Le domaine de la vision, qui est précurseur des modèles DNN, a en premier observé cet effet et des travaux conséquents ont répondu à ce problème. La RAP bénéficie maintenant de la bonne capacité de modélisation des DNN. Au fur et à mesure que les performances de ceux-ci s'améliorent, l'impact relatif des erreurs de transcription augmente. Nous sommes donc à un stade où il est encore plus pertinent de corriger ce problème. Jusqu'à présent, les travaux portant sur la prise en compte des erreurs pour limiter leurs impacts sur l'apprentissage sont peu nombreux.

Nous avons identifié trois catégories de méthodes possibles pour prendre en compte les erreurs de transcription. Parmis elles, la stratégie de modification de la fonction de

coût n'est pas encore étudiée dans le domaine de la RAP. Les modèles DNN ont souvent pour particularité de répondre au problème posé avec un seul « grand » modèle, de l'extraction des caractéristiques à la création des probabilités de sortie. Cela est aussi appelé apprentissage de « bout en bout ». Nous pensons que la stratégie de modification de la fonction de coût est compatible pour créer une nouvelle méthode de bout en bout. C'est pourquoi nous prenons cette direction dans cette thèse.

3. L'algorithme Lead2gold

Dans ce chapitre nous proposons l'algorithme Lead2Gold [Dufraux et al., 2019]. Cet algorithme repose sur une fonction de coût qui exploite le fait que les transcriptions peuvent être mal étiquetées. Il se base sur l'exploitation d'un modèle de bruit de transcription $p(\tilde{Y}|Y^*)$. À l'aide de ce modèle de bruit, nous cherchons pendant la phase d'apprentissage quelles sont les potentielles transcriptions non bruitées Y^* qui peuvent nous conduire à la transcription bruitée $\tilde{Y}$ présente dans le jeu de données. Nous faisons cela avec une recherche en faisceau différentiable. La méthode ne nécessite pas d'avoir un alignement explicite entre Y^*, $\tilde{Y}$ et le signal acoustique X.

La partie 3.1 présente tout d'abord les types de modèles de bruit de transcription que nous proposons. Cela a une influence directe sur la conception de l'algorithme Lead2Gold. La partie 3.2 présente l'algorithme. Nous justifierons mathématiquement les choix effectués, qui sont inspirés d'une formulation des algorithmes dynamiques de calcul de CTC et ASG. Cette formulation justifiera l'utilisation d'une recherche en faisceau. Nous expliquerons également comment rendre différentiable cette dernière. Les preuves mathématiques présentées dans cette partie sont de nouvelles contributions pour ce manuscrit par rapport à l'article original [Dufraux et al., 2019]. La partie 3.3 présente les expériences faites sur des jeux de données synthétiques.

3.1. Modèle de bruit proposé

Considérons $\tilde{Y} = [\tilde{y}_1, ..., \tilde{y}_{L_1}]$ une transcription bruitée de longueur L_1 présente dans le jeu de données. Nous notons également $Y^* = [y^*_1, ..., y^*_{L_2}]$ la transcription correcte inconnue de longueur L_2. Nous présentons ici le modèle de bruit $p(\tilde{Y}|Y^*)$ qui fournit la vraisemblance d'une transcription bruitée sachant la transcription correcte. Remarquons que le modèle de bruit ne dépend pas du signal acoustique X, il dépend uniquement de la transcription bruitée et de la transcription correcte. C'est un choix que nous faisons pour pouvoir exprimer ce modèle de bruit de manière simple.

Un modèle de bruit au niveau du symbole Dans ce chapitre nous considérons uniquement des modèles de bruit de transcription au niveau du symbole. Une transcription correcte Y^* peut être transformée en une transcription bruitée $\tilde{Y}$ par des opérations de substitution, de délétion ou d'insertion de symboles. Les symboles utilisés sont des lettres (26 symboles) auxquels nous ajoutons l'apostrophe et le symbole d'espacement entre les mots. Le dictionnaire de symboles $\mathcal{D}$ est donc de taille $M = 28$. Pour plus de simplicité nous nous interdisons la délétion ou l'insertion de deux symboles consécutifs. De plus, une

insertion et une délétion ne peuvent se suivre : il faut utiliser une substitution à la place. Enfin les substitutions, délétions et insertions effectuées sont indépendantes entre elles. Pour modéliser ces contraintes, nous utilisons un symbole « vide » $\emptyset$. Nous introduisons ce symbole entre chaque symbole de $\tilde{Y}$ et Y^* de sorte que

$$\tilde{Y} = [\emptyset, \tilde{y}_1, \emptyset, ..., \emptyset, \tilde{y}_{L_1}, \emptyset] \text{ et } Y^* = [\emptyset, y_1^*, \emptyset, ..., \emptyset, y_{L_2}^*, \emptyset]. \tag{3.1}$$

Les symboles $\emptyset$ de Y^* servent à modéliser les insertions de symboles. Ils doivent tous être utilisés : soit il y a une insertion, soit il n'y en a pas. Les symboles $\emptyset$ de $\tilde{Y}$ servent à modéliser les délétions. Ils ne sont utilisés que lorsqu'il y a une délétion. Tous les symboles y_i^* et $\tilde{y}_i$ doivent être utilisés. Le modèle de bruit de transcription est composé des termes :

- de substitution $p(\tilde{y}_i|y_j^*)$. Il y en a $M^2 = 784$, un pour chaque paire de symboles possible. Si $\tilde{y}_i = y_j^*$, alors c'est un terme de non-substitution.
- de délétion $p(\emptyset|y_j^*)$. Il y en a 28, un pour chaque symbole.
- d'insertion $p(\tilde{y}_i|\emptyset)$. Il y en a 28, un pour chaque symbole. Nous ajoutons aussi la probabilité de ne pas faire d'insertion $p(\emptyset|\emptyset)$.

Ces termes forment une distribution de probabilité telle que

$$p(\emptyset|y_j^*) + \sum_{\tilde{y}_i \in \mathcal{D}} p(\tilde{y}_i|y_j^*) = 1 \quad \forall y_j^* \in \mathcal{D} \tag{3.2}$$

$$p(\emptyset|\emptyset) + \sum_{\tilde{y}_i \in \mathcal{D}} p(\tilde{y}_i|\emptyset) = 1. \tag{3.3}$$

Plusieurs alignements possibles Comme $\tilde{Y}$ et Y^* ne sont pas forcément de la même longueur et que nous autorisons les insertions et les délétions, il y a plusieurs façons de transformer Y^* en $\tilde{Y}$. Une façon de transformer Y^* en $\tilde{Y}$ correspond en fait à un alignement entre ces deux transcriptions, c'est-à-dire une suite d'opérations de substitution, de délétion ou d'insertion. Un tel alignement est noté

$$a = [(\tilde{a}_1, a_1^*), \ldots, (\tilde{a}_{L_a}, a_{L_a}^*)], \tag{3.4}$$

avec chaque $(\tilde{a}_i, a_i^*)$ une paire représentant l'opération de substitution, de délétion ou d'insertion. Les a_i^* et $\tilde{a}_i$ sont des symboles de $\mathcal{D}$ ou alors le symbole $\emptyset$. L_a est le nombre d'opérations nécessaires. La Fig. 3.1 donne deux exemples d'alignements. Nous notons $A_{Y^*}^{\tilde{Y}}$ l'ensemble de tous les alignements possibles entre $\tilde{Y}$ et Y^*.

Utilisation du modèle de bruit L'utilisation du modèle de bruit $p(\tilde{Y}|Y^*)$ dépend de l'alignement choisi entre $\tilde{Y}$ et Y^*. Pour un alignement a fixé, nous calculons :

$$p(\tilde{Y}|Y^*, a) = \prod_{i=1}^{L_a} p(\tilde{a}_i|a_i^*). \tag{3.5}$$

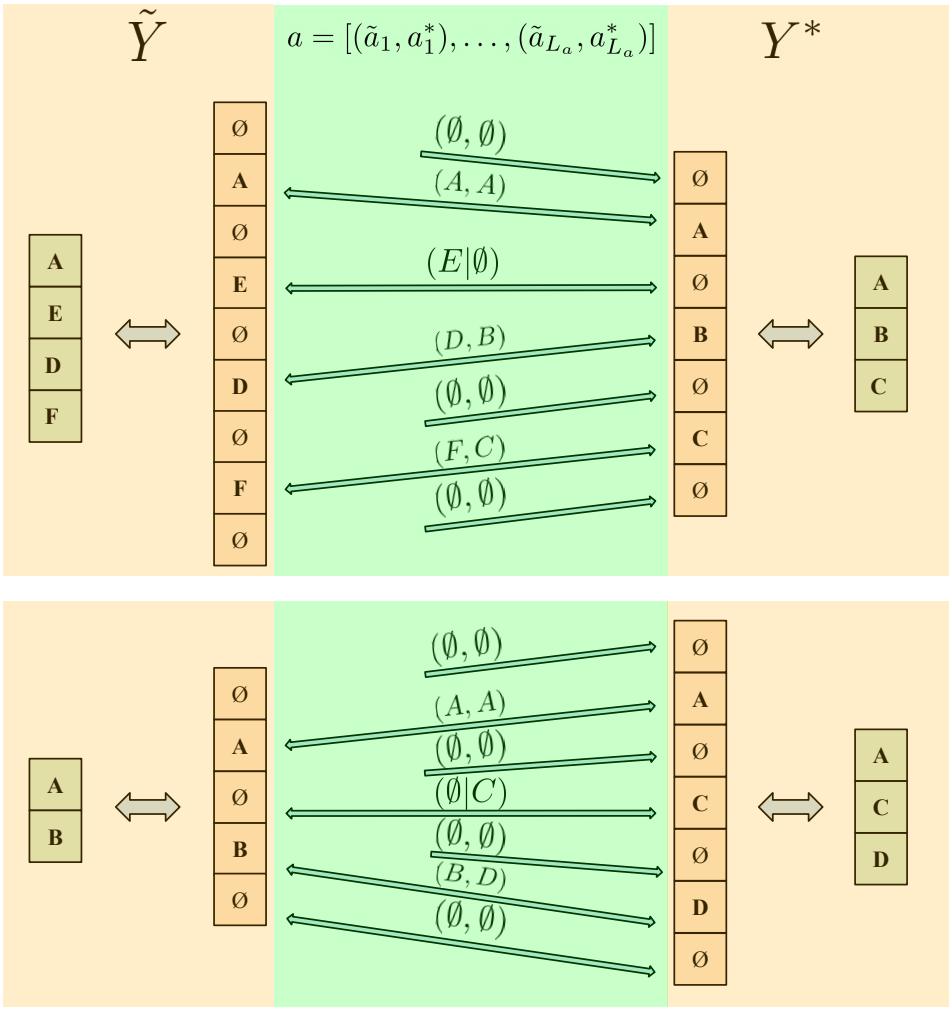


FIGURE 3.1. – Exemples d'alignements entre transcriptions. Sur la première figure (haut), il faut utiliser au moins une opération d'insertion d'un symbole bruité. Sur la seconde figure (bas), il faut utiliser au moins une opération de délétion d'un symbole correct. Pour les deux exemples, il existe beaucoup d'autres façons de passer de Y^* à $\tilde{Y}$.

Ensuite pour obtenir $p(\tilde{Y}|Y^*)$, nous prenons en compte tous les alignements possibles entre $\tilde{Y}$ et Y^* dans $A^{\tilde{Y}}_{Y^*}$. Nous calculons ici directement la log-probabilité :

$$\log p(\tilde{Y}|Y^*) = \log \sum_{a \in A^{\tilde{Y}}_{Y^*}} p(\tilde{Y}|Y^*, a) \tag{3.6}$$

$$= \log \sum_{a \in A^{\tilde{Y}}_{Y^*}} \prod_{i=1}^{L_a} p(\tilde{a}_i|a^*_i) \tag{3.7}$$

$$= \log \sum_{a \in A^{\tilde{Y}}_{Y^*}} \exp\left(\sum_{i=1}^{L_a} \log p(\tilde{a}_i|a^*_i)\right) \tag{3.8}$$

$$= \underset{a \in A^{\tilde{Y}}_{Y^*}}{\text{logadd}} \sum_{i=1}^{L_a} \log p(\tilde{a}_i|a^*_i). \tag{3.9}$$

Nous notons $\mathcal{Y}_{\tilde{Y}}$ l'ensemble des transcriptions qui peuvent se transformer en $\tilde{Y}$ selon un tel modèle de bruit :

$$\mathcal{Y}_{\tilde{Y}} = \{Y^* \mid p(\tilde{Y}|Y^*) > 0\}. \tag{3.10}$$

Cas simplifié Nous considérons également le cas où nous autorisons seulement les opérations de substitution. Cela force Y^* et $\tilde{Y}$ à être de la même longueur. Il n'y a plus qu'un seul alignement possible entre les deux transcriptions, donc $L_a = L_1 = L_2 = L$. L'expression du modèle de bruit se réduit alors à

$$\log p(\tilde{Y}|Y^*) = \sum_{i=1}^{L} \log p(\tilde{y}_i|y_i^*). \tag{3.11}$$

3.2. Algorithme d'apprentissage proposé

3.2.1. Fonction de coût prenant en compte le bruit

Nous prenons l'exemple de Sukhbaatar et al. [2014] pour intégrer le modèle de bruit de transcription dans une fonction de coût. Pour cela, toutes les transcriptions Y^* pouvant mener à la transcription bruitée $\tilde{Y}$ selon le modèle de bruit $p(\tilde{Y}|Y^*)$ sont prises en compte :

$$\begin{aligned}
\mathrm{L2G}(\tilde{Y}) &= -\log p(\tilde{Y}|X) && (3.12)\\
&= -\log \sum_{Y^* \in \mathcal{Y}_{\tilde{Y}}} p(\tilde{Y}|Y^*)p(Y^*|X) && (3.13)\\
&= -\log \sum_{Y^* \in \mathcal{Y}_{\tilde{Y}}} \exp\Big(\log p(\tilde{Y}|Y^*) + \log p(Y^*|X)\Big) && (3.14)\\
&= -\underset{Y^* \in \mathcal{Y}_{\tilde{Y}}}{\mathrm{logadd}} \Big[\log p(\tilde{Y}|Y^*) + \log p(Y^*|X)\Big]. && (3.15)
\end{aligned}$$

Le travail de Sukhbaatar et al. [2014] porte sur la reconnaissance d'images. Il y a un nombre de catégories restreint à prendre en compte. Mais, dans notre cas, l'ensemble $\mathcal{Y}_{\tilde{Y}}$ est très grand. Le problème est ici plus difficile car la fonction de coût ne peut pas être calculée de façon exacte. Pour l'instant, partons du fait que nous connaissons le modèle de bruit de transcription $p(\tilde{Y}|Y^*)$. Nous verrons dans la partie expérimentale 3.3 comment nous l'obtenons. Dans l'équation (3.15), il nous reste à expliciter $p(Y^*|X)$, la vraisemblance d'une transcription Y^* sachant le signal acoustique X. Nous choisissons d'utiliser la fonction de coût ASG pour modéliser le terme $p(Y^*|X)$. Tout d'abord nous définissons :

$$\begin{aligned}
Z &= \underset{\pi \in G_{\mathrm{full}}(T)}{\mathrm{logadd}} z_\pi(X) && (3.16)\\
S_{\mathrm{ASG}}(Y) &= \underset{\pi \in G_{\mathrm{ASG}}(Y,T)}{\mathrm{logadd}} z_\pi(X), && (3.17)
\end{aligned}$$

où $S_{\text{ASG}}(Y)$ est le score non normalisé de la fonction de coût ASG et Z est le terme de normalisation au niveau de la séquence. $G_{\text{ASG}}(Y,T)$ est un graphe dont l'ensemble des chemins acceptés représente un alignement valide entre la transcription Y et le signal acoustique comportant T trames. $z_\pi(X)$ est le score d'un chemin π. La fonction de coût ASG s'exprime alors par

$$\begin{aligned}
\text{ASG}(Y) &= -\log p(Y|X) && (3.18)\\
&= \underset{\pi\in G_{\text{full}}(T)}{\text{logadd}}\ z_\pi(X) - \underset{\pi\in G_{\text{ASG}}(Y,T)}{\text{logadd}}\ z_\pi(X) && (3.19)\\
&= Z - S_{\text{ASG}}(Y). && (3.20)
\end{aligned}$$

Donc l'équation (3.15) devient

$$\begin{aligned}
\text{L2G}(\tilde{Y}) &= -\underset{Y^*\in\mathcal{Y}_{\tilde{Y}}}{\text{logadd}} \left[\log p(\tilde{Y}|Y^*) - \text{ASG}(Y^*)\right] && (3.21)\\
&= -\underset{Y^*\in\mathcal{Y}_{\tilde{Y}}}{\text{logadd}} \left[\log p(\tilde{Y}|Y^*) + S_{\text{ASG}}(Y^*) - Z\right]. && (3.22)
\end{aligned}$$

Le terme Z étant une constante, nous pouvons le sortir de la fonction logadd (voir propriété 2 de l'annexe A). Nous obtenons

$$\begin{aligned}
\text{L2G}(\tilde{Y}) &= Z - \underset{Y^*\in\mathcal{Y}_{\tilde{Y}}}{\text{logadd}} \left[\log p(\tilde{Y}|Y^*) + S_{\text{ASG}}(Y^*)\right] && (3.23)\\
&= Z - S_{\text{L2G}}(\tilde{Y}). && (3.24)
\end{aligned}$$

La fonction de coût Lead2Gold est composée du même terme de normalisation Z que celui d'ASG et du score non-normalisé $S_{\text{L2G}}(\tilde{Y})$. Celui-ci prend en compte toutes les transcriptions Y^* pouvant produire la transcription bruitée $\tilde{Y}$. Pour chaque Y^* possible, il faut aussi prendre en compte tous les alignements entre ce Y^* et le signal acoustique X. Pour mettre en évidence cela, nous écrivons :

$$\begin{aligned}
S_{\text{L2G}}(\tilde{Y}) &= \underset{Y^*\in\mathcal{Y}_{\tilde{Y}}}{\text{logadd}} \left[\log p(\tilde{Y}|Y^*) + S_{\text{ASG}}(Y^*)\right] && (3.25)\\
&= \underset{Y^*\in\mathcal{Y}_{\tilde{Y}}}{\text{logadd}} \left[\log p(\tilde{Y}|Y^*) + \underset{\pi^*\in G_{\text{ASG}}(Y^*,T)}{\text{logadd}}\ z_{\pi^*}(X)\right] && (3.26)\\
&= \underset{Y^*\in\mathcal{Y}_{\tilde{Y}}}{\text{logadd}}\ \underset{\pi^*\in G_{\text{ASG}}(Y^*,T)}{\text{logadd}} \left[z_{\pi^*}(X) + \log p(\tilde{Y}|Y^*)\right] && (3.27)\\
&= \underset{\substack{Y^*\in\mathcal{Y}_{\tilde{Y}}\\ \pi^*\in G_{\text{ASG}}(Y^*,T)}}{\text{logadd}} \left[z_{\pi^*}(X) + \log p(\tilde{Y}|Y^*)\right]. && (3.28)
\end{aligned}$$

Nous intégrons maintenant notre modèle de bruit au niveau de la lettre. Celui-ci prend en compte tous les alignements possibles entre deux transcriptions $(\tilde{Y}, Y^*)$. Le score de

Lead2Gold devient :

$$S_{\text{L2G}}(\tilde{Y}) = \underset{\substack{Y^* \in \mathcal{Y}_{\tilde{Y}} \\ \pi^* \in G_{\text{ASG}}(Y^*,T)}}{\text{logadd}} \left[z_{\pi^*}(X) + \underset{a \in A^{\tilde{Y}}_{Y^*}}{\text{logadd}} \sum_{i=1}^{L_a} \log p(\tilde{a}_i | a_i^*) \right] \tag{3.29}$$

$$= \underset{\substack{Y^* \in \mathcal{Y}_{\tilde{Y}} \\ \pi^* \in G_{\text{ASG}}(Y^*,T)}}{\text{logadd}} \underset{a \in A^{\tilde{Y}}_{Y^*}}{\text{logadd}} \left[z_{\pi^*}(X) + \sum_{i=1}^{L_a} \log p(\tilde{a}_i | a_i^*) \right] \tag{3.30}$$

$$= \underset{\substack{Y^* \in \mathcal{Y}_{\tilde{Y}} \\ \pi^* \in G_{\text{ASG}}(Y^*,T) \\ a \in A^{\tilde{Y}}_{Y^*}}}{\text{logadd}} \left[z_{\pi^*}(X) + \sum_{i=1}^{L_a} \log p(\tilde{a}_i | a_i^*) \right] \tag{3.31}$$

$$= \underset{(Y^*,\pi^*,a) \in G_{\text{L2G}}(\tilde{Y},T)}{\text{logadd}} s(Y^*, \pi^*, a|X), \tag{3.32}$$

où $G_{\text{L2G}}(\tilde{Y}, T)$ est un graphe dont l'ensemble des chemins représente tous les alignements π^* sur T trames de toutes les transcriptions possibles Y^* alignées de toutes les manières possibles sur la transcription bruitée $\tilde{Y}$. Ce nombre de chemins est très grand. Nous ne construisons pas le graphe $G_{\text{L2G}}(\tilde{Y}, T)$ de façon exacte. Avec un algorithme de recherche en faisceau, nous prenons en compte un sous-ensemble de ces chemins et obtenons une approximation de $S_{\text{L2G}}(\tilde{Y})$. Pour cela, nous nous inspirons d'un algorithme dynamique qui permet de calculer S_{ASG} et S_{CTC}. Pour obtenir la fonction de coût finale, il faut aussi pouvoir calculer le terme de normalisation Z.

3.2.2. Calcul de Z, S_{ASG} et S_{CTC}

Nous présentons maintenant les méthodes permettant d'obtenir ces trois valeurs. Nous introduisons tout d'abord quelques notations :

- $C_{:l} = [c_1, c_2, ..., c_l]$ est un chemin composé de l états.
- $C_{:l} \in G_{:l}(Y, T)$ est un chemin partiel du graphe $G(Y, T)$. Les chemins acceptés de $G(Y, T)$ sont de longueur T. $C_{:l} \in G_{:l}(Y, T)$ est constitué des l premiers éléments d'un chemin accepté. Selon le contexte, nous utiliserons G_{full}, G_{ASG}, G_{CTC} ou G_{L2G}. Dans le cas de G_{full}, G_{ASG} et G_{CTC}, un chemin C représente un alignement π entre la transcription Y et le signal acoustique X.
- $\mathcal{C}^l_{\to[e_1,...,e_n]}$ est l'ensemble des chemins $C_{:l}$ de longueur l pouvant être suivis des états $[e_1, \ldots, e_n]$ dans le graphe $G(Y, T)$:

$$\mathcal{C}^l_{\to[e_1,...,e_n]} = \{C_{:l} | [C_{:l}, e_1, \ldots, e_n] \in G_{:l+n}(Y, T)\}. \tag{3.33}$$

- E^l est l'ensemble des états pouvant se trouver en l-ième position dans le graphe $G(Y, T)$:

$$E^l = \{e | \exists\, C_{:l-1} \in G_{:l-1}(Y, T) \text{ tel que } [C_{:l-1}, e] \in G_{:l}(Y, T)\}. \tag{3.34}$$

— $E^l_{\to s}$ est l'ensemble des états pouvant se trouver en l-ème position dans le graphe $G(Y,T)$ tel qu'il existe un chemin menant de ces états à l'état s situé en position $l+1$:

$$E^l_{\to s} = \{e|\ \exists\ C_{:l-1} \in G_{:l-1}(Y,T) \text{ tel que } [C_{:l-1}, e, s] \in G_{:l+1}(Y,T)\}\,. \tag{3.35}$$

— $s(C_{:l}|X)$ est le score cumulé du chemin partiel $C_{:l}$. Pour ASG et CTC c'est le score cumulé d'un alignement partiel $\pi_{:l}$:

$$s(C_{:l}|X) = s(\pi_{:l}|X) = \sum_{t=1}^{l} s_{\pi_t}(X) + g_{\pi_{t-1},\pi_t} \text{ pour ASG et} \tag{3.36}$$

$$s(C_{:l}|X) = s(\pi_{:l}|X) = \sum_{t=1}^{l} f_{\pi_t}(X) \text{ pour CTC.} \tag{3.37}$$

Pour chacun des calculs que nous allons conduire, le but est de séparer le problème en T étapes : une étape pour chaque trame. Pour cela nous notons

$$\alpha_{e,l} = \underset{C_{:l-1} \in \mathcal{C}^{l-1}_{\to [e]}}{\text{logadd}}\ s\left([C_{:l-1}, e]|X\right). \tag{3.38}$$

C'est le score cumulé des chemins de longueur l de $G(Y,T)$ se terminant par l'état e. Le score cumulé des chemins de longueur l se terminant par n'importe quel état s'obtient alors par

$$\alpha_l = \underset{e \in E^l}{\text{logadd}}\ \alpha_{e,l} \tag{3.39}$$

$$= \underset{C_{:l} \in G_{:l}(Y,T)}{\text{logadd}}\ s(C_{:l}|X). \tag{3.40}$$

Nous remarquons alors que selon le graphe G que nous utilisons nous avons

— $\alpha_T = S_{\text{ASG}}(Y)$ si $G = G_{\text{ASG}}$,

— $\alpha_T = S_{\text{CTC}}(Y)$ si $G = G_{\text{CTC}}$,

— $\alpha_T = S_{\text{L2G}}(Y)$ si $G = G_{\text{L2G}}$,

— $\alpha_T = Z$ si $G = G_{\text{full}}$.

Reformulation de $\alpha_{e,l}$ La formulation de l'équation (3.38) n'est pas encore idéale. Elle opère sur un ensemble de chemins difficiles à déterminer. L'idée de l'algorithme dynamique est de déterminer comment $\alpha_{e,l}$ peut être exprimé à partir de valeurs de α de

trames précédentes. Pour la démonstration ci-dessous nous calculons les scores des chemins avec la méthode ASG de l'équation (3.36) :

$$\alpha_{e,l} = \operatorname*{logadd}_{C:l-1 \in \mathcal{C}^{l-1}_{\to[e]}} s\left([C_{:l-1}, e]|X\right) \tag{3.41}$$

$$= \operatorname*{logadd}_{\substack{m \in E^{l-1}_{\to e} \\ C_{:l-2} \in \mathcal{C}^{l-2}_{\to[m]}}} s([C_{:l-2}, m, e]|X) \tag{3.42}$$

$$= \operatorname*{logadd}_{m \in E^{l-1}_{\to e}} \operatorname*{logadd}_{C_{:l-2} \in \mathcal{C}^{l-2}_{\to[m]}} s([C_{:l-2}, m, e]|X) \tag{3.43}$$

$$= \operatorname*{logadd}_{m \in E^{l-1}_{\to e}} \operatorname*{logadd}_{C_{:l-2} \in \mathcal{C}^{l-2}_{\to[m]}} \left[s_{c_l=e}(X) + g_{m,e} + \sum_{t=1}^{l-1} \left(s_{c_t}(X) + g_{c_{t-1},c_t} \right) \right] \tag{3.44}$$

$$= s_{c_l=e}(X) + \operatorname*{logadd}_{m \in E^{l-1}_{\to e}} \left[g_{m,e} + \operatorname*{logadd}_{C_{:l-2} \in \mathcal{C}^{l-2}_{\to[m]}} \sum_{t=1}^{l-1} \left(s_{c_t}(X) + g_{c_{t-1},c_t} \right) \right] \tag{3.45}$$

$$= s_{c_l=e}(X) + \operatorname*{logadd}_{m \in E^{l-1}_{\to e}} \left[g_{m,e} + \operatorname*{logadd}_{C_{:l-2} \in \mathcal{C}^{l-2}_{\to[m]}} s([C_{:l-2}, m]|X) \right] \tag{3.46}$$

$$= s_{c_l=e}(X) + \operatorname*{logadd}_{m \in E^{l-1}_{\to e}} \left[g_{m,e} + \alpha_{m,l-1} \right]. \tag{3.47}$$

Dans le cas où nous calculons les scores avec la méthode CTC, nous obtenons alors une formulation plus simple :

$$\alpha_{e,l} = f_{c_l=e}(X) + \operatorname*{logadd}_{m \in E^{l-1}_{\to e}} \alpha_{m,l-1}. \tag{3.48}$$

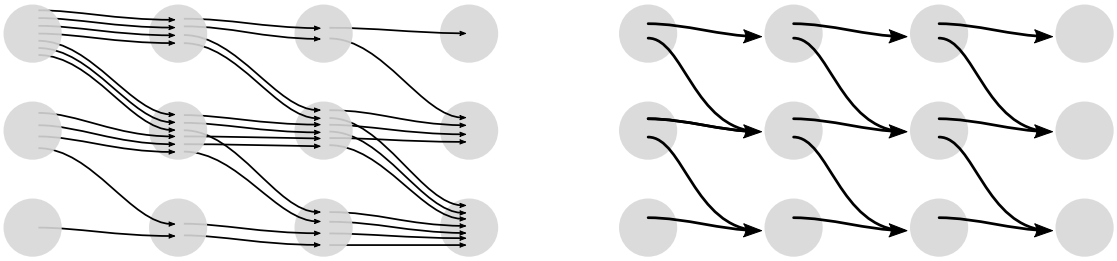
La Fig. 3.2 présente visuellement pourquoi la formulation de l'équation (3.47) est efficace.

3.2.2.1. Formulation générale de l'algorithme dynamique

L'Algorithme 1 décrit comment calculer les $\alpha_{e,l}$ pour les T trames. Pour la première trame, les $\alpha_{e,1}$ ne viennent d'aucun état puisque ce sont les premiers. Nous initialisons l'algorithme en associant à chaque $\alpha_{e,1}$ le score acoustique de la première trame pour le symbole de l'état e. Pour les graphes G_{full}, G_{ASG} et G_{CTC}, l'ensemble E^l, qui est l'ensemble des états autorisés à la trame T, est en fait composé de tous les symboles du dictionnaire $\mathcal{D}$. Il y a donc au maximum M états par trame.

3.2.2.2. Cas $G = G_{\text{full}}$

Reprenons le calcul pour le cas $G = G_{\text{full}}$, afin d'obtenir le terme de normalisation Z. Celui-ci est utilisé pour ASG mais aussi pour notre algorithme Lead2Gold. $G_{\text{full}}(T)$ est



(a) Pour chaque état, tous les alignements sont pris en compte. C'est la formulation de l'équation (3.41).

(b) Les alignements passant par des états en commun sont regroupés. C'est la formulation de l'équation (3.47) et cela est bien plus efficace.

FIGURE 3.2. – Comparaison visuelle de l'efficacité de l'algorithme dynamique. Les figures proviennent de Hannun [2017].

Algorithme 1 : Algorithme dynamique pour le calcul de Z, S_{ASG} ou S_{CTC}

`/* Initialisation */`
pour chaque état $e \in E^1$ **faire**
  $\alpha_{e,1} = s_{c_1=e}(X)$ si ASG
  $\alpha_{e,1} = f_{c_1=e}(X)$ si CTC
fin

`/* Algorithme */`
pour chaque trame $l \in [2 \dots T]$ **faire**
  pour chaque état $e \in E^l$ **faire**
    $\alpha_{e,l} = -\infty$
    pour chaque état $m \in E^{l-1}_{\to e}$ **faire**
      $\alpha_{e,l} = \text{logadd}(\alpha_{e,l}, g_{m,e} + \alpha_{m,l-1})$ si ASG
      $\alpha_{e,l} = \text{logadd}(\alpha_{e,l}, \alpha_{m,l-1})$ si CTC
    fin
    $\alpha_{e,l} = \alpha_{e,l} + s_{c_l=e}(X)$ si ASG
    $\alpha_{e,l} = \alpha_{e,l} + f_{c_l=e}(X)$ si CTC
  fin
fin

`/* Fin */`
$\alpha_T = -\infty$
pour chaque état $e \in E^T$ **faire**
  $\alpha_T = \text{logadd}(\alpha_T, \alpha_{e,T})$
fin
retourner α_T

un graphe qui comprend toutes les combinaisons possibles de chemins de longueur T. Donc $E_{\to e}^{l-1}$ est composé de tous les symboles du dictionnaire puisque l'état e peut être précédé de n'importe quel symbole. Nous obtenons

$$\alpha_{e,l} = s_e(X) + \underset{m \in \mathcal{D}}{\text{logadd}} \left(g_{m,e} + \alpha_{m,l-1} \right). \tag{3.49}$$

Le calcul d'un $\alpha_{e,l}$ nécessite d'utiliser l'opération logadd sur tous les symboles du dictionnaire de taille M. Pour chaque trame, il y en a M à calculer. De plus, il y a T trames à traiter. La complexité algorithmique du calcul de Z est donc $\mathcal{O}(T \times M^2)$.

3.2.2.3. Cas $G = G_{\text{ASG}}$

Dans le cas de G_{ASG}, il n'y a que deux possibilités pour $E_{\to e}^{l-1}$:

- L'état m précédent modélise le même symbole que e.
- L'état m précédent modélise un autre symbole. Il y a eu une transition de symbole entre les états m et e.

Soit $Y = [y_1, \ldots, y_i, \ldots, y_L]$ la transcription sur laquelle nous voulons calculer la fonction de coût ASG. Un alignement acceptable de Y sur les T trames de X s'écrit $\pi = [\pi_1, ...\pi_T]$, avec chaque état π_i égal à un symbole de Y. Pour ASG, l'équation (3.47) donne alors

$$\alpha_{y_i,l} = s_{y_i}(X) + \text{logadd } (g_{y_i,y_i} + \alpha_{y_i,l-1}, g_{y_{i-1},y_i} + \alpha_{y_{i-1},l-1}) \tag{3.50}$$

$$= s_{y_i}(X) + \log \left(e^{g_{y_i,y_i} + \alpha_{y_i,l-1}} + e^{g_{y_{i-1},y_i} + \alpha_{y_{i-1},l-1}} \right). \tag{3.51}$$

3.2.2.4. Cas $G = G_{\text{CTC}}$

Dans le cas de CTC, le symbole ϵ peut s'intercaler entre deux symboles de Y pour modéliser des moments de silence. Il faut aussi obligatoirement en ajouter un entre deux symboles identiques consécutifs de Y. Nous pouvons alors distinguer les cas suivants :

- Si l'état e courant modélise ϵ :
  - L'état m précédent modélise aussi ϵ.
  - L'état m précédent modélise un symbole. Il y a eu une transition d'un symbole vers ϵ.
- Si l'état e courant est un symbole :
  - L'état m précédent modélise aussi ce symbole.
  - L'état m précédent modélise ϵ.
  - L'état m précédent modélise un autre symbole. Nous ne pouvons activer ce cas que si ces deux symboles sont différents.

Pour faciliter la description du calcul de $\alpha_{s,l}$, définissons la transcription Y^ϵ que nous obtenons en introduisant ϵ entre chaque symbole de Y :

$$Y^\epsilon = [\epsilon, y_1, \epsilon, \ldots, \epsilon, y_i, \epsilon, y_L, \epsilon] = [y_1^\epsilon, \ldots, y_{2\times L+1}^\epsilon]. \tag{3.52}$$

En reprenant l'équation (3.48), nous retrouvons une formulation similaire à celle présentée par Graves et al. [2006] :

$$\alpha_{y_i^\epsilon,l} = \begin{cases} f_s(X) + \log\left(e^{\alpha_{\epsilon,l-1}} + e^{\alpha_{y_{i-1}^\epsilon,l-1}}\right) & \text{si } y_i^\epsilon = \epsilon \\ f_s(X) + \log\left(e^{\alpha_{\epsilon,l-1}} + e^{\alpha_{y_i^\epsilon,l-1}}\right) & \text{si } y_i^\epsilon \neq \epsilon \text{ et } y_{i-2}^\epsilon = y_i^\epsilon \\ f_s(X) + \log\left(e^{\alpha_{\epsilon,l-1}} + e^{\alpha_{y_i^\epsilon,l-1}} + e^{\alpha_{y_{i-1}^\epsilon,l-1}}\right) & \text{si } y_i^\epsilon \neq \epsilon \text{ et } y_{i-2}^\epsilon \neq y_i^\epsilon. \end{cases} \tag{3.53}$$

Pour les cas G_{ASG} et G_{CTC}, l'ensemble $E_{\to e}^{l-1}$ ne comporte que deux ou trois éléments. Le calcul d'un $\alpha_{e,l}$ s'effectue en temps constant. Dans le pire des cas, nous avons tout de même besoin de calculer M $\alpha_{e,l}$ par trame. La complexité algorithmique totale est donc de $\mathcal{O}(T \times M)$.

3.2.3. Calcul exact de S_{L2G}

Dans cette partie nous nous inspirons des formulations que nous avons proposées pour Z, S_{ASG} et S_{CTC} pour voir si un algorithme dynamique similaire pourrait être utilisé pour le calcul de S_{L2G}. Rappelons tout d'abord sa formulation :

$$S_{\text{L2G}}(\tilde{Y}) = \underset{C=(Y^*,\pi^*,a)\in G_{\text{L2G}}(\tilde{Y},T)}{\text{logadd}} s(C|X) \tag{3.54}$$

avec

$$s(C=(Y^*,\pi^*,a)|X) = z_C(X) + \sum_{i=1}^{L_a} \log p(\tilde{a}_i|a_i^*) \tag{3.55}$$

$$= \sum_{t=1}^{T} \left(s_{c_t}(X) + g_{c_{t-1},c_t}\right) + \sum_{i=1}^{L_a} \log p(\tilde{a}_i|a_i^*). \tag{3.56}$$

Alignement du modèle de bruit sur les trames L'algorithme dynamique repose sur le fait que le calcul est séparé pour chaque trame. Pour cela, il nous faut une formulation de $s(C_{:l} = (Y^*,\pi^*,a)_{:l}|X)$, le score cumulé d'un chemin $C = (Y^*,\pi^*,a)$ sur les l premières trames. C'est pourquoi nous voulons reformuler la contribution du modèle de bruit de transcription pour répartir les L_a termes sur les T trames. Nous posons

$$\sum_{i=1}^{L_a} \log p(\tilde{a}_i|a_i^*) = \sum_{t=1}^{T} b^t(C). \tag{3.57}$$

Les valeurs de $b^t(C)$ dépendent du chemin C qui est construit. Nous choisissons la stratégie d'ajouter les contributions du modèle de bruit lorsqu'un nouveau symbole est produit dans ce chemin. La Fig. 3.3 reprend l'exemple de la Fig. 3.1 en déterminant les valeurs des $b^t(C)$. Avec cette stratégie, $b^t(C)$ ne dépend en fait que des deux états c_{t-1} et c_t du chemin $C = [c_1,\ldots,c_t,\ldots,c_T]$. Nous pouvons écrire

$$b^t(C) = b^t(c_{t-1},c_t). \tag{3.58}$$

L'équation (3.56) s'écrit alors

$$s(C = (Y^*, \pi^*, a)|X) = \sum_{t=1}^{T} \left(s_{c_t}(X) + g_{c_{t-1},c_t} + b^t(c_{t-1}, c_t) \right). \tag{3.59}$$

Le score cumulé jusqu'à la l-ème trame est alors

$$s(C_{:l} = (Y^*, \pi^*, a)_{:l}|X) = \sum_{t=1}^{l} \left(s_{c_t}(X) + g_{c_{t-1},c_t} + b^t(c_{t-1}, c_t) \right). \tag{3.60}$$

Formulation exacte pour G_{L2G} Rappelons que $G_{\text{L2G}}(\tilde{Y}, T)$ est un graphe dont l'ensemble des chemins représente tous les alignements π^* sur T trames de toutes les transcriptions possibles Y^* alignées de toutes les manières possibles sur la transcription bruitée $\tilde{Y}$. Reprenons la formulation à partir de l'équation (3.43) :

$$\alpha_{e,l} = \underset{m \in E^{l-1}_{\to e}}{\text{logadd}} \quad \underset{C_{:l-2} \in \mathcal{C}^{l-2}_{\to [m]}}{\text{logadd}} \; s([C_{:l-2}, m, e]|X) \tag{3.61}$$

$$= \underset{m \in E^{l-1}_{\to e}}{\text{logadd}} \quad \underset{C_{:l-2} \in \mathcal{C}^{l-2}_{\to [m]}}{\text{logadd}} \left[s_{c_l=e}(X) + g_{m,e} + b^l(m, e) + \sum_{t=1}^{l-1} \left(s_{c_t}(X) + g_{c_{t-1},c_t} + b^t(c_{t-1}, c_t) \right) \right] \tag{3.62}$$

$$= s_{c_l=e}(X) + \underset{m \in E^{l-1}_{\to e}}{\text{logadd}} \left[g_{m,e} + b^l(m, e) + \underset{C_{:l-2} \in \mathcal{C}^{l-2}_{\to [m]}}{\text{logadd}} \sum_{t=1}^{l-1} \left(s_{c_t}(X) + g_{c_{t-1},c_t} + b^t(c_{t-1}, c_t) \right) \right] \tag{3.63}$$

$$= s_{c_l=e}(X) + \underset{m \in E^{l-1}_{\to e}}{\text{logadd}} \left[g_{m,e} + b^l(m, e) + \underset{C_{:l-2} \in \mathcal{C}^{l-2}_{\to [m]}}{\text{logadd}} \; s([C_{:l-2}, m]|X) \right] \tag{3.64}$$

$$= s_{c_l=e}(X) + \underset{m \in E^{l-1}_{\to e}}{\text{logadd}} \left[g_{m,e} + b^l(m, e) + \alpha_{m,l-1} \right]. \tag{3.65}$$

Précisions sur les états de G_{L2G} Précisons ce que représente un état e à la trame l du graphe $G_{\text{L2G}}(\tilde{Y}, T)$. En particulier, cet état fait partie d'un chemin partiel $C^1_{:l} = [C^1_{:l-1}, e]$. L'état e encode un symbole y^*_j d'une transcription Y^*. De plus $C^1_{:l}$ encode $Y^{*1}_{:j}$, les j premiers symboles d'une transcription Y^{*1}. Avec $Y^{*1}_{:j}$ qui peut être transformé en $\tilde{Y}_{:i}$, les i premiers symboles de $\tilde{Y}$, par des opérations de substitution, de délétion et d'insertion. Mais l'état e fait aussi partie d'un autre chemin $C^2_{:l} = [C^2_{:l-2}, e]$ qui encode la transformation de $Y^{*2}_{:j}$ en $\tilde{Y}^*_{:i}$ par d'autres opérations de substitution, de délétion et d'insertion.

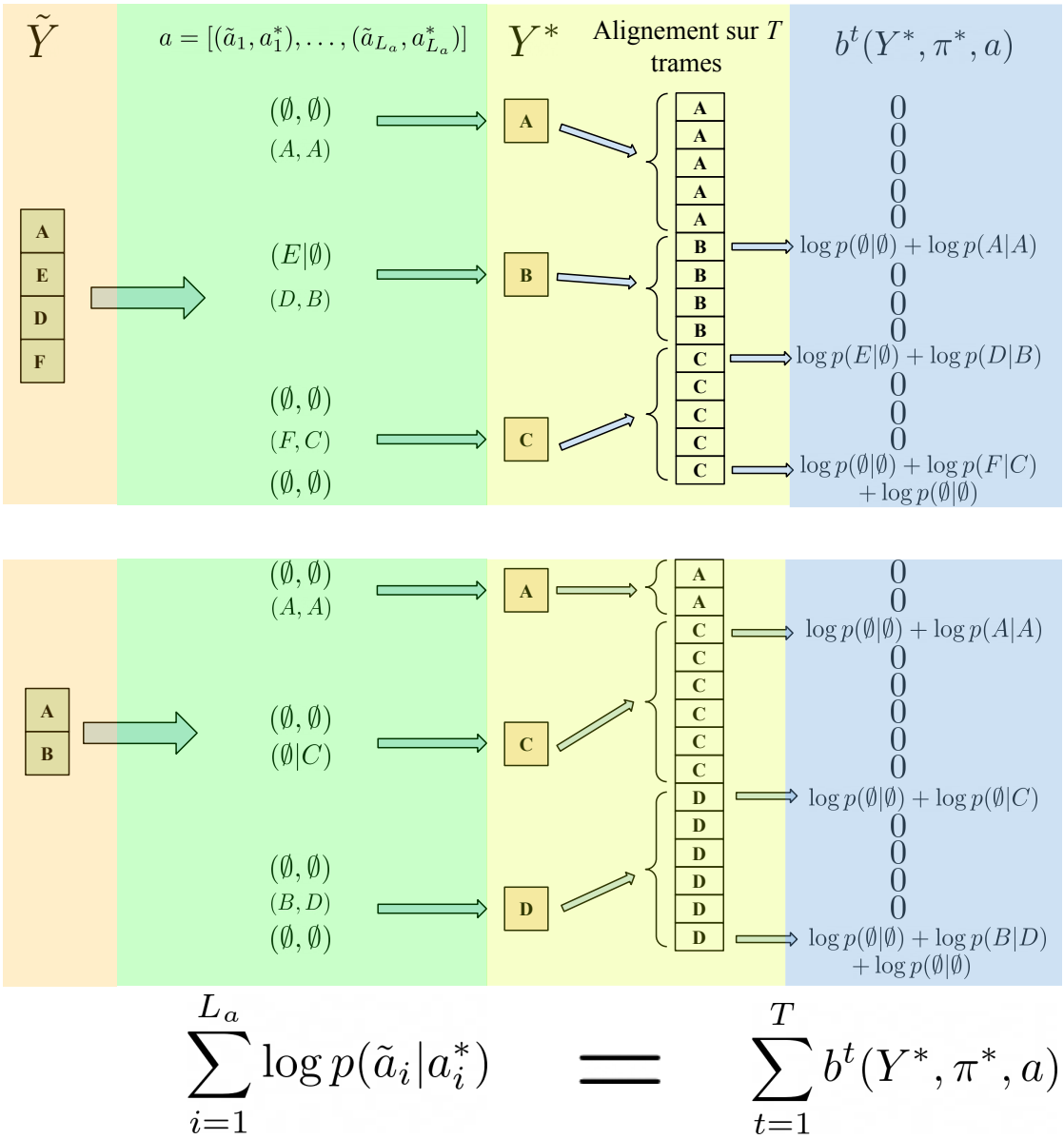


FIGURE 3.3. – Alignement des termes du modèle de bruit sur les trames. Quand un nouveau symbole de Y^* est produit, nous ajoutons les termes du modèle de bruit de transcription du symbole précédent. La contribution du dernier symbole est ajoutée à la dernière trame avec en plus une éventuelle insertion après ce dernier symbole.

En d'autres termes, si plusieurs chemins encodent plusieurs transcriptions qui se terminent par le même symbole y_j^* et qui peuvent se transformer en $\tilde{Y}_{:i}$ alors ces chemins se terminent par le même état e. Pour représenter cela, nous stockons un **curseur** dans l'état e qui indique le symbole de $\tilde{Y}$ qui est en train d'être utilisé pour produire y_j^*. Pour résumer, un état e est composé :

— d'un symbole y_j^*,

— d'un curseur sur $\tilde{Y}$.

Construction des états de G_{L2G} Dans le cas de G_{L2G} nous pouvons aussi énumérer les possibilités pour $m \in E^l_{\to e}$, c'est-à-dire les états m de la trame l qui peuvent mener à l'état e de la trame $l+1$. Il est cependant plus intuitif de procéder dans l'autre sens. Soit $m \in E^l$. À partir de cet état, énumérons les états e qui peuvent suivre. De cette manière, nous allons construire les états de l'ensemble E^{l+1} à partir des états de l'ensemble E^l.

— L'état e suivant représente le même symbole que celui de m. Le terme $b^{l+1}(m, e)$ est nul, il n'y a pas d'opération de substitution, de délétion ou d'insertion. Le curseur de e est le même que celui de m.

— L'état e suivant représente un autre symbole que celui de m. La modélisation du symbole y_j^* de m est donc finie. Le curseur de m est sur le symbole $\tilde{y}_i$. Le terme $b^{l+1}(m, e)$ n'est pas nul (voir Fig. 3.3) et nous avons alors trois cas :

  — Il n'y a pas d'insertion. Le symbole y_j^* est transformé en $\tilde{y}_i$ par une opération de substitution et $b^{l+1}(m, e) = \log p(\emptyset|\emptyset) + \log p(\tilde{y}_i|y_j^*)$. Le curseur de e est sur $\tilde{y}_{i+1}$.

  — Il n'y a pas d'insertion. Le symbole y_j^* est supprimé par une opération de délétion et $b^{l+1}(m, e) = \log p(\emptyset|\emptyset) + \log p(\emptyset|y_j^*)$. Le curseur de e reste sur $\tilde{y}_i$.

  — Il y a une opération d'insertion du symbole $\tilde{y}_i$. Le symbole y_j^* est transformé en $\tilde{y}_{i+1}$ par une opération de substitution et $b^{l+1}(m, e) = \log p(\tilde{y}_i|\emptyset) + \log p(\tilde{y}_{i+1}|y_j^*)$. Le curseur de e est sur $\tilde{y}_{i+2}$.

Contrairement à G_{ASG} ou G_{CTC}, il y a beaucoup plus que M états possibles par trame. En effet, un état $m \in E^l$ peut être suivi d'un état $e \in E^{l+1}$ qui peut représenter n'importe quel symbole de $\mathcal{D}$ et dont le curseur sur $\tilde{Y}$ peut prendre trois positions différentes. Avec cette configuration le nombre d'états explose rapidement au fur et à mesure des trames.

3.2.4. Approximation de S_{L2G} par une recherche en faisceau

Construction d'un graphe $\hat{G}_{\text{L2G}}$ approché La recherche en faisceau que nous proposons permet de fournir une approximation $\hat{S}_{\text{L2G}}$ de S_{L2G}. Pour cela nous allons construire itérativement le graphe $\hat{G}_{\text{L2G}}$ qui est composé d'un sous-ensemble d'états et d'arcs de G_{L2G}. Nous partons d'un ensemble d'états $\hat{E}^l$ connu à la trame l pour construire une liste d'états candidats $\hat{E}^{l+1}_{\text{candidat}}$ pour la trame $l+1$. Pour réduire la taille de $\hat{E}^{l+1}_{\text{candidat}}$, nous élaguons (*pruning*) de cette liste les candidats dont le score ne dépasse pas une valeur s_{seuil} définie à l'avance. Ensuite, nous fusionnons les candidats qui portent le même symbole y_j^* et le même curseur sur $\tilde{Y}$. Leurs scores sont agrégés par la fonction logadd. De cette manière, nous formons l'ensemble $E^l_{\to e}$ pour chaque état e fusionné. Mais même après ces étapes, nous ne contrôlons pas la taille de $\hat{E}^{l+1}_{\text{candidat}}$. Dans cette liste d'états nous ne gardons que les B meilleurs qui vont former la liste finale $\hat{E}^{l+1}$. C'est l'étape de réduction

de la largeur du faisceau (*beam*). L'Algorithme 2 présente ces étapes pour la création du graphe $\hat{G}_{\text{L2G}}$.

Algorithme 2 : Construction du graphe $\hat{G}_{\text{L2G}}$.

`/* Initialisation */`

Pour construire $\hat{E}^1$ nous créons un état m pour chaque symbole y_1^* autorisé selon le modèle de bruit. Le curseur de ces états est sur la position $\tilde{y}_1$ et le score est $\alpha_{m,1} = s_{c_1=y_1^*}(X)$.

pour chaque trame $l \in [1, \ldots, T-1]$ **faire**

`/* Construction de` $\hat{E}^{l+1}$ `à partir de` $\hat{E}^l$ `*/`

$\hat{E}^{l+1}_{\text{candidat}} = \emptyset$

$\hat{E}^{l+1} = \emptyset$

pour chaque état $m \in \hat{E}^l$ **faire**

Ajouter à $\hat{E}^{l+1}_{\text{candidat}}$ les candidats e_{candidat} créés à partir de m.

$\text{Score}(e_{\text{candidat}}) = s_{c_l=e_{\text{candidat}}}(X) + g_{m,e_{\text{candidat}}} + b^l(m, e_{\text{candidat}}) + \alpha_{m,l}$

fin

`/* Étape d'élagage */`

MaxScore = le meilleurs des scores de $\hat{E}^{l+1}_{\text{candidat}}$

pour chaque $e_{\text{candidat}} \in \hat{E}^{l+1}_{\text{candidat}}$ **faire**

si $\text{MaxScore} - \text{Score}(e_{\text{candidat}}) > s_{\text{seuil}}$ **alors**

Enlever e_{candidat} de $\hat{E}^{l+1}_{\text{candidat}}$

fin

fin

`/* Étape de fusion */`

Regrouper les états de $\hat{E}^{l+1}_{\text{candidat}}$ avec le même symbole et le même curseur pour former des candidats e^i

pour chaque groupe G^i de e_{candidat} utilisé pour créer un candidat e^i **faire**

$\text{Score}(e^i) = \underset{e_{\text{candidat}} \in G^i}{\text{logadd}} \text{Score}(e_{\text{candidat}})$

Ajouter e^i à $\hat{E}^{l+1}$

fin

`/* Étape de réduction de la largeur du faisceau */`

Garder dans $\hat{E}^{l+1}$ les B meilleurs candidats selon le score de ces candidats.

fin

Influence du modèle de bruit sur la sélection de candidats Avec un algorithme de recherche en faisceau, plusieurs termes influencent la sélection des candidats. Lors d'une étape de décodage classique, nous utilisons les contributions du modèle acoustique et du modèle de langage. Mais ceux-ci n'opèrent pas à la même granularité : la trame pour le modèle acoustique et le mot pour le modèle de langage. Nous ajoutons alors un poids pour régler la contribution du modèle de langage par rapport au modèle acoustique. Nous faisons de même dans notre cas afin de régler la contribution du modèle de bruit de transcription par rapport à celle du modèle acoustique. En ajoutant le poids α_{bruit}, qui est un nouvel hyper-paramètre à choisir, nous pouvons écrire

$$\hat{S}_{\text{L2G}}(\tilde{Y}) = \underset{(Y^*,\pi^*,a)\in\hat{G}_{\text{L2G}}(\tilde{Y},T)}{\text{logadd}} \left[z_{\pi^*}(X) + \alpha_{\text{bruit}} \log p(\tilde{Y}|Y^*) \right]. \tag{3.66}$$

Algorithme final pour le calcul de $\hat{S}_{\text{L2G}}$ L'Algorithme 3 décrit précisément la méthode de calcul de $\hat{S}_{\text{L2G}}$, qui inclut la construction itérative du graphe $\hat{G}_{\text{L2G}}$ et l'ajout du terme α_{bruit}. De plus, dans la partie 3.2.3 nous avons déterminé une méthode pour énumérer les états d'une trame $l+1$ créée à partir d'un état m de la trame l. Nous incluons cette méthode dans notre algorithme. Cela correspond à la ligne 8 de l'Algorithme 2. La Fig. 3.4 propose un exemple visuel d'application de cet algorithme.

Avec cet algorithme, chaque $\hat{E}^l$ est de taille B au maximum. Pour chaque hypothèse de $\hat{E}^l$, il faut tester dans le pire des cas tous les symboles de $\mathcal{D}$ (de taille M) pour générer les candidats de la trame suivante. Nous faisons cela pour chacune des T trames : la complexité algorithmique du calcul de $\hat{S}_{\text{L2G}}$ est donc $\mathcal{O}(T \times B \times M)$.

Algorithme 3 : Algorithme Lead2Gold complet pour le calcul de $\hat{S}_{\text{L2G}}$.

`AjouteCandidat(`l, y^*, $\tilde{y}$, s`) :`
  Créer un nouveau candidat qui représente le symbole y^* à la trame l avec un score de s et un curseur sur $\tilde{y}$. Ajouter ce candidat à $\hat{E}^l_{\text{candidat}}$.

`/* Initialisation */`

Pour construire $\hat{E}^1$ nous créons un état m pour chaque symbole y_1^* autorisé selon le modèle de bruit. Le curseur de ces états est sur la position $\tilde{y}_1$ et le score est $\alpha_{m,1} = s_{c_1=y_1^*}(X)$.

pour chaque trame $l \in [1, \ldots, T-1]$ **faire**
  `/* Construction de` $\hat{E}^{l+1}$ `à partir de` $\hat{E}^l$ `*/`
  $\hat{E}^{l+1}_{\text{candidat}} = \emptyset$
  $\hat{E}^{l+1} = \emptyset$
  pour chaque état $m \in \hat{E}^l$ **faire**
    L'état m a le curseur sur $\tilde{y}_i$, représente le symbole y_l^* et a un score de $\alpha_{m,l}$.
    pour chaque symbole $y^*_{l+1} \in \mathcal{D}$ **faire**
      $s_{\text{Commun}} = s_{c_l=y^*_{l+1}} + g(y^*_{l+1}|y^*l) + \alpha_{m,l}$
      si $y_l^* = y^*_{l+1}$ **alors**
        `AjouteCandidat` $(l+1,\ y^*_{l+1},\ \tilde{y}_i,\ s_{\text{Commun}})$
      sinon
        Délétion de y_l^* :
        $s = s_{\text{Commun}} + \alpha_{\text{bruit}} \log p(\emptyset|y_l^*)$
        `AjouteCandidat` $(l+1,\ y^*_{l+1},\ \tilde{y}_i,\ s)$
        Substitution de y_l^* par $\tilde{y}_i$:
        $s = s_{\text{Commun}} + \alpha_{\text{bruit}} \log p(\tilde{y}_i|y_l^*)$
        `AjouteCandidat` $(l+1,\ y^*_{l+1},\ \tilde{y}_{i+1},\ s)$
        Insertion de $\tilde{y}_i$ et substitution de y_l^* par $\tilde{y}_{i+1}$:
        $s = s_{\text{Commun}} + \alpha_{\text{bruit}} \left(\log p(\tilde{y}_i|\emptyset) + \log p(\tilde{y}_{i+1}|y_l^*)\right)$
        `AjouteCandidat` $(l+1,\ y^*_{l+1},\ \tilde{y}_{i+2},\ s)$
      fin
    fin
  fin

  Élagage de $\hat{E}^{l+1}_{\text{candidat}}$.
  Fusion de $\hat{E}^{l+1}_{\text{candidat}}$ pour créer $\hat{E}^{l+1}$.
  Réduction de $\hat{E}^{l+1}$ aux B meilleurs états.
fin
Pour obtenir $\hat{S}_{\text{L2G}}$: ajouter la dernière contribution du modèle de bruit à chaque état de $\hat{E}^T$ et fusionner tous ces états.
retourner S_{L2G}

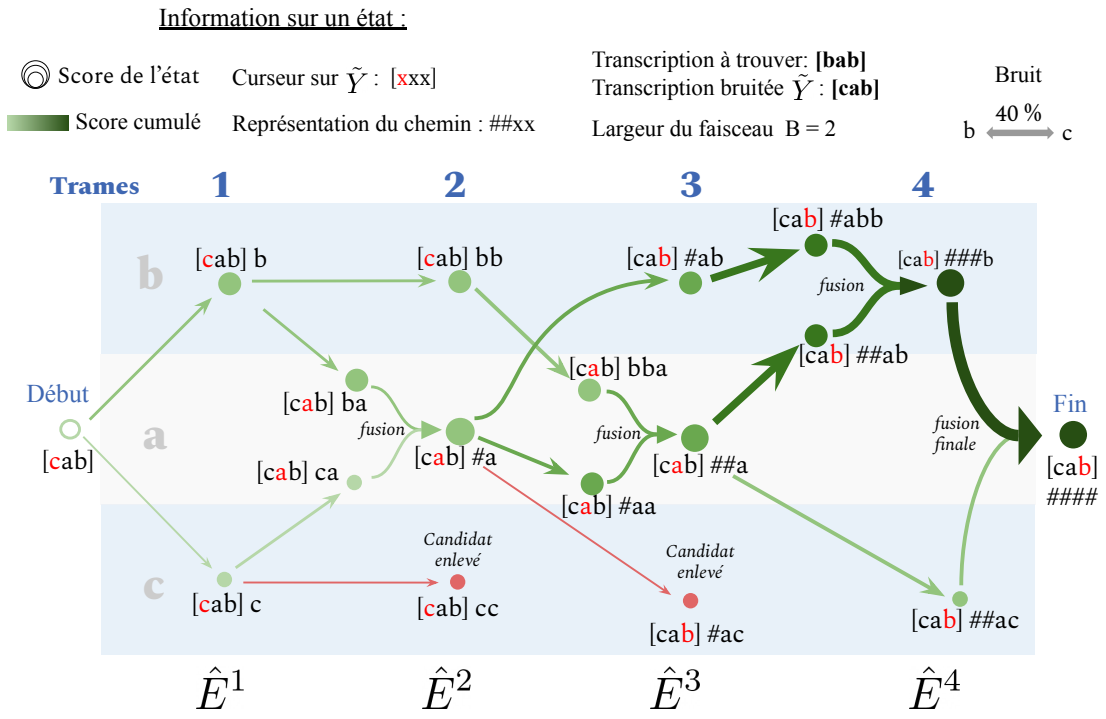


FIGURE 3.4. – Exemple d'application de Lead2Gold. Le signal acoustique est sur 4 trames. Nous avons accès à la transcription bruitée **[cab]**. Selon le modèle de bruit de transcription, les lettres **b** et **c** peuvent être confondues avec une probabilité de 40%. La véritable transcription que nous voulons trouver est **[bab]**. Nous pouvons observer que les flux de score cumulés en vert sont en effet concentrés sur des alignements de la véritable transcription.

3.2.5. Recherche en faisceau différentiable

L'algorithme Lead2Gold consiste également à pouvoir calculer les dérivées partielles de $\hat{S}_{\text{L2G}}$ par rapport aux scores fournis par le modèle acoustique. Dans cette partie, nous fournissons un algorithme permettant de calculer itérativement ces dérivées partielles. C'est la phase de rétropropagation de Lead2Gold. Nous créons cet algorithme à partir de la démonstration ci-dessous.

Nous notons :

— $\vec{\alpha_l} = (\alpha_{1,l}, \ldots, \alpha_{N_l,l})$ un vecteur composé des α de la trame l où $N_l = \text{Card}(\hat{E}^l)$. Pour rappel, ce sont les scores associés à chaque état de $\hat{E}^l$. Il y a au maximum B états par trame donc $N_l \leq B$.

— $\vec{\alpha} = (\vec{\alpha_1}, \ldots, \vec{\alpha_l}, \ldots, \vec{\alpha_T})$ l'ensemble des α associés à tous les états de toutes les trames.

— $\vec{X_l} = (s_1^l, \ldots, s_M^l)$ un vecteur composé des scores fournis par le modèle acoustique à la trame l. Il y en a M, un pour chaque symbole du dictionnaire de symboles $\mathcal{D}$.

— $X = (\vec{X_1}, \ldots, \vec{X_l}, \ldots, \vec{X_T})$ l'ensemble des scores acoustiques de toutes les trames.

Propagation par rapport aux α Le score $\hat{S}_{\text{L2G}}$ est obtenu à partir d'une suite d'opérations sur les T trames. Chaque opération f_l de la trame $l \in [1, \ldots, T-1]$ opère sur les $\vec{\alpha_l}$ et donne les $\vec{\alpha_{l+1}}$:

$f_l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$

$$\vec{\alpha_l} \mapsto \left(\begin{array}{c} f_l^1(\vec{\alpha_l}) = s_{e_1}^{l+1}(X) + \underset{m \in E^l_{\to e_1}}{\text{logadd}} \left(g_{m,e_1} + b^l(m, e_1) + \alpha_{m,l} \right) \\ \vdots \\ f_l^i(\vec{\alpha_l}) = s_{e_i}^{l+1}(X) + \underset{m \in E^l_{\to e_i}}{\text{logadd}} \left(g_{m,e_i} + b^l(m, e_i) + \alpha_{m,l} \right) \\ \vdots \\ f_l^{N_{l+1}}(\vec{\alpha_l}) = s_{e_{N_{l+1}}}^{l+1}(X) + \underset{m \in E^l_{\to e_{N_{l+1}}}}{\text{logadd}} \left(g_{m,e_{N_{l+1}}} + b^l(m, e_{N_{l+1}}) + \alpha_{m,l} \right) \end{array} \right) = \vec{\alpha_{l+1}}. \tag{3.67}$$

L'opération d'agrégation finale est notée f_T et est définie par :

$$\begin{aligned} &f_T : \mathbb{R}^{N_T} \to \mathbb{R} \\ &\vec{\alpha_T} \mapsto \underset{e \in E^T}{\text{logadd}} \left(\alpha_{e,T} + b^T(e) \right) = \hat{S}_{\text{L2G}}, \end{aligned} \tag{3.68}$$

où $b^T(e)$ est la dernière contribution du modèle de bruit à ajouter à la dernière trame. Avec ces notations, nous obtenons une formulation de $\hat{S}_{\text{L2G}}$ pour chaque trame :

$$\begin{aligned} \hat{S}_{\text{L2G}} &= f_T(\vec{\alpha_T}) && (3.69) \\ &= f_T \circ f_{T-1}(\vec{\alpha_{T-1}}) && (3.70) \\ &= f_T \circ \cdots \circ f_{l+1} \circ f_l(\vec{\alpha_l}) && (3.71) \\ &= f_T \circ \cdots \circ f_1(\vec{\alpha_1}). && (3.72) \end{aligned}$$

Chaque formulation ne dépend pas des mêmes α, nous les différencions en notant :

$$\begin{aligned} &\hat{S}^l : \mathbb{R}^{N_l} \to \mathbb{R} \\ &\vec{\alpha_l} \mapsto f_T \circ \cdots \circ f_{l+1} \circ f_l(\vec{\alpha_l}) = \hat{S}_{\text{L2G}}. \end{aligned} \tag{3.73}$$

Pour trouver les dérivées partielles de $\hat{S}_{\text{L2G}}$ par rapport à tous les scores acoustiques, nous cherchons d'abord à les calculer par rapport aux poids α en prenant les formulations

$\hat{S}^l$ pour chaque trame. En appliquant le théorème de dérivation des fonctions composées, nous obtenons :

$$\begin{aligned} J_{\hat{S}^l}(\vec{\alpha_l}) &= J_{f_T \circ \cdots \circ f_{l+1} \circ f_l}(\vec{\alpha_l}) && (3.74)\\ &= J_{f_T \circ \cdots \circ f_{l+1}}(f_l(\vec{\alpha_l}))\ J_{f_l}(\vec{\alpha_l}) && (3.75)\\ &= J_{f_T \circ \cdots \circ f_{l+1}}(\alpha_{\vec{l+1}})\ J_{f_l}(\vec{\alpha_l}) && (3.76)\\ &= J_{\hat{S}^{l+1}}(\alpha_{\vec{l+1}})\ J_{f_l}(\vec{\alpha_l}). && (3.77) \end{aligned}$$

Les applications $\hat{S}^l$ sont à valeur dans $\mathbb{R}$. La matrice jacobienne d'une de ces applications est donc un vecteur ligne (aussi égal à la transposée du gradient) et vaut :

$$J_{\hat{S}^l}(\vec{\alpha_l}) = \left[\frac{\partial \hat{S}^l}{\partial \alpha_{1,l}}(\vec{\alpha_l}), \ldots, \frac{\partial \hat{S}^l}{\partial \alpha_{i,l}}(\vec{\alpha_l}), \ldots, \frac{\partial \hat{S}^l}{\partial \alpha_{N_l,l}}(\vec{\alpha_l}) \right]. \tag{3.78}$$

Pour chaque $\alpha_{e,l}$ du graphe G_{L2G}, nous associons la valeur $\beta_{e,l}$ définie par

$$\beta_{e,l} = \frac{\partial \hat{S}^l}{\partial \alpha_{e,l}}(\vec{\alpha_l}). \tag{3.79}$$

$J_{\hat{S}^l}(\vec{\alpha_l})$ s'écrit alors :

$$J_{\hat{S}^l}(\vec{\alpha_l}) = [\beta_{1,l}, \ldots, \beta_{i,l}, \ldots, \beta_{N_l,l}]. \tag{3.80}$$

Nous pouvons calculer explicitement chaque $J_{f_l}(\vec{\alpha_l})$. Tout d'abord, en prenant $l = T$, nous obtenons pour l'opération d'agrégation finale :

$$J_{\hat{S}^T}(\vec{\alpha_T}) = J_{f_T}(\vec{\alpha_T}) = [\beta_{1,T}, \ldots, \beta_{i,T}, \ldots, \beta_{N_T,T}], \tag{3.81}$$

avec

$$\begin{aligned} \beta_{i,T} = \frac{\partial f_T}{\partial \alpha_{i,T}}(\vec{\alpha_T}) &= \frac{\partial}{\partial \alpha_{i,T}} \left(\underset{e \in E^T}{\mathrm{logadd}} \left(\alpha_{e,T} + b^T(e) \right) \right) && (3.82)\\ &= \frac{\exp\left(\alpha_{i,T} + b^T(i) \right)}{\sum_{e \in E^T} \exp\left(\alpha_{e,T} + b^T(e) \right)}. && (3.83) \end{aligned}$$

Ensuite pour $l \in [1, \ldots, T-1]$, nous obtenons

$$J_{f_l}(\vec{\alpha_l}) = \begin{bmatrix} \frac{\partial f_l^1}{\partial \alpha_{1,l}}(\vec{\alpha_l}) & \cdots & \frac{\partial f_l^1}{\partial \alpha_{N_l,l}}(\vec{\alpha_l}) \\ \vdots & \frac{\partial f_l^i}{\partial \alpha_{j,l}}(\vec{\alpha_l}) & \vdots \\ \frac{\partial f_l^{N_{l+1}}}{\partial \alpha_{1,l}}(\vec{\alpha_l}) & \cdots & \frac{\partial f_l^{N_{l+1}}}{\partial \alpha_{N_l,l}}(\vec{\alpha_l}) \end{bmatrix}, \tag{3.84}$$

avec

$$\frac{\partial f_l^i}{\partial \alpha_{j,l}}(\vec{\alpha_l}) = \frac{\partial}{\partial \alpha_{j,l}} \left(s_{e_i}^l(X) + \underset{m \in E_{\to e_i}^l}{\text{logadd}} \left(g_{m,e_i} + b^l(m,e_i) + \alpha_{m,l} \right) \right) \tag{3.85}$$

$$= \frac{\partial}{\partial \alpha_{j,l}} \left(\underset{m \in E_{\to e_i}^l}{\text{logadd}} \left(s_{e_i}^l(X) + g_{m,e_i} + b^l(m,e_i) + \alpha_{m,l} \right) \right) \tag{3.86}$$

$$= \frac{\exp\left(s_{e_i}^l(X) + g_{j,e_i} + b^l(j,e_i) + \alpha_{j,l} \right)}{\sum_{m \in E_{\to e_i}^l} \exp\left(s_{e_i}^l(X) + g_{m,e_i} + b^l(m,e_i) + \alpha_{m,l} \right)} \mathbb{1}_{[j \in E_{\to e_i}^l]} \tag{3.87}$$

$$= \frac{\exp\left(s_{e_i}^l(X) + g_{j,e_i} + b^l(j,e_i) + \alpha_{j,l} \right)}{\exp(\alpha_{i,l+1})} \mathbb{1}_{[j \in E_{\to e_i}^l]}. \tag{3.88}$$

Avec le produit matriciel de l'équation (3.77) et la formulation de $J_{\hat{S}^l}(\vec{\alpha_l})$ de l'équation (3.80), nous obtenons une relation entre les β d'une trame et ceux de la trame suivante :

$$\beta_{e,l} = \sum_{k=1}^{N_{l+1}} \beta_{k,l+1} \frac{\partial f_l^k}{\partial \alpha_{e,l}}(\vec{\alpha_l}). \tag{3.89}$$

L'équation (3.88) indique que la plupart des termes de cette somme sont nuls. Pour tous les états k de la trame $l+1$, le terme est non nul si cet état k est construit à partir de l'état e. Nous notons $E_{\leftarrow e}^{l+1}$ l'ensemble de ces états construits à partir de e :

$$E_{\leftarrow e}^{l+1} = \{s |\ \exists\ C_{:l-1} \in G_{:l-1}(Y,T) \text{ tel que } [C_{:l-1}, e, s] \in G_{:l+1}(Y,T)\}\,. \tag{3.90}$$

Il reste alors

$$\beta_{e,l} = \sum_{k \in E_{\leftarrow e}^{l+1}} \beta_{k,l+1} \frac{\partial f_l^k}{\partial \alpha_{e,l}}(\vec{\alpha_l}). \tag{3.91}$$

Propagation par rapport aux scores acoustiques En itérant sur toutes les trames, de la dernière à la première, nous avons une méthode pour obtenir tous les β, les dérivées partielles de $\hat{S}_{\text{L2G}}$ par rapport aux α. Mais ce que nous voulons, ce sont les dérivées partielles par rapport aux éléments de X, les scores acoustiques. Le résultat n'est pas immédiat car un score acoustique peut être utilisé pour plusieurs états. Nous définissons

g_l une application permettant d'obtenir $\vec{\alpha_{l+1}}$ à partir de $\vec{X_l}$:

$g_l : \mathbb{R}^M \to \mathbb{R}^{N_l}$

$$\vec{X_l} \mapsto \begin{pmatrix} g_l^1(\vec{X_l}) = s_{e_1}^l(X) + \underset{m \in E_{\to e_1}^{l-1}}{\text{logadd}} \left(g_{m,e_1} + b^l(m,e_1) + \alpha_{m,l-1}\right) \\ \vdots \\ g_l^i(\vec{X_l}) = s_{e_i}^l(X) + \underset{m \in E_{\to e_i}^{l-1}}{\text{logadd}} \left(g_{m,e_i} + b^l(m,e_i) + \alpha_{m,l-1}\right) \\ \vdots \\ g_l^{N_{l+1}}(\vec{X_l}) = s_{e_{N_{l+1}}}^l(X) + \underset{m \in E_{\to e_{N_{l+1}}}^{l-1}}{\text{logadd}} \left(g_{m,e_{N_{l+1}}} + b^l(m,e_{N_{l+1}}) + \alpha_{m,l-1}\right) \end{pmatrix} = \vec{\alpha_l}. \tag{3.92}$$

Nous définissons alors

$$\begin{aligned} &\hat{S}_X^l : \mathbb{R}^M \to \mathbb{R} \\ &\vec{X_l} \mapsto f_T \circ \cdots \circ f_l \circ g_l(\vec{X_l}) = \hat{S}_{\text{L2G}}. \end{aligned} \tag{3.93}$$

Le théorème de dérivation des fonctions composées nous donne :

$$J_{\hat{S}_X^l}(\vec{X_l}) = J_{f_T \circ \cdots \circ f_l \circ g_l}(\vec{X_l}) \tag{3.94}$$

$$= J_{f_T \circ \cdots \circ f_l}(g_l(\vec{X_l}))\ J_{g_l}(\vec{X_l}) \tag{3.95}$$

$$= J_{f_T \circ \cdots \circ f_l}(\vec{\alpha_l})\ J_{g_l}(\vec{X_l}) \tag{3.96}$$

$$= J_{\hat{S}^l}(\vec{\alpha_l})\ J_{g_l}(\vec{X_l}). \tag{3.97}$$

Nous avons déjà une méthode pour calculer $J_{\hat{S}^l}(\vec{\alpha_l})$. Pour $J_{g_l}(\vec{X_l})$ nous obtenons :

$$J_{g_l}(\vec{X_l}) = \begin{bmatrix} \frac{\partial g_l^1}{\partial s_1^l}(\vec{X_l}) & \cdots & \frac{\partial g_l^1}{\partial s_M^l}(\vec{X_l}) \\ \vdots & \frac{\partial g_l^i}{\partial s_j^l}(\vec{X_l}) & \vdots \\ \frac{\partial g_l^{N_{l+1}}}{\partial s_1^l}(\vec{X_l}) & \cdots & \frac{\partial g_l^{N_{l+1}}}{\partial s_M^l}(\vec{X_l}) \end{bmatrix}, \tag{3.98}$$

avec

$$\frac{\partial g_l^i}{\partial s_j^l}(\vec{X_l}) = \frac{\partial \left[s_{e_i}^l(X) + \underset{m \in E_{\to e_i}^l}{\text{logadd}} \left(g_{m,e_i} + b^l(m,e_i) + \alpha_{m,l}\right) \right]}{\partial s_j^l}(\vec{X_l}) \tag{3.99}$$

$$= \begin{cases} 1 & \text{si } e_i = j \\ 0 & \text{sinon.} \end{cases} \tag{3.100}$$

En reprenant l'équation (3.97), un élément de $J_{\hat{S}^l_X}(\vec{X}_l)$ s'écrit :

$$\frac{\partial \hat{S}_{\text{L2G}}}{\partial s_i^l}(\vec{X}_l) = \sum_{k'=1}^{N_l} \beta_{k',l} \times \frac{\partial g_l^{k'}}{\partial s_i^l}(\vec{X}_l) \tag{3.101}$$

$$= \sum_{k'=1}^{N_l} \beta_{k',l} \times \mathbb{1}_{[e_{k'}=i]}. \tag{3.102}$$

Finalement, pour obtenir les dérivées partielles de la fonction de coût par rapport à chacun des poids du modèle acoustique à la trame l, il faut additionner les β de la trame l qui utilisent ce score acoustique. Nous pouvons alors formuler l'Algorithme 4 pour calculer toutes les dérivées partielles lors de la phase de rétropropagation.

Pour chacune des T trames il faut calculer au maximum B β. Pour obtenir un β de l'état m, il faut agréger les $\text{Card}(E^{l+1}_{\leftarrow m})$ β de la trame suivante. La taille maximale de cet ensemble dépend de la structure du graphe $\hat{G}_{\text{L2G}}$, notamment du demi-degré intérieur maximal MaxDegInt. Ce nombre ne peut pas excéder B mais en pratique il est bien plus petit. La complexité algorithmique de la phase de propagation arrière de Lead2Gold est donc $\mathcal{O}(T \times B \times \text{MaxDegInt})$.

3.3. Mise en place des expériences et observations préliminaires

3.3.1. Introduction

Dans ce chapitre nous utilisons des données synthétiques pour évaluer l'algorithme Lead2Gold. Pour cela, nous utilisons le jeu de données Librispeech qui est considéré comme ayant des transcriptions correctes. Ensuite nous ajoutons des erreurs à ces transcriptions à partir d'un modèle de bruit de transcription connu. Ainsi nous utiliserons l'algorithme Lead2Gold avec le bon modèle de bruit sur des transcriptions erronées. Cela permet d'évaluer l'algorithme indépendamment de l'estimation (potentiellement erronée) du modèle de bruit. Pour générer un modèle de bruit de transcription, nous utilisons les prédictions incorrectes d'un modèle de RAP peu performant. Nous faisons l'hypothèse ici que ces erreurs ressemblent à celles produites par des annotateurs humains. En effet, comme nous les verrons au chapitre 5, les erreurs produites par les systèmes de RAP sont souvent phonétiquement semblables à la transcription correcte, tout comme les erreurs faites par les humains.

Présentation du jeu de données Librispeech Dans ce manuscrit nous utiliserons la plupart du temps le corpus Librispeech [Panayotov et al., 2015]. C'est un corpus d'environ 1 000 heures de parole en anglais échantillonnées à 16 kHz. Ces données proviennent du projet LibriVox qui contient des livres audio. Les données ont été segmentées en plusieurs

Algorithme 4 : Calcul des dérivées partielles avec l'algorithme de rétropropagation de Lead2Gold.

`/* Initialisation */`

$\beta_{i,T} = \dfrac{\exp\left(\alpha_{i,T} + b^T(i)\right)}{\sum_{e \in E^T} \exp\left(\alpha_{e,T} + b^T(e)\right)} \; \forall i \in [1, \ldots, N_T]$

$\dfrac{\partial \hat{S}_{\text{L2G}}}{\partial s_i^T}(X) = \sum_{k'=1}^{N_T} \beta_{k',T} \times \mathbb{1}_{[e_{k'}=i]} \; \forall i \in [1, \ldots, M]$

pour chaque trame $l \in [T-1, \ldots, 1]$ **faire**

  `/* Calcul de` $\beta_{m,l}$ `*/`

  $\dfrac{\partial \hat{S}_{\text{L2G}}}{\partial s^l_{y_j}}(X) = 0 \; \forall j \in [1, \ldots, M]$

  pour chaque état $m \in \hat{E}^l$ **faire**

    $\beta_{m,l} = 0$

    pour chaque état $k \in E^{l+1}_{\leftarrow m}$ **faire**

      $\beta_{m,l} = \beta_{m,l} + \beta_{k,l+1} \times \dfrac{\exp\left(s^l_{e_k}(X) + g_{m,e_k} + b^l(m, e_k) + \alpha_{m,l}\right)}{\exp(\alpha_{k,l+1})}.$

    fin

    Avec y^j le symbole de m,

    $\dfrac{\partial \hat{S}_{\text{L2G}}}{\partial s^l_{y_j}}(X) = \dfrac{\partial \hat{S}_{\text{L2G}}}{\partial s^l_{y_j}}(X) + \beta_{m,l}.$

  fin

fin

retourner $\dfrac{\partial \hat{S}_{\text{L2G}}}{\partial s^l_{y_j}}(X) \; \forall j \in [1, \ldots, M], \forall l \in [1, \ldots, T]$

milliers de phrases ou groupes de phrases dont la durée maximale est de 35 secondes. Ces segments ont ensuite été alignés sur le texte du livre correspondant. L'ensemble d'apprentissage comporte trois parties : *train-clean-100*, *train-clean-360* et *train-other-500*. Les deux premières sont composées de respectivement 100 et 360 heures de parole considérées plus faciles à reconnaître. Pour cela les locuteurs ont été classés selon le WER (*Word Error Rate*) obtenu par un modèle de RAP sur les paroles qu'ils ont prononcées. Les données provenant de la meilleure moitié des locuteurs ont été mises dans *train-clean-100* et *train-clean-360*. Le reste forme l'ensemble *train-other-500*. Nous avons également à disposition les ensembles de développement *dev-clean* et *dev-other*. Avec les performances obtenues sur ces deux derniers, nous pouvons calibrer tous les choix d'architecture et d'hyper-paramètres. Enfin, pour mesurer les performances finales, nous avons les ensembles de test *test-clean* et *test-other*. Chaque ensemble de développement et de test est composé d'un peu plus de 5 heures de parole.

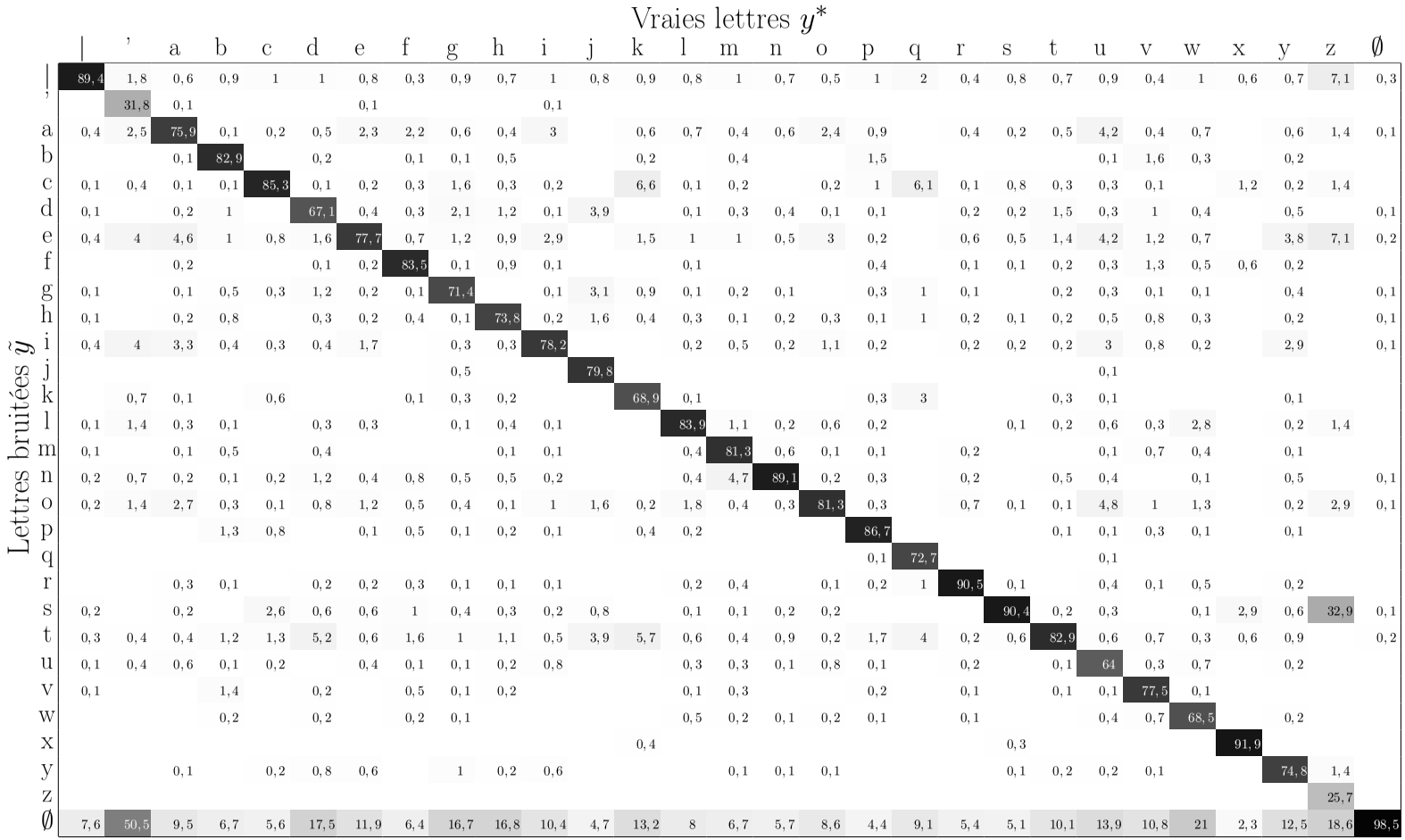
3.3.2. Obtention du modèle de bruit de transcription

Apprentissage d'un modèle de RAP peu performant Nous entraînons et évaluons le modèle de RAP peu performant sur le jeu de données WSJ (*Wall street Journal*) [Paul and Baker, 1992] dont l'ensemble d'apprentissage *si284* contient 82 heures de parole. Nous utilisons un autre jeu de données que Librispeech car nous souhaitons que l'obtention du modèle de bruit soit indépendante du jeu de données utilisé pour évaluer la méthode. Ce modèle a la même architecture et utilise les mêmes hyper-paramètres que celui utilisé pour évaluer la méthode Lead2Gold (voir partie 3.4.1). Nous utilisons la fonction de coût ASG pour l'apprentissage. Le modèle est peu performant car nous arrêtons l'apprentissage avant la convergence de la fonction de coût. Pour cela, l'apprentissage s'effectue pendant 200 itérations sur le jeu de données WSJ et nous sauvegardons une copie du modèle toutes les 5 itérations. Nous évaluons chaque copie sur l'ensemble de développement *nov93dev* sans modèle de langage et calculons le LER (*Letter Error Rate*). Nous gardons les copies des itérations 40, 50 et 110 dont le LER est respectivement proche de 30, 20 et 10% [1]. Nous appelons M30, M20 et M10 ces trois modèles.

Décodage Nous évaluons chacun de ces trois modèles sur les ensembles *nov93dev* et *nov92* de WSJ (836 phrases en tout) et décodons le résultat avec l'algorithme de Viterbi. Aucun modèle de langage n'est ici utilisé. Nous obtenons des transcriptions bruitées que nous comparons aux transcriptions considérées sans erreur présentes dans le jeu de données. Pour cela, nous observons les transformations de substitution, d'insertion et de délétion nécessaires pour transformer la transcription sans erreur avec sa version erronée. Ces transformations sont celles utilisées pour calculer la distance de Levenshtein au niveau de la lettre (ou LER) entre ces deux transcriptions. Enfin, nous calculons la distribution empirique basée sur la fréquence de chaque transformation pour obtenir les $p(\tilde{y}|y*)$, $p(\tilde{y}|\emptyset)$ et $p(\emptyset|y*)$. Ces probabilités forment alors le modèle de bruit de transcription $P(\tilde{Y}|Y^*)$. Chacun des trois modèles de RAP (M30, M20 et M10) donne ainsi naissance à un modèle de bruit différent.

Augmentation ou réduction de la quantité d'erreurs Pour générer plus de variations de modèles de bruit, nous augmentons ou réduisons artificiellement le nombre de substitutions, d'insertions et de délétions en multipliant le nombre effectif par un facteur multiplicatif $f \in \{0, 5, 1, 2\}$ avant de calculer les probabilités correspondantes. Nous obtenons ainsi 9 modèles de bruit que nous notons « M* f* ». Nous créons aussi 9 modèles de bruit « S M* f* » qui ignorent les insertions et les délétions. Nous obtenons ainsi 18 modèles de bruit différents en tout. Par exemple, pour obtenir le modèle de bruit « S M10 f2 », nous calculons le nombre de transformations nécessaires pour passer des transcriptions sans erreur à celles obtenues en décodant le modèle M10, puis nous multiplions le nombre de substitutions par 2, nous remplaçons le nombre d'insertions et de délétions par 0 et nous calculons les probabilités correspondantes.

1. Plus précisément, le LER respectif de ces trois modèles est égal à 30,79, 20,83 et 9,88%.



	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	∅
|	89,4	1,8	0,6	0,9	1	1	0,8	0,3	0,9	0,7	1	0,8	0,9	0,8	1	0,7	0,5	1	2	0,4	0,8	0,7	0,9	0,4	1	0,6	0,7	7,1	0,3
'		31,8	0,1				0,1				0,1																		
a	0,4	2,5	75,9	0,1	0,2	0,5	2,3	2,2	0,6	0,4	3		0,6	0,7	0,4	0,6	2,4	0,9		0,4	0,2	0,5	4,2	0,4	0,7		0,6	1,4	0,1
b			0,1	82,9		0,2		0,1	0,1	0,5			0,2		0,4			1,5					0,1	1,6	0,3		0,2		
c	0,1	0,4	0,1	0,1	85,3	0,1	0,2	0,3	1,6	0,3	0,2		6,6	0,1	0,2		0,2	1	6,1	0,1	0,8	0,3	0,3	0,1		1,2	0,2	1,4	
d	0,1		0,2	1		67,1	0,4	0,3	2,1	1,2	0,1	3,9		0,1	0,3	0,4	0,1	0,1		0,2	0,2	1,5	0,3	1	0,4		0,5		0,1
e	0,4	4	4,6	1	0,8	1,6	77,7	0,7	1,2	0,9	2,9		1,5	1	1	0,5	3	0,2		0,6	0,5	1,4	4,2	1,2	0,7		3,8	7,1	0,2
f			0,2			0,1	0,2	83,5	0,1	0,9	0,1			0,1				0,4		0,1	0,1	0,2	0,3	1,3	0,5	0,6	0,2		
g	0,1		0,1	0,5	0,3	1,2	0,2	0,1	71,4		0,1	3,1	0,9	0,1	0,2	0,1		0,3	1	0,1		0,2	0,3	0,1	0,1		0,4		0,1
h	0,1		0,2	0,8		0,3	0,2	0,4	0,1	73,8	0,2	1,6	0,4	0,3	0,1	0,2	0,3	0,1	1	0,2	0,1	0,2	0,5	0,8	0,3		0,2		0,1
i	0,4	4	3,3	0,4	0,3	0,4	1,7		0,3	0,3	78,2			0,2	0,5	0,2	1,1	0,2		0,2	0,2	0,2	3	0,8	0,2		2,9		0,1
j									0,5			79,8											0,1						
k		0,7	0,1		0,6			0,1	0,3	0,2			68,9	0,1				0,3	3			0,3	0,1				0,1		
l	0,1	1,4	0,3	0,1		0,3	0,3		0,1	0,4	0,1			83,9	1,1	0,2	0,6	0,2			0,1	0,2	0,6	0,3	2,8		0,2	1,4	
m	0,1		0,1	0,5		0,4				0,1	0,1			0,4	81,3	0,6	0,1	0,1		0,2			0,1	0,7	0,4		0,1		
n	0,2	0,7	0,2	0,1	0,2	1,2	0,4	0,8	0,5	0,5	0,2			0,4	4,7	89,1	0,2	0,3		0,2		0,5	0,4		0,1		0,5		0,1
o	0,2	1,4	2,7	0,3	0,1	0,8	1,2	0,5	0,4	0,1	1	1,6	0,2	1,8	0,4	0,3	81,3	0,3		0,7	0,1	0,1	4,8	1	1,3		0,2	2,9	0,1
p				1,3	0,8		0,1	0,5	0,1	0,2	0,1		0,4	0,2				86,7				0,1	0,1	0,3	0,1		0,1		
q																		0,1	72,7				0,1						
r			0,3	0,1		0,2	0,2	0,3	0,1	0,1	0,1			0,2	0,4		0,1	0,2	1	90,5	0,1		0,4	0,1	0,5		0,2		
s	0,2		0,2		2,6	0,6	0,6	1	0,4	0,3	0,2	0,8		0,1	0,1	0,2	0,2				90,4	0,2	0,3		0,1	2,9	0,6	32,9	0,1
t	0,3	0,4	0,4	1,2	1,3	5,2	0,6	1,6	1	1,1	0,5	3,9	5,7	0,6	0,4	0,9	0,2	1,7	4	0,2	0,6	82,9	0,6	0,7	0,3	0,6	0,9		0,2
u	0,1	0,4	0,6	0,1	0,2		0,4	0,1	0,1	0,2	0,8			0,3	0,3	0,1	0,8	0,1		0,2		0,1	64	0,3	0,7		0,2		
v	0,1			1,4		0,2		0,5	0,1	0,2				0,1	0,3			0,2		0,1		0,1	0,1	77,5	0,1				
w				0,2		0,2		0,2	0,1					0,5	0,2	0,1	0,2	0,1		0,1			0,4	0,7	68,5		0,2		
x													0,4								0,3					91,9			
y			0,1		0,2	0,8	0,6		1	0,2	0,6				0,1	0,1	0,1				0,1	0,2	0,2	0,1			74,8	1,4	
z																												25,7	
∅	7,6	50,5	9,5	6,7	5,6	17,5	11,9	6,4	16,7	16,8	10,4	4,7	13,2	8	6,7	5,7	8,6	4,4	9,1	5,4	5,1	10,1	13,9	10,8	21	2,3	12,5	18,6	98,5

TABLEAU 3.1. – Le modèle de bruit de transcription « M20 f1 ». Les probabilités sont affichées en pourcentage. Le symbole « | » représente l'espace entre les mots. La ligne ∅ contient les probabilités de délétion et la colonne ∅ les probabilités d'insertion. $p(\emptyset|\emptyset)$ est la probabilité de ne pas insérer un nouveau symbole.

L'étude des modèles « S » est intéressante car cela simplifie la complexité de la recherche en faisceau de Lead2Gold. Nous obtenons des résultats nettement meilleurs pour ces cas.

Exemple Le Tab. 3.1 montre le modèle de bruit « M10 f2 ». Nous pouvons observer que les probabilités de substitution $p(\tilde{y} = \text{s}|y^* = \text{z})$ et $p(\tilde{y} = \text{c}|y^* = \text{k})$ sont importantes car ces lettres produisent souvent le même son. Dans cette configuration, les insertions sont peu probables car le modèle de RAP a tendance à produire des transcriptions trop courtes.

3.3.3. Création des jeux de données bruitées

Nous utilisons les 18 modèles de bruit de transcription pour créer différentes versions de transcriptions erronées. Pour cela, nous utilisons l'ensemble *train-clean-100* de Librispeech. Nous nous limitons à 100 heures car notre approche est assez coûteuse en calculs. Cette quantité est tout de même suffisante pour l'apprentissage d'un modèle de RAP et pour effectuer des expériences intéressantes. Pour chacune des transcriptions Y^* du jeu de données, nous appliquons le modèle de bruit pour tirer aléatoirement une nouvelle transcription erronée $\tilde{Y}$. Le Tab. 3.2 propose un exemple de transcription corrompue selon chacun des modèles de bruit.

3.3.4. Évaluation

Pour développer et tester l'algorithme Lead2Gold, nous utilisons les ensembles *dev-clean* et *test-clean* de Librispeech. Nous calculons le WER (*Word Error Rate*) entre les transcriptions décodées par le modèle de RAP appris par Lead2Gold et la transcription de référence. Le WER sur *test-clean* est calculé avec ou sans modèle de langage. L'utilisation d'un modèle de langage nécessite de décoder les résultats du modèle de RAP avec une recherche en faisceau.

Le modèle de langage est un 4-gramme. Les hyper-paramètres du décodeur sont choisis selon les performances obtenues sur *dev-clean* par un modèle de RAP appris sur *train-clean-100* avec la fonction de coût ASG. Les hyper-paramètres du décodeur ainsi trouvés sont :

— nombre d'hypothèses à explorer : 2 500.

— score minimal d'une hypothèse : 50.

— poids attribué pour la contribution du modèle de langage : 2,66.

— pénalité d'insertion d'un mot : 1,33.

— pénalité d'insertion d'un silence (espacement de mots sur plusieurs trames) : −1,33.

Pour effectuer une comparaison, nous évaluons le WER de base de modèles appris sur les jeux de données bruitées avec la fonction de coût ASG (sans l'utilisation de Lead2Gold). Cela nous indique le seuil de performance qu'il faut dépasser. Nous indiquons

Mode	Modèle	f	Transcription transformée $\tilde{Y}$
Substitutions uniquement	M10	0,5	superior industries international fell three and a quarter to fifteen ang a half
		1	superior intustries international fell three and a quarter to fifteen and a half
		2	saperior indastries itternational fall tnree and a cuartor to fifteen and o half
	M20	0,5	superaor indusyries intirnational fell phree and a quarter to fifteen and a half
		1	superiir induseries internationae fell three and a quarter to fifteen and a half
		2	superiir endustries enternational fill three alg i qoarter ta fiateen emd a lalf
	M30	0,5	superior induslries intsrnational fell three and a quarter te fnfteen und a halv
		1	superigr industoyes itternational fill thria and a quarter to fifteen and a half
		2	sspirior entostriee intertational fell throi ant n kuartir to fnfteen any e talf
Substitutions, insertions et délétions	M10	0,5	superior endustries intdertnational fell three ad a quarer to fiften and a half
		1	supereor indstrias international fell three and eiquareer o fifteen ad a half
		2	sup'rior inustriec international fell tree and ai q'uarter t fifteen and a half
	M20	0,5	superier indostris internationat fel thee ag a qoardr toe fifteen onda half
		1	sperior intustoies internatenul fel tre and a uarter to fifteenand a al
		2	supeaior ngudtries nternainhl vell thrie an a aesr tr ftdn gand eal
	M30	0,5	sproe indastries international elb threnean a qearder tw fiftee and a haf
		1	sipro industres itertitonal fell aree ad a qarter o tasateen end a half
		2	srir indtsiesitnasiona fll three ataiataer wafhee o i ef

TABLEAU 3.2. – Exemple d'utilisation des modèles de bruit. Pour chacun des modèles de bruit, nous partons de la phrase sans erreur Y^* = « *superior industries international fell three and a quarter to fifteen and a half* » pour générer aléatoirement une phrase erronée $\tilde{Y}$.

également le WER d'un modèle « oracle » appris sur les données non-bruitées. Cela nous indique la borne de performance que nous ne pouvons pas dépasser.

3.3.5. Exemple qualitatif de résultat de Lead2Gold

Nous pouvons observer quelles sont les meilleures hypothèses gardées pendant la recherche en faisceau de Lead2Gold. Un exemple est présenté dans le Tab. 3.3. Dans l'algorithme Lead2Gold, les hypothèses sont fusionnées si la dernière lettre produite est identique et si le curseur est sur le même symbole de $\tilde{Y}$. Donc une hypothèse ne correspond normalement pas à une transcription en particulier, mais plutôt à un mélange de transcriptions. Pour pouvoir observer une transcription en particulier, nous partons du graphe $\hat{G}_{\text{L2G}}$ obtenu par la méthode Lead2Gold et nous appliquons une nouvelle recherche en faisceau sur celui-ci, cette fois-ci en ne fusionnant les hypothèses que si elles représentent la même transcription. Nous stockons le score agrégé des chemins menant à une transcription donnée et nous le normalisons avec une opération softmax pour obtenir son poids. L'exemple montre que notre méthode peut retrouver des bonnes hypothèses avec un poids assigné qui semble raisonnable. Dans cet exemple, la meilleure transcription trouvée correspond à la transcription non-corrompue.

	Transcription	Poids	LER
Véritable transcription	could not give his hand to the bride	-	-
Transcription bruitée $\tilde{Y}$	ool not ive his han to the rride	-	15,8
Meilleures transcriptions hypothèses Y^*	could not give his hand to the bride	0,22	0
	could not ive his hand to the bride	0,13	2,6
	could not give his hand to the brie	0,05	2,6
	could not ive his hand to the brie	0,03	5,3
	coul not give his hand to the bride	0,029	2,6
	ould not give his hand to the bride	0,024	2,6
	could not cive his hand to the bride	0,023	2,6
	could not give his and to the bride	0,022	2,6
	coud not give his hand to the bride	0,022	2,6
	could not give is hand to the bride	0,018	2,6
	could not giv his hand to the bride	0,018	2,6

TABLEAU 3.3. – Exemple des 10 meilleures (selon le poids associé) hypothèses Y^* obtenues pendant l'algorithme de recherche en faisceau de Lead2Gold. La méthode est appliquée sur une transcription bruitée $\tilde{Y}$ du jeu de données « M20 f1 ». Nous indiquons le poids associé à chacune des hypothèses ainsi que le LER entre celle-ci et la véritable transcription.

3.4. Expériences

3.4.1. Protocole expérimental

Nous implémentons Lead2Gold dans la bibliothèque Flashlight [Pratap et al., 2019], un outil spécialisé pour l'apprentissage profond de modèles de RAP écrit en C++. Nous apprenons un modèle de RAP pour chacun des 18 modèles de bruit. Nous effectuons une phase de pré-apprentissage des modèles en utilisant la fonction de coût standard ASG pendant 1 000 itérations sur le jeu de données. Le modèle de bruit n'est pas utilisé pendant cette étape. Nous procédons de cette manière car la fonction de coût Lead2Gold est 15 fois plus lente que ASG et est jusqu'à 30 fois plus lente si nous utilisons un nombre d'hypothèses $B = 300$ dans la recherche en faisceau. Après cette étape, nous affinons l'apprentissage en activant Lead2Gold pour 200 itérations dans le cas où seules les substitutions sont autorisées. Dans le cas général, Lead2Gold n'est activé que pour une itération (voir partie 3.4.2). Le modèle « oracle » qui utilise *train-clean-100* est appris pendant 1 200 itérations avec ASG. Pendant les 200 dernières itérations, le pas d'apprentissage est divisé par 2 afin d'obtenir le meilleur WER possible.

Architecture du modèle de RAP L'architecture que nous utilisons se base sur celle fournie par la bibliothèque Flashlight en 2019. C'est une architecture *Gated ConvNet* (voir partie 2.3.2). Le choix des hyper-paramètres tels que le nombre de couches, la taille

des couches ou encore le *stride*, dépend de la taille du jeu de données dont nous disposons. Les paramètres fournis par Flashlight sont adaptés pour l'utilisation des 1 000 heures de Librispeech.

Avant d'aboutir au choix d'utiliser *train-clean-100* pour l'évaluation de Lead2Gold, nous avions conduit des expériences de recherche d'architecture en fonction de la taille du jeu de données. Plus précisément, nous avons déterminé quels hyper-paramètres de l'architecture *Gated ConvNet* sont les plus adaptés pour un jeu de données de taille 2,5, 10, 40, 160, 640 et 960 heures venant de Librispeech. Ces ensembles de données sont constitués d'un mélange de *train-clean-100*, *train-clean-360* et *train-other-500*. Chaque test d'architecture utilise 4 GPU sur une durée allant de 10 heures à 3 jours. La meilleure architecture pour les 960 heures possède 219 millions de paramètres. Celle pour les 2,5 heures seulement 4,36 millions. La mise en place de ces expériences demande beaucoup de ressources GPU. C'est pourquoi, pour l'évaluation de notre méthode sur *train-clean-100*, nous prenons la meilleure architecture trouvée sur les 160 heures.

Cette architecture contient 17 couches de convolution 1D *Gated* avec une taille de noyau de convolution fixée à 13 pour chaque couche et une valeur de *dropout* fixée à 0,25. Le nombre de noyaux pour chaque couche est augmenté linéairement de 100 à 200 entre la première et la dernière couche. La première couche utilise un *stride* de 2 pour réduire le nombre de trames à traiter et prend en entrée 40 coefficients *log mel-filterbanks* par trame. Ces couches de convolution sont suivies de deux couches totalement connectées. La première augmente la dimension du vecteur d'entrée (venant de la dernière couche de convolution) de 200 à 400. La seconde réduit la dimension de 400 à 29 qui est le nombre de symboles que nous utilisons.

Cette architecture contient seulement 10 millions de paramètres, ce qui est suffisant pour notre jeu de données et permet d'effectuer des expériences plus rapidement. Nous obtenons des performances correctes. Sur *dev-clean*, le WER est évalué à 16,9%, ce qui est proche des 14,7% obtenus par Lüscher et al. [2019] pour des modèles de bout en bout. Avec un modèle de langage nous obtenons un WER de 9,5% que nous pouvons comparer au 7,3% de Billa [2017] dont le but est d'atteindre les meilleures performances dans cette configuration.

Apprentissage Les hyper-paramètres d'apprentissage sont fournis par Flashlight. Nous adaptons seulement le pas d'apprentissage en fonction de la taille du *batch*. Nous utilisons l'optimiseur SGD avec des gradients normalisés et tronqués à une valeur maximale de 0,05. Le pas d'apprentissage est réglé à 8 pendant la phase de pré-apprentissage. Pour les paramètres de transition de symboles $g(\pi_t|\pi_{t-1})$, nous utilisons un pas d'apprentissage de 0,004. Nous utilisons une taille de *batch* de 320. Pour le calcul de Lead2Gold, nous traitons chaque transcription du *batch* en parallèle en assignant un CPU à chaque transcription. Pendant la phase de pré-apprentissage nous utilisons 8 GPU, ce qui nous permet d'effectuer une itération sur le jeu de données en approximativement 30 secondes. Quand

l'apprentissage se poursuit avec Lead2Gold, le nombre de GPU importe peu car le temps passé à calculer la fonction de coût représente la majorité du temps d'une itération. Lead2Gold est en effet uniquement implémenté sur CPU. Pour toutes les expériences nous réglons le nombre d'hypothèses du faisceau à 300. L'augmenter plus n'a que peu d'impact sur les résultats. Une itération de Lead2Gold dure entre 7 et 15 minutes.

Le paramètre du modèle de bruit α_{bruit} doit être réglé. Dans toutes nos expériences, le modèle ne converge que si celui-ci est en dessous de 0,7. En général, plus la quantité de bruit dans les transcriptions est importante, plus ce paramètre doit être réduit. Nous cherchons quel est le meilleur α_{bruit} en fonction du WER obtenus sur *dev-clean* en testant avec un pas de 0,1 tous les α_{bruit} entre 0 et 1.

3.4.2. Résultats

Cas des opérations de substitution uniquement Le Tab. 3.4 montre les résultats de notre méthode en n'activant que les opérations de substitution. Nous gardons un pas d'apprentissage à 8 et α_{bruit} est réglé à 0,5 sauf pour « S M30 f2 » où il est réglé à 0,3. Dans tous les cas, Lead2Gold obtient un meilleur WER que la fonction de coût ASG, sauf pour « S M10 f0,5 » où les résultats ne sont pas significativement différents sans modèle de langage. Nous pouvons presque atteindre le WER du modèle oracle lorsque le niveau de bruit est faible. Le gain relatif de Lead2Gold est plus important lorsque le niveau de bruit est important. Pour le modèle appris sur « S M30 f2 », nous obtenons une réduction absolue du WER de 5,1%. Il est à noter que dans certain cas, l'ajout d'un modèle de langage fait décroitre le WER pour les modèles appris avec ASG. Une explication possible est que nous n'avons pas réglé les hyper-paramètres de décodage pour chaque configuration. La Fig. 3.5 montre l'évolution du LER lorsque Lead2Gold est activé ou non sur le jeu de données « S M30 f2 ».

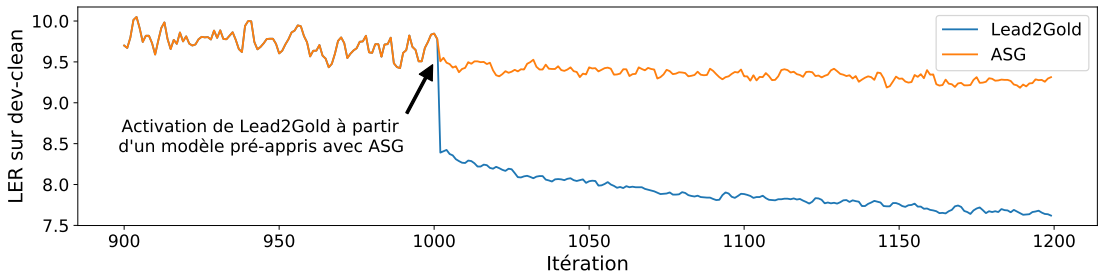


FIGURE 3.5. – Évaluation du LER au cours de l'apprentissage sur le jeu de données « S M30 f2 ». À l'itération 1 000, nous activons Lead2Gold. Par rapport à l'unique utilisation de ASG, Lead2Gold est capable de réduire la perte de performance (ici le LER) due aux erreurs dans les transcriptions.

Apprentissage	Test					
	dev-clean		test-clean		test-clean+LM	
	ASG	L2G	ASG	L2G	ASG	L2G
S M10 f0,5	**17, 5**	**17, 1**	**17, 7**	**17, 3**	12, 1	**10, 2**
S M10 f1	17, 9	**17, 1**	18, 1	**17, 2**	23, 4	**10, 3**
S M10 f2	18, 8	**17, 4**	18, 8	**18, 6**	21, 4	**10, 4**
S M20 f0,5	18, 1	**17, 1**	18, 3	**17, 5**	15, 0	**10, 4**
S M20 f1	19, 0	**17, 5**	19, 2	**18, 1**	22, 3	**10, 7**
S M20 f2	20, 6	**18, 2**	20, 9	**18, 6**	49, 1	**11, 2**
S M30 f0,5	18, 3	**17, 3**	18, 8	**17, 6**	19, 9	**10, 6**
S M30 f1	20, 2	**18, 2**	20, 4	**18, 3**	41, 1	**11, 0**
S M30 f2	24, 8	**19, 6**	25, 1	**20, 0**	80, 1	**12, 7**
train-clean-100	16, 9	-	17, 3	-	10, 0	-

TABLEAU 3.4. – WER (%) atteint par les modèles de RAP appris sur des jeux de données bruitées en activant Lead2Gold avec uniquement les substitutions. Nous effectuons le test de significativité MAPSSWE (*Matched Pairs Sentence-Segment Word Error*) [Gillick and Cox, 1989] entre les WER obtenus avec ASG et Lead2Gold. Le résultat est noté en gras s'il est significativement meilleur avec au moins une valeur de confiance p de 0,05. Les deux résultats sont en gras s'ils ne sont pas significativement différents.

Apprentissage	Test					
	dev-clean		test-clean		test-clean+LM	
	ASG	L2G	ASG	L2G	ASG	L2G
M10 f0,5	**18, 2**	**18, 5**	**18, 8**	19	13, 7	**12, 8**
M10 f1	**19, 5**	**20, 1**	**19, 5**	20, 3	23, 4	**19, 4**
M10 f2	**21, 7**	**22, 6**	**22, 2**	**22, 8**	59, 9	**41, 7**
M20 f0,5	**20, 2**	**21, 1**	**20, 7**	21, 8	28, 6	**22, 6**
M20 f1	23, 9	**22, 8**	24, 3	**23, 2**	71, 2	**58, 3**
M20 f2	44, 0	**31, 2**	44, 6	**31, 7**	96, 5	**68, 9**
M30 f0,5	23, 3	**22, 0**	23, 4	**22, 0**	60, 8	**39, 8**
M30 f1	39, 1	**28, 6**	39, 6	**28, 7**	95, 3	**76, 1**
M30 f2	87, 7	**46, 4**	87, 9	**46, 9**	99, 0	**84, 8**
train-clean-100	16, 9	-	17, 3	-	10, 0	-

TABLEAU 3.5. – WER (%) atteint par les modèles de RAP sur des jeux de données en activant Lead2Gold et en autorisant toutes les opérations de substitution, d'insertion et de délétion. La notation pour la significativité des résultats est la même que pour le Tab. 3.4.

Cas général Le Tab. 3.5 montre les résultats lorsque les opérations de substitution, d'insertion et de délétion sont autorisées. Le pas d'apprentissage est réglé à 1 et α_{bruit} à 0,1. Dans cette configuration, Lead2Gold n'améliore pas les résultats pour tous les cas. Nous observons tout de même une nette amélioration du WER lorsque la quantité de bruit dans les transcriptions est importante. De façon générale, nous observons des problèmes de convergence importants dans cette configuration. Dans certains cas, le WER s'améliore durant les premières itérations puis se détériore durant les suivantes. C'est pourquoi nous reportons ici les résultats obtenus après l'activation de Lead2Gold pendant une itération.

3.4.3. Interprétation

Dans le cas où les substitutions, insertions, et délétions sont activées, nous voulons essayer de comprendre pourquoi les résultats se détériorent après la première itération. Pour cela, nous observons les hypothèses présentes dans la recherche en faisceau. En particulier, nous étudions à quel rythme les lettres de la transcription bruitée sont consommées. L'idée est que les lettres doivent être consommées plus ou moins uniformément au cours des trames. Pour cela, pour une hypothèse du faisceau, nous calculons à la trame t :

$$\text{RatioLettre}(t, hyp) = \frac{\text{nombre de lettres consommées à la trame } t}{\text{nombre de lettres dans } \tilde{Y}}. \tag{3.103}$$

Nous voulons une mesure commune pour les B hypothèses. Nous pouvons les agréger en calculant :

$$\text{RatioLettre}(t) = \sum_{hyp} \text{RatioLettre}(t, hyp) \times w_{\text{hyp}}, \tag{3.104}$$

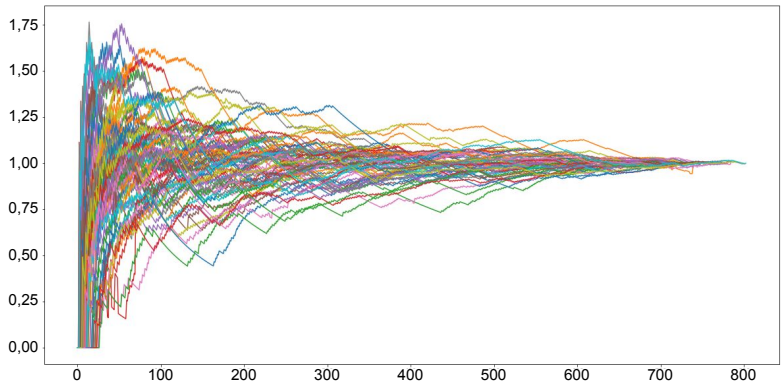
avec w_{hyp} un poids attribué à chaque hypothèse en fonction du score $\alpha_{e,t}$ de cette hypothèse à la trame t. w_{hyp} est obtenu en appliquant la fonction softmax sur les B scores. Nous voulons comparer RatioLettre(t) au ratio de la trame courante t par rapport au nombre total de trames T :

$$\text{RatioTrame}(t) = \frac{t}{T}. \tag{3.105}$$

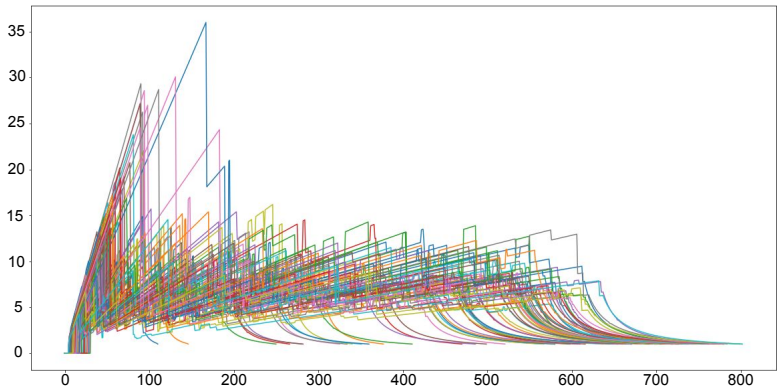
Nous calculons enfin $R(t)$, le rapport entre ces deux ratios à la trame t :

$$R(t) = \frac{\text{RatioTrame}(t)}{\text{RatioLettre}(t)}. \tag{3.106}$$

Si l'hypothèse d'uniformité de consommation de lettres le long des trames est vraie, alors $R(t)$ doit être autour de 1 pour chaque trame t. Nous prenons l'exemple de 80 transcriptions prises aléatoirement dans le cas « M20 f1 ». Nous choisissons ici $\alpha_{\text{bruit}} = 0,5$. La Fig. 3.6a montre l'évolution de $R(t)$ le long des trames dans le cas où seules les substitutions sont autorisées. La Fig. 3.6b montre cette évolution dans le cas général. Nous pouvons observer que dans ce cas, la répartition de la consommation des lettres de $\tilde{Y}$ n'est pas bonne.



(a) Seules les substitutions sont autorisées



(b) Toutes les transformations sont autorisées

FIGURE 3.6. – Évolution de $R(t)$ pour 80 transcriptions de « M20 f1 ». Dans le cas où seules les substitutions sont autorisées (a) $R(t)$ oscille autour de 1. Dans le cas général (b), $R(t)$ est largement supérieur à 1. Cela indique que les lettres ne sont pas assez consommées le long des trames. L'algorithme Lead2Gold force cependant toutes les lettres à être consommées. Les hypothèses des dernières trames doivent donc consommer chacune une lettre.

Des expériences complémentaires nous ont permis d'écarter l'hypothèse que ce mauvais résultat puisse être contourné en choisissant un meilleur α_{bruit}. Ainsi, nous avons fait l'expérience d'assigner un α_{bruit} différent en fonction de la transformation appliquée (substitution, insertion ou délétion). Nous avons alors trois hyper-paramètres à choisir. L'annexe C montre $R(t)$ pour différentes valeurs de ces hyper-paramètres. Pour obtenir un $R(t)$ autour de 1, il faut assigner une très petite valeur au α_{bruit} de l'opération d'insertion.

Pour le cas « M20 f1 », nous avons assigné les poids 0,5, 0,01 et 0,3 aux opérations de substitution, d'insertion et de délétion. Nous avons obtenu au bout de 37 itérations un WER de 21,08% sur *test-clean*. Nous avions obtenu 23,2% en utilisant un α_{bruit} unique de 0,1. La méthode fonctionne donc mieux de cette manière. Après l'itération 37, les

résultats se dégradent tout de même. Nous avons toujours un problème de convergence à la fin de l'apprentissage, mais celui-ci est retardé.

3.4.4. Conclusion

Dans ce chapitre, nous avons proposé une solution au problème d'erreurs dans les transcriptions en formulant une fonction de coût adéquate. La méthode est originale : elle repose sur l'exploitation d'un modèle de bruit pour générer un ensemble de transcritions choisies avec un algorithme de recherche en faisceau. Que ce soit pour la phase de propagation avant ou arrière, nous avons démontré méthodiquement la formulation de l'algorithme Lead2Gold. Nous avons conduit des expériences sur des jeux de données synthétiques pour tester et valider la méthode. Lead2Gold améliore grandement le WER dans les cas où les données d'apprentissage bruitées et le modèle de bruit ne contiennent que des substitutions. Dans le cas général où les données d'apprentissage bruitées et le modèle de bruit contiennent aussi des insertions et des délétions, nous observons la plupart du temps une amélioration malgré les problèmes de convergence.

En pratique, la méthode est plutôt lente même si celle-ci est implémentée en C++. Pour régler la contribution du modèle acoustique par rapport au modèle de bruit de transcription, nous avons ajouté les termes α_{bruit} qu'il faut soigneusement choisir.

La méthode repose sur l'hypothèse forte que les erreurs dans les transcriptions peuvent être modélisées avec un modèle de bruit simple au niveau de la lettre. Pour tester la méthode avec cette hypothèse, nous avons créé synthétiquement des jeux de données. Mais pour adapter la méthode à d'autres configurations plus réalistes, il faudrait réécrire en grande partie l'algorithme. Dans un cas réel, nous pouvons imaginer que les erreurs dans les transcriptions sont en effet plus difficiles à modéliser. L'unité minimale de symbole pour le modèle de bruit d'une telle configuration pourrait être des morceaux de mots ou des mots entiers.

Dans le chapitre suivant, nous proposons une solution pour rendre modulable notre méthode. Nous allons nous affranchir de l'écriture d'une recherche en faisceau de façon explicite en utilisant des wFST. Nous allons formuler la fonction de coût uniquement avec des compositions de différents wFST.

4. Extension modulaire de Lead2Gold basée sur les wFST

Dans ce chapitre, nous proposons d'utiliser les wFST pour créer des fonctions de coût similaires à l'algorithme Lead2gold. L'implémentation de celui-ci est dépendante des choix de conception que nous avons faits et il est difficile de changer ces choix sans revoir l'implémentation entièrement. L'utilisation des wFST permet de rendre la fonction de coût modulaire.

Nous pouvons facilement changer une fonctionnalité en modifiant le wFST qui modélise cette fonctionnalité. La partie 4.1 présente tous les éléments nécessaires pour intégrer chaque fonctionnalité avec un wFST. L'utilisation de wFST pour formuler la fonction de coût permet aussi d'intégrer de nouveaux modules tels que le lexique. La partie 4.2 présente ces nouvelles fonctionnalités. Le modèle de bruit de transcription est aussi intégré avec un wFST. La partie 4.3 présente comment celui-ci est créé. Enfin dans la partie 4.4, nous présentons des exemples d'utilisation de fonctions de coûts formulées avec des wFST. Nous mettrons en avant les difficultés rencontrées et les solutions possibles.

4.1. Formulation de la fonction de coût

4.1.1. Lien entre Lead2Gold et les wFST

Dans le chapitre 3, nous avons présenté l'algorithme Lead2Gold. Pour rappel, sa formulation sans approximation par une recherche en faisceau s'exprime de la façon suivante :

$$\mathrm{L2G}(\tilde{Y}) = Z - S_{\mathrm{L2G}}(\tilde{Y}) \tag{4.1}$$

$$= \operatorname*{logadd}_{\pi \in G_{\mathrm{full}}(T)} z_\pi(X) - \operatorname*{logadd}_{C=(Y^*,\pi^*,a) \in G_{\mathrm{L2G}}(\tilde{Y},T)} s(C|X), \tag{4.2}$$

avec $z_\pi(X)$ le score total d'un chemin π du graphe $G_{\mathrm{full}}(T)$ et $s(C|X)$ le score total d'un chemin du graphe $G_{\mathrm{L2G}}(\tilde{Y}, T)$. Ces scores s'expriment par :

$$z_\pi(X) = \sum_{t=1}^{T} s_{\pi_t}(X) + g_{\pi_{t-1},\pi_t} \tag{4.3}$$

$$s(C = (Y^*, \pi^*, a)|X) = z_{\pi^*}(X) + \log p(\tilde{Y}|Y^*). \tag{4.4}$$

Dans la partie 2.2, nous avons introduit les wFST différentiables. C'est un outil de manipulation de graphe dont les arcs portent un symbole d'entrée, un symbole de sortie et un poids. Nous y avons notamment défini la fonction Score(.). En notant $\pi = [\pi_1, \ldots, \pi_n]$ un chemin d'un wFST A dont les poids sont $[w_1, \ldots, w_n]$, $s(\pi)$ le score total de ce chemin, alors nous avons :

$$\text{Score}(A) = \underset{\pi \in A}{\text{logadd}}\ s(\pi) \tag{4.5}$$

$$= \underset{\pi \in A}{\text{logadd}} \sum_{i=1}^{n} w_i. \tag{4.6}$$

Nous pouvons utiliser la fonction Score(.) pour exprimer la fonction de coût de Lead2Gold. En effet, si l'on considère $G_{\text{full}}(T)$ et $G_{\text{L2G}}(\tilde{Y}, T)$ comme étant des wFST nous avons directement :

$$\text{L2G}(\tilde{Y}) = \text{Score}(G_{\text{full}}(T)) - \text{Score}(G_{\text{L2G}}(\tilde{Y}, T)). \tag{4.7}$$

De plus, les fonctions de coût CTC et ASG peuvent être construites de la même manière :

$$\text{CTC}(Y) = -\text{Score}(G_{\text{CTC}}(Y, T)) \tag{4.8}$$

$$\text{ASG}(Y) = \text{Score}(G_{\text{full}}(T)) - \text{Score}(G_{\text{ASG}}(Y, T)). \tag{4.9}$$

L'article de présentation de GTN [Hannun et al., 2020] formule également les fonctions de coûts ASG et CTC de cette manière.

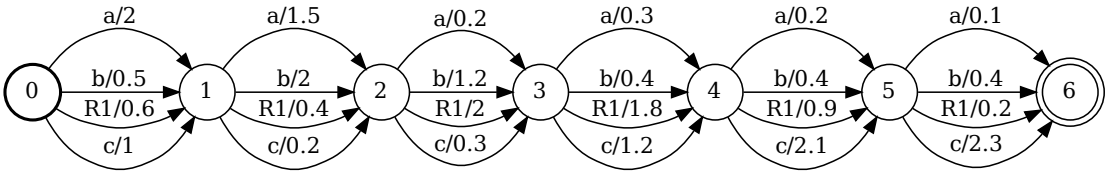
Pour décrire complètement les formulations des équations (4.7), (4.8) et (4.9), il faut encore construire les wFST sur lesquelles nous appliquons la fonction Score(.).

4.1.2. Construction de G_{full}, G_{CTC} et G_{ASG} avec des wFST

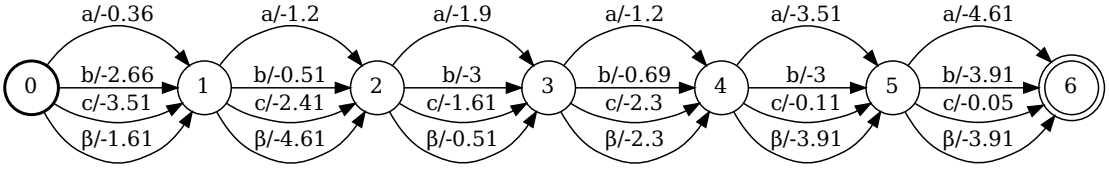
Nous commençons par la construction de ces trois wFST car certains des éléments présentés dans cette partie servent à la construction de G_{L2G}. Ces éléments sont :

- Les wFST A_{ASG} et A_{CTC} portant les scores acoustiques de CTC et d'ASG.
- Le wFST B modélisant le modèle de transition bi-gramme d'ASG.
- Le wFST Y modélisant une transcription.
- Les wFST MAP_{ASG} et MAP_{CTC} modélisant les règles d'alignement de CTC et d'ASG.

Construction des wFSA portant sur les scores acoustiques Nous construisons d'abord les wFSA A_{ASG} et A_{CTC}. La différence entre les deux porte sur l'utilisation du symbole **R1** ou β. Les chemins de ces wFSA sont tous les alignements possibles entre les T trames du modèle acoustique et de n'importe quelle transcription. Sur chaque arc, nous reportons le score acoustique $s_{\pi_t}(X)$ correspondant. Un exemple est donné Fig. 4.1. Le score d'un chemin de A_{ASG} ou de A_{CTC} est la somme des scores acoustiques de chaque arc de ce chemin.



(a) Le wFST A_{ASG}.



(b) Le wFST A_{CTC}.

FIGURE 4.1. – Exemple des wFSA A_{ASG} et A_{CTC} sur 6 trames. Ici, les transcriptions peuvent être composées des symboles **a**, **b**, **c** ou **R1**/β. **R1** est utilisé dans ASG pour modéliser une répétition de deux symboles consécutifs. β, qui est utilisé pour CTC, permet en plus de pouvoir modéliser un moment de silence. Chaque arc représente un symbole d'une trame et porte le score acoustique correspondant $s_{\pi_t}(X)$. Dans le cas de CTC ce sont des log-probabilités. Un chemin représente un alignement d'une transcription. Le score total d'un tel chemin est la somme des scores acoustiques de chacun de ses arcs.

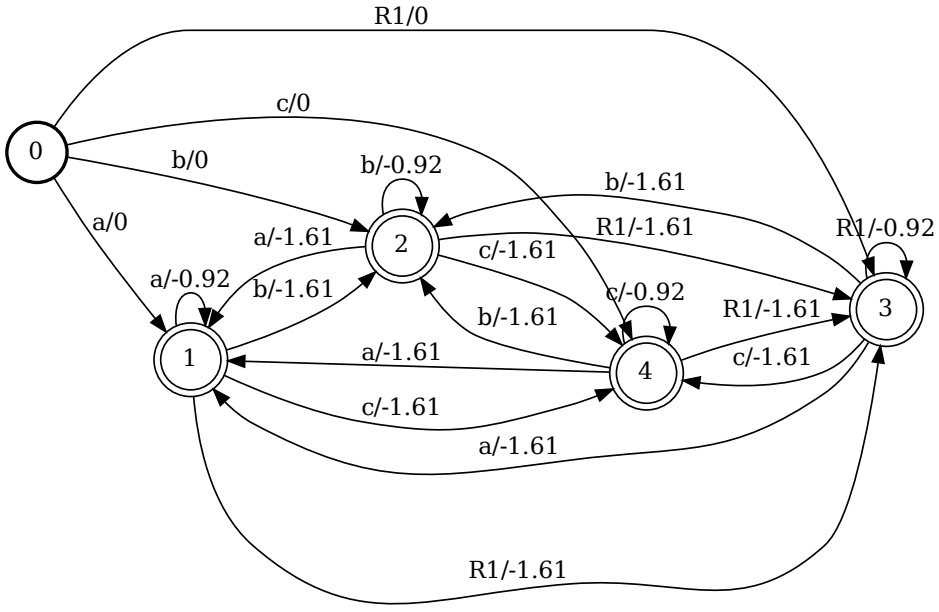


FIGURE 4.2. – Exemple d'un wFSA B. L'état **0** est l'état initial d'où partent les arcs pour rejoindre les états **1**, **2**, **3** et **4** en produisant les 4 symboles possibles **a**, **b**, **c** et **R1**. Chacun des états **1**, **2**, **3** et **4** ne peut être atteint qu'en produisant le même symbole. Par exemple, chacun des arcs entrant dans l'état **2** produit le symbole **b**. Les arcs partant de cet état portent les probabilités g_{π_{t-1},π_t} avec $\pi_{t-1} = \mathbf{b}$.

Construction du wFSA portant les probabilités du bi-gramme Le wFSA B porte les probabilités g_{π_{t-1},π_t} venant du modèle bi-gramme d'ASG. Un chemin de ce wFSA peut représenter n'importe quel alignement de longueur arbitraire. Les arcs représentent les transitions d'un symbole à un autre. La Fig. 4.2 donne un exemple.

Construction de G_{full} L'opération d'intersection des deux wFSA A et B permet d'obtenir un wFSA dont les chemins acceptés sont acceptés par A et par B. L'intersection de A et B permet donc d'obtenir un wFSA dont les chemins sont tous les alignements possibles entre les T trames du modèle acoustique et n'importe quelle transcription. De plus, le score d'un chemin de $A \circ B$ est obtenu en agrégeant les scores de A et de B de ce même chemin par l'opération $\otimes$ (c'est-à-dire l'addition dans le demi-anneau Log). Comme le score d'un chemin de A est la somme des scores acoustiques de chacun des T arcs de ce chemin, et que le score d'un chemin de B est la somme des probabilités du modèle bi-gramme, alors le score d'un chemin π de $A \circ B$ correspond au score $z_\pi(X)$ de l'équation (4.3). Nous obtenons alors :

$$G_{\text{full}} = A_{\text{ASG}} \circ B. \tag{4.10}$$

La Fig. 4.3 représente le graphe G_{full} résultant de l'opération d'intersection entre le wFSA A_{ASG} de la Fig. 4.1a et le wFSA B de la Fig. 4.2.

Construction des wFSA portant une transcription Nous voulons représenter les transcriptions présentes dans le jeu de données par un wFSA. Pour cela, nous créons un wFSA avec un seul chemin dont les symboles correspondent aux symboles de la transcription. Les poids sont fixés à 0. Ils n'auront pas d'influence sur les scores calculés par la suite.

Nous distinguons Y_{R1} et Y, les wFSA qui encodent une transcription avec ou sans utilisation du symbole **R1**. La Fig. 4.4 fournit un exemple d'une transcription encodée des deux manières différentes.

Construction du wFST modélisant l'alignement Le système d'alignement consiste à associer un symbole de la transcription avec une ou plusieurs trames (voir partie 2.3.3). Les poids des wFST présentés ici sont fixés à 0 et n'interviennent pas pour la suite.

ASG utilise le symbole **R1** pour modéliser la répétition de deux symboles identiques dans la transcription. Celui-ci n'est pas différent des autres symboles pour la gestion de l'alignement. La Fig. 4.5a présente le wFST MAP_{ASG} permettant d'associer une séquence d'un symbole présent plusieurs fois de suite à un seul symbole. Par exemple la séquence d'entrée [**a**, **a**, **a**, **b**, **R1**, **R1**] produit [**a**, **b**, **R1**].

Pour CTC, le symbole β ne correspond pas à un symbole de la transcription. La Fig. 4.5b présente le wFST MAP_{CTC} qui modélise cela. Par exemple, la séquence d'entrée [**a**, **a**, β, **b**, β, β, **b**] produit [**a**, **b**, **b**].

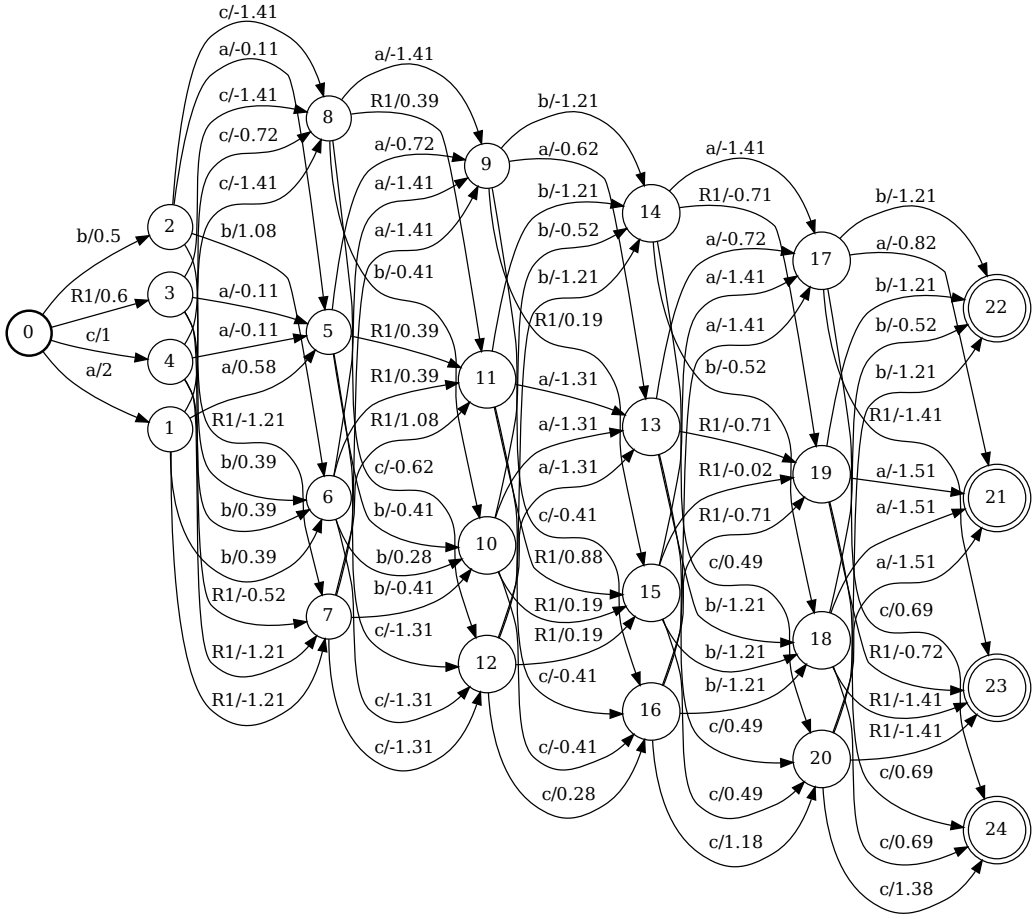
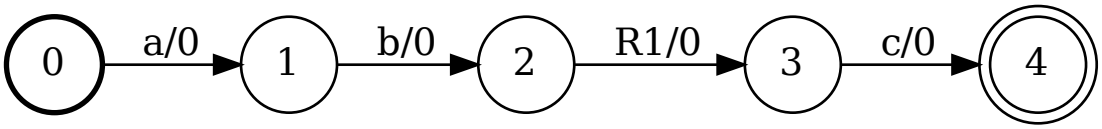
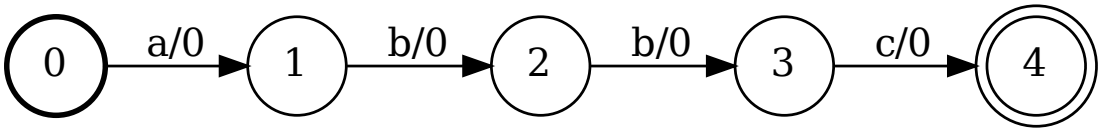


FIGURE 4.3. – Exemple d'un wFSA $G_{\text{full}} = A_{\text{ASG}} \circ B$. Les chemins représentent bien des alignements de longueur $T = 6$. L'algorithme de composition apparie un arc de A_{ASG} et un arc de B pour former un nouvel arc dont le score est la somme de ces arcs.

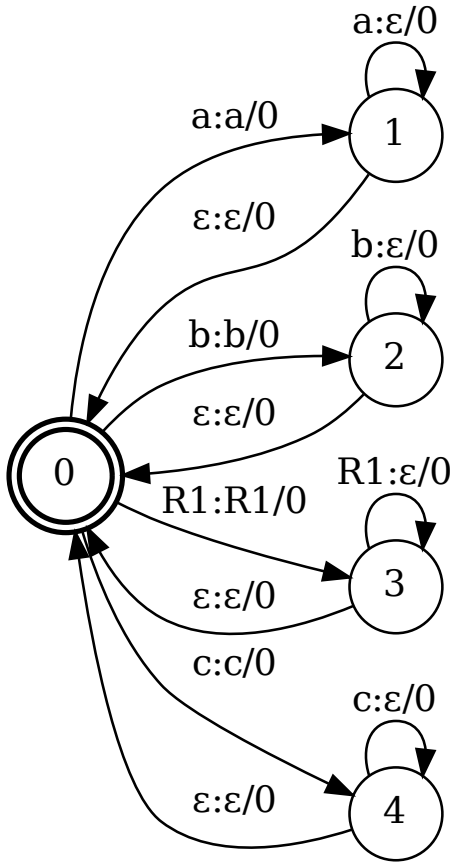


(a) Le wFSA Y_{R1}.

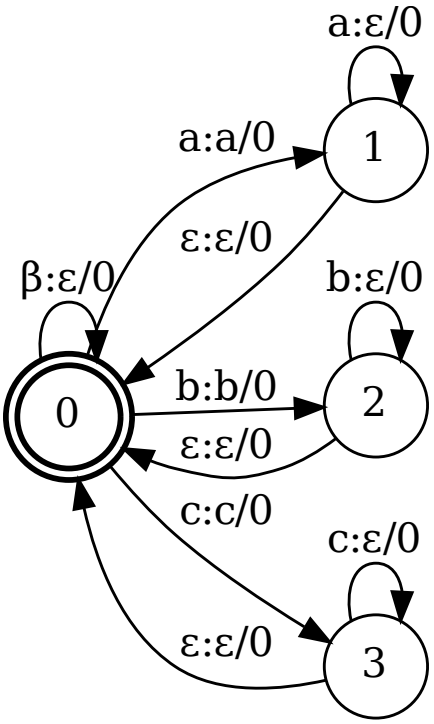


(b) Le wFSA Y.

FIGURE 4.4. – Exemple des wFSA Y_{R1} et Y. Le seul chemin accepté de Y_{R1} (a) est [**a**, **b**, **R1**, **c**] ce qui encode la transcription [**a**, **b**, **b**, **c**]. Le seul chemin accepté de Y (b) encode la même transcription.



(a) Le wFST MAP_{ASG} modélisant le système d'alignement d'ASG.

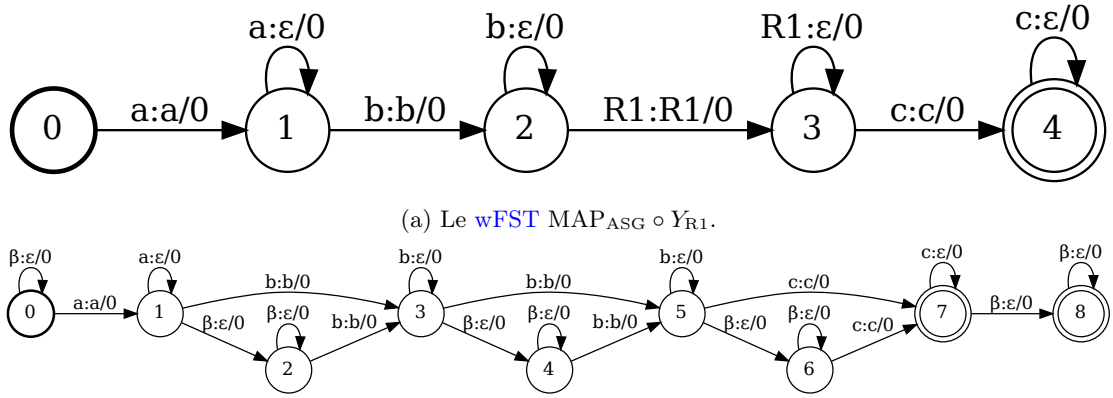


(b) Le wFST MAP_{CTC} modélisant le système d'alignement de CTC.

FIGURE 4.5. – Comparaison des deux wFST permettant de modéliser l'alignement pour ASG et CTC.

Construction des alignements d'une transcription Nous avons les wFST Y et Y_{R1} acceptant les symboles d'une transcription. Nous avons aussi les wFST MAP_{CTC} et MAP_{ASG} qui acceptent en entrée tous les alignements de n'importe quelle transcription en sortie. Les opérations de composition $\text{MAP}_{\text{CTC}} \circ Y$ et $\text{MAP}_{\text{ASG}} \circ Y_{\text{R1}}$ permettent donc d'obtenir des wFST acceptant tous les alignements de longueurs arbitraires d'une transcription Y ou Y_{R1}. La Fig. 4.6 présente les résultats de ces compositions.

Construction de G_{CTC} et G_{ASG} D'une part, nous avons A_{CTC}, un wFSA dont les chemins représentent tous les alignements sur T trames de n'importe quelle transcription. De plus, le score total d'un de ces chemins correspond bien au score de la fonction de coût CTC. D'autre part, nous avons $\text{MAP}_{\text{CTC}} \circ Y$, un wFST qui contient tous les alignements d'une transcription Y selon les règles de CTC. Ce wFST ne contient que des poids fixés à



(a) Le wFST $\mathrm{MAP}_{\mathrm{ASG}} \circ Y_{\mathrm{R1}}$.

(b) Le wFST $\mathrm{MAP}_{\mathrm{CTC}} \circ Y$.

FIGURE 4.6. – wFST qui représentent tous les alignements de la transcription [**a**, **b**, **b**, **c**] avec le système d'alignement de CTC et de ASG. Pour rappel, le symbole vide ϵ des wFST permet de modéliser une absence de symbole. On peut remarquer dans (b) que l'arc produisant le symbole **b** entre les états **3** et **5** n'est pas bon puisque le système d'alignement de CTC force normalement la production d'un symbole β entre deux symboles consécutifs. Pour éviter ce problème, notre implémentation construit directement $\mathrm{MAP}_{\mathrm{CTC}} \circ Y$ sans l'opération de composition.

0. La composition de ces deux graphes nous donne donc bien un wFST dont les chemins sont tous les alignements sur T trames d'une transcription Y et dont les scores sont ceux de CTC. Nous avons :

$$G_{\mathrm{CTC}} = A_{\mathrm{CTC}} \circ \mathrm{MAP}_{\mathrm{CTC}} \circ Y. \tag{4.11}$$

Nous faisons le même raisonnement pour G_{ASG}. Il faut en plus inclure les probabilités du modèle bi-gramme pour que le score d'un chemin corresponde bien au score ASG. Nous obtenons :

$$G_{\mathrm{ASG}} = A_{\mathrm{ASG}} \circ B \circ \mathrm{MAP}_{\mathrm{ASG}} \circ Y. \tag{4.12}$$

La Fig. 4.7 représente G_{CTC} et G_{ASG} construits à partir des exemples précédents. Nous pouvons maintenant appliquer la fonction Score(.) sur ces wFST pour obtenir les résultats d'ASG et CTC sur ces exemples :

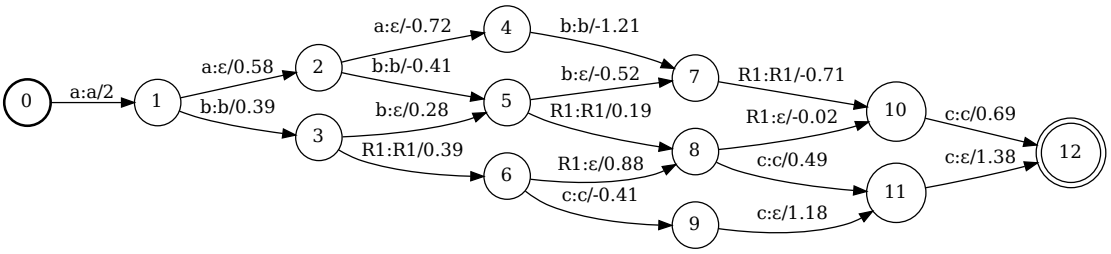
$$\mathrm{CTC}([\mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}]) = -\mathrm{Score}(G_{\mathrm{CTC}}) \approx 1,86 \tag{4.13}$$

$$\mathrm{ASG}([\mathbf{a}, \mathbf{b}, \mathbf{b}, \mathbf{c}]) = \mathrm{Score}(G_{\mathrm{full}}) - \mathrm{Score}(G_{\mathrm{ASG}}) \approx 8,86 - 6,59 \approx 2,27. \tag{4.14}$$

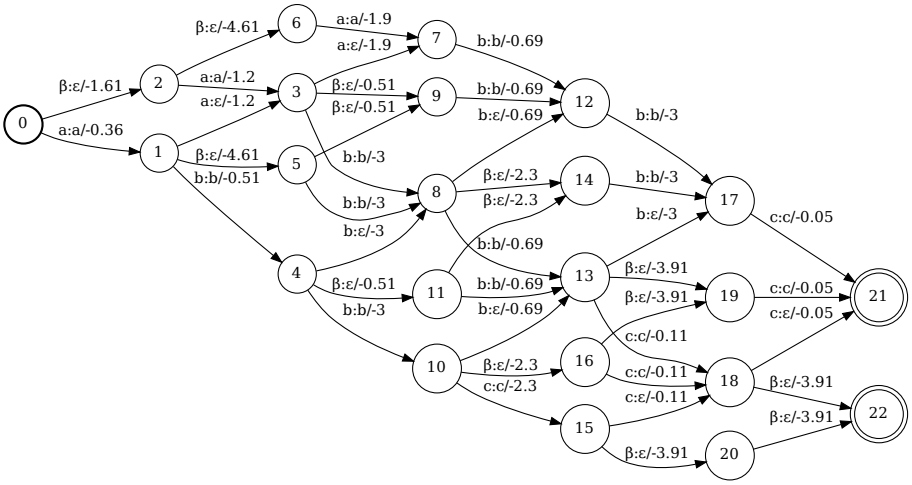
Les valeurs obtenues sont des opposés de log-probabilités. Pour CTC, cela correspond à une probabilité de 0,16 et pour ASG une probabilité de 0,10.

La librairie GTN [Hannun et al., 2020] que nous utilisons peut ensuite calculer automatiquement les dérivées partielles du score par rapport à n'importe quel wFST utilisé pour la construction de ce score (voir partie 2.2.4). Nous demandons de faire ce calcul par

rapport aux scores de A_{CTC}, A_{ASG} et B. Par exemple, nous représentons les gradients de A_{CTC} dans la Fig. 4.8. Nous pouvons alors utiliser ces gradients pour mettre à jour les poids du modèle acoustique qui a produit les scores en utilisant un optimiseur et la rétro-propagation (voir partie 2.3.4).



(a) Le wFST G_{ASG}.



(b) Le wFST G_{CTC}.

FIGURE 4.7. – Les wFST G_{ASG} et G_{CTC} obtenus après composition des différents éléments. Chaque chemin représente un alignement valide entre la transcription [**a**, **b**, **b**, **c**] et les 6 trames du modèle acoustique. Nous pouvons remarquer que la méthode ASG produit un graphe final G_{ASG} plus petit que G_{CTC}.

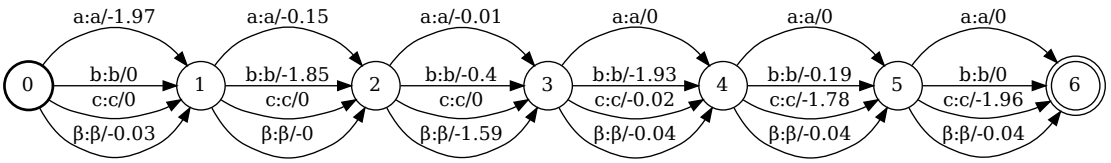


FIGURE 4.8. – Exemple de gradient sur CTC. Un gradient important indique qu'il faut effectuer une forte correction du modèle acoustique pour obtenir un meilleur score.

4.1.3. Construction de G_{L2G} avec des wFST

Construction du wFST modélisant le modèle de bruit de transcription G_{L2G} est un wFST qui accepte tous les alignements d'un ensemble de transcriptions Y^* obtenu à partir de la transcription bruitée $\tilde{Y}$ présente dans le jeu de données. Pour générer cet ensemble de transcriptions, nous utilisons le wFST N qui porte les probabilités du modèle de bruit de transcription. Pour pouvoir visualiser le résultat, nous présentons dans la Fig. 4.9 un modèle de bruit simple qui génère peu d'hypothèses Y^* à partir de $\tilde{Y}$. Dans la partie 4.3 nous présenterons des modèles de bruit réalistes.

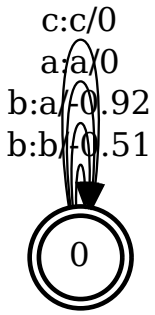


FIGURE 4.9. – Exemple de modèle de bruit de transcription N. Les poids affichés sont des log-probabilités. Ce modèle de bruit autorise seulement 4 transformations dont les probabilités sont $p(\tilde{y} = a|y^* = a) = 1$, $p(\tilde{y} = c|y^* = c) = 1$, $p(\tilde{y} = a|y^* = b) = 0.4$ et $p(\tilde{y} = b|y^* = b) = 0.6$.

Construction de l'ensemble des hypothèses Y^* avec le modèle de bruit Les modèles de bruit que nous utilisons n'utilisent pas le symbole **R1** car celui-ci ne représente pas un symbole en particulier. Nous encodons donc la transcription $\tilde{Y}$ dans un wFSA sans ce symbole et nous générons les hypothèses Y^* en calculant $N \circ \tilde{Y}$. La Fig. 4.10 présente le résultat de ce calcul sur notre exemple. Le score total d'un des chemins vaut $\log p(\tilde{Y}|Y^*) = \sum_i \log p(\tilde{y}_i|y_i^*)$.

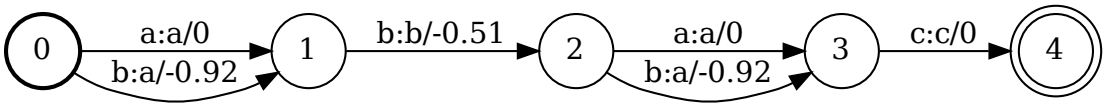


FIGURE 4.10. – Exemple d'utilisation du modèle de bruit de transcription avec $N \circ \tilde{Y}$. Les chemins de ce wFST sont toutes les hypothèses Y^* obtenues à partir de $\tilde{Y}$.

Construction du wFST modélisant le symbole R1 Lors de la génération des hypothèses Y^*, nous n'avons pas utilisé le symbole **R1** pour modéliser la répétition de deux

symboles. Nous utilisons le wFST R pour introduire ce symbole dans les Y^* lorsque celui-ci est nécessaire.

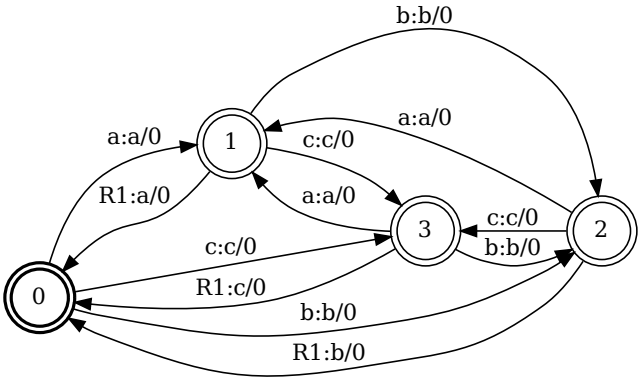


FIGURE 4.11. – Modélisation du symbole **R1** avec le wFST R. Les états **1**, **2** et **3** indiquent qu'un symbole **a**, **b** ou **c** est présent en sortie. Si le même symbole est ensuite produit, alors on transite vers l'état neutre **0** en produisant **R1**.

Utilisation du wFST R La Fig. 4.12 présente le résultat du calcul $R \circ N \circ \tilde{Y}$ permettant d'encoder les hypothèses Y^* avec le symbole **R1**.

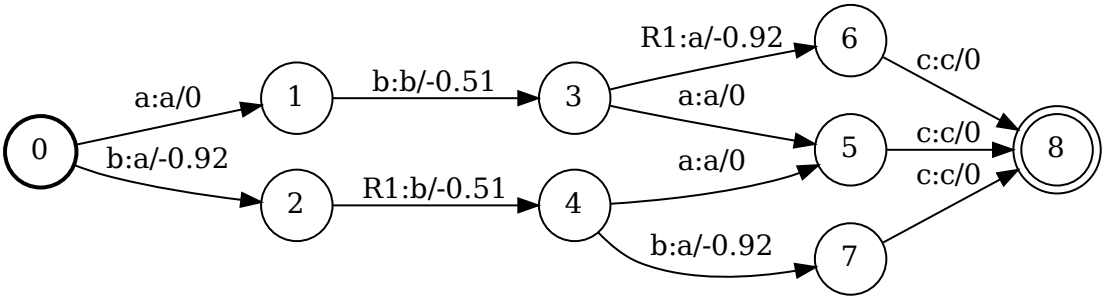


FIGURE 4.12. – Exemple d'utilisation du wFST R avec $R \circ N \circ \tilde{Y}$. Les 4 hypothèses Y^* de la Fig. 4.10 obtenues par $N \circ \tilde{Y}$ sont maintenant encodées avec le symbole **R1**.

Obtention de tous les alignements possibles des hypothèses $Y*$ Comme pour la construction de G_{ASG}, nous utilisons le wFST MAP_{ASG} pour générer tous les alignements possibles entre les hypothèses Y^* et un nombre arbitraire de trames selon les règles d'ASG. La Fig. 4.13 présente le résultat du calcul de $\text{MAP}_{\text{ASG}} \circ R \circ N \circ \tilde{Y}$. Un chemin de ce wFST représente tous les alignements d'une hypothèse Y^*. Le score total d'un tel chemin est $\log p(\tilde{Y}|Y^*)$.

Construction de G_{L2G} D'une part, nous avons le wFST $A_{\text{ASG}} \circ B$ dont les chemins représentent tous les alignements sur T trames de n'importe quelle transcription. Le

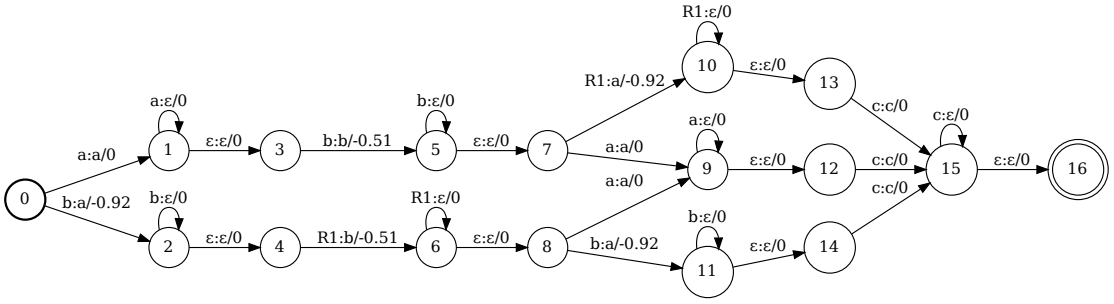


FIGURE 4.13. – Alignement des hypothèses Y^* sur un nombre arbitraire de trames selon les règles d'ASG par le calcul de $\text{MAP}_{\text{ASG}} \circ R \circ N \circ \tilde{Y}$.

score total d'un de ces chemins correspond au score de la fonction de coût ASG de l'équation (4.3). D'autre part, nous avons $\text{MAP}_{\text{ASG}} \circ R \circ N \circ \tilde{Y}$, un wFST qui contient tous les alignements de toutes les hypothèses Y^* générées à partir de la transcription source $\tilde{Y}$ et du modèle de bruit de transcription N. Le score total d'un de ses chemins est $\log p(\tilde{Y}|Y^*)$. La composition de ces deux graphes nous donne donc bien un wFST dont les chemins sont ceux du graphe Lead2Gold. Le score total d'un de ces chemins est aussi celui de Lead2Gold (voir équation (4.4)). Nous notons $G_{\text{L2G}}^{\text{ASG}}$ le graphe de Lead2Gold obtenu avec les règles d'ASG. Pour résumer nous avons :

$$L2G_{\text{ASG}} = \text{Score}(G_{\text{full}}(T)) - \text{Score}(G_{\text{L2G}}^{\text{ASG}}(\tilde{Y}, T)) \tag{4.15}$$

$$G_{\text{L2G}}^{\text{ASG}}(\tilde{Y}, T) = A_{\text{ASG}} \circ B \circ \text{MAP}_{\text{ASG}} \circ R \circ N \circ \tilde{Y} \tag{4.16}$$

$$G_{\text{full}}(T) = A_{\text{ASG}} \circ B. \tag{4.17}$$

La Fig. 4.14 représente le résultat de $G_{\text{L2G}}^{\text{ASG}}(\tilde{Y}, T)$ sur notre exemple. Nous pouvons calculer la valeur de $L2G_{\text{ASG}}$ et nous obtenons :

$$L2G_{\text{ASG}} \approx 8,86 - 5,48 \approx 3,38. \tag{4.18}$$

Cette valeur correspond à une probabilité de 0,034. GTN nous permet de calculer automatiquement les gradients de $L2G_{\text{ASG}}$ par rapport à A_{ASG}. La Fig. 4.15 représente la valeur des gradients obtenus sur ce wFST.

Une version de Lead2Gold avec les règles de CTC Pour l'algorithme Lead2Gold nous avions choisi une formulation qui s'inspire de la fonction de coût ASG. La formulation à base de wFST est maintenant modulable et il est immédiat d'obtenir une formulation de Lead2Gold avec les règles de CTC. Nous obtenons :

$$L2G_{\text{CTC}} = -\text{Score}(G_{\text{L2G}}^{\text{CTC}}(\tilde{Y}, T)) \tag{4.19}$$

$$G_{\text{L2G}}^{\text{CTC}}(\tilde{Y}, T) = A_{\text{CTC}} \circ \text{MAP}_{\text{CTC}} \circ N \circ \tilde{Y}. \tag{4.20}$$

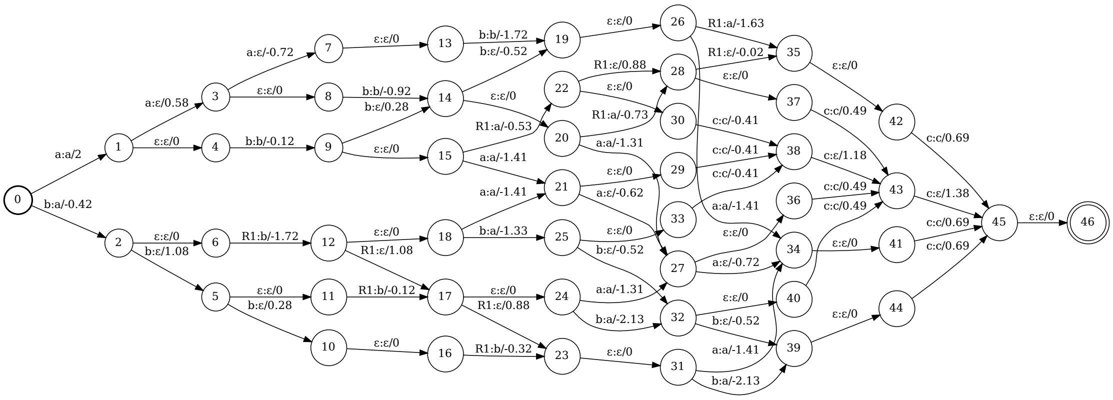


FIGURE 4.14. – Exemple de wFST $G_{\text{L2G}}^{\text{ASG}}$. Les compositions successives nous ont permis de recréer le graphe de Lead2Gold sans programmation explicite.

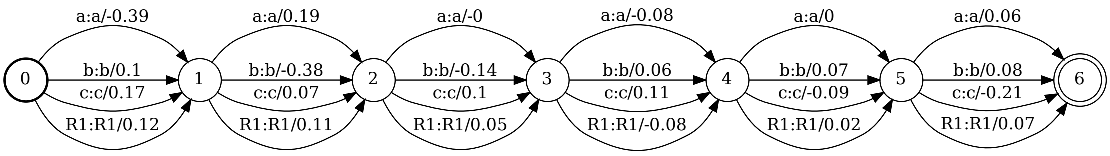


FIGURE 4.15. – Calcul des gradients de $L2G_{\text{ASG}}$. Les gradients sont répartis sur plus de chemins que la version sans modèle de bruit car nous prenons en compte un ensemble d'hypothèses Y^*.

La Fig. 4.16 représente le résultat de $G_{\text{L2G}}^{\text{CTC}}(\tilde{Y}, T)$ sur notre exemple. Nous pouvons calculer la valeur de $L2G_{\text{CTC}}$ et nous obtenons :

$$L2G_{\text{CTC}} \approx 2,21. \tag{4.21}$$

Cette valeur correspond à une probabilité de 0,11.

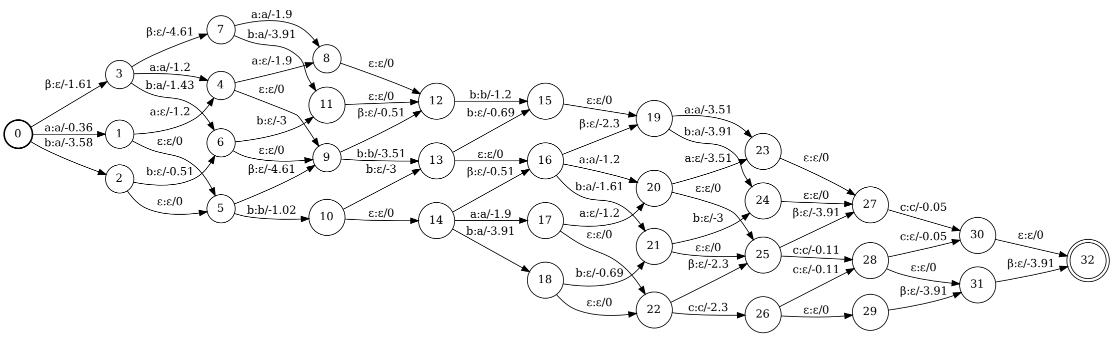


FIGURE 4.16. – Exemple de wFST $G_{\text{L2G}}^{\text{CTC}}$. La formulation modulaire de Lead2Gold nous permet d'obtenir immédiatement une version avec les règles de CTC.

4.2. Nouveaux modules

4.2.1. Utilisation de symboles différents pour le modèle de bruit

Les modèles acoustiques de ce manuscrit utilisent les lettres comme symboles. Avec Lead2Gold nous avons ajouté un modèle de bruit de transcription qui utilise également les lettres comme symboles. C'est une hypothèse forte qui n'est pas forcément réaliste. Avec l'utilisation des wFST, nous pouvons changer facilement les symboles du modèle de bruit tout en gardant les lettres pour le modèle acoustique. Nous notons $\mathcal{D}_A$ et $\mathcal{D}_N$ les dictionnaires de symboles pour le modèle acoustique et le modèle de bruit. Nous gardons les lettres pour $\mathcal{D}_A$.

Modélisation par un wFST Nous utilisons le wFST $D_{A\leftrightarrow N}$ pour passer du système de symbole de $\mathcal{D}_A$ à celui de $\mathcal{D}_N$. La Fig. 4.17 propose un exemple d'un tel wFST.

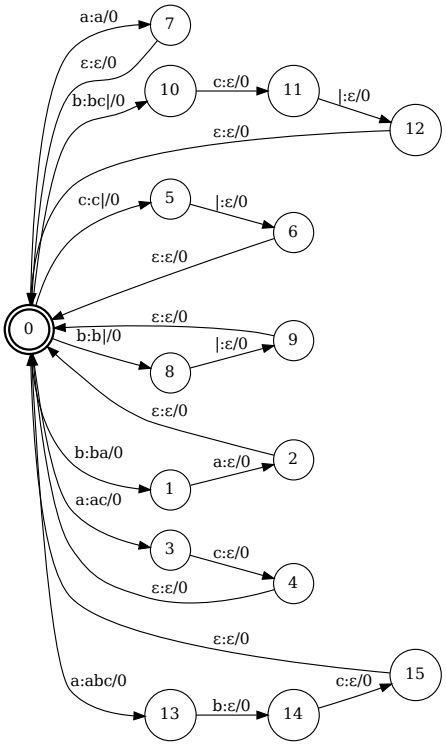


FIGURE 4.17. – Exemple de wFST $D_{A\leftrightarrow N}$. Ici $\mathcal{D}_N = \{\mathbf{a}, \mathbf{c}|, \mathbf{b}|, \mathbf{bc}|, \mathbf{abc}, \mathbf{ba}, \mathbf{ac}\}$ et $\mathcal{D}_A = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, |\}$ [1]. Il y a plusieurs manières de représenter un tel wFST mais nous remarquons que cette structure permet d'effectuer des opérations de composition plus rapides par la suite.

Construction de l'ensemble $\mathcal{D}_N$ Nous utilisons des morceaux de mots (*word pieces*) comme nouveaux symboles. Nous utilisons l'implémentation « SentencePiece » de l'al-

gorithme unigramme de Kudo [2018] pour générer les morceaux de mots. C'est un algorithme qui ne demande que du texte pour son utilisation. Il faut aussi déterminer à l'avance le nombre de morceaux de mots que l'on veut obtenir et l'algorithme trouve ceux qui minimisent le nombre de symboles à utiliser pour encoder le texte. Dans la partie 4.3, nous déterminerons quelle est la bonne taille pour $\mathcal{D}_N$. Il est à noter que l'encodage d'un texte peut ne pas être unique avec des morceaux de mots.

4.2.2. Restrictions des hypothèses Y^* avec un lexique

Il peut y avoir beaucoup d'hypothèses Y^* générées lors de la composition du modèle de bruit par la transcription source $\tilde{Y}$. Cela peut en pratique rendre le calcul de la fonction de coût très lent. Nous proposons de limiter cet ensemble d'hypothèses en utilisant un lexique. Ainsi nous ne gardons que celles dont les symboles forment des mots existants. Les symboles du lexique sont ceux de $\mathcal{D}_N$. Pour construire le wFSA modélisant le lexique nous procédons en deux étapes afin obtenir la représentation idéale. Nous voulons une taille de wFSA la plus petite possible car cela impacte grandement la vitesse d'exécution des futures opérations de composition.

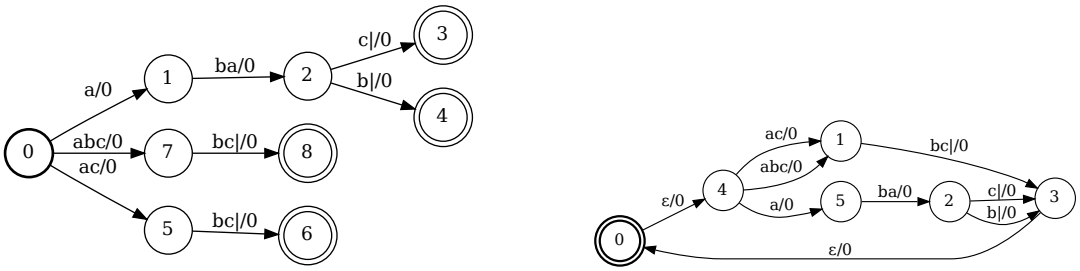
La première étape est de représenter chaque mot du lexique par un chemin dans un wFSA. L'encodage du mot est fourni par le modèle unigramme qui génère l'ensemble $\mathcal{D}_N$. Nous n'autorisons qu'un seul encodage par mot. Le modèle unigramme indique celui qui est le meilleur. À cette étape nous pouvons déjà mettre en commun les chemins dont les mots commencent de la même façon. Ce wFSA est aussi appelé arbre préfixe. La Fig. 4.18a présente le résultat de cette étape. Nous obtenons ainsi un graphe acyclique et déterministe.

La seconde étape est l'utilisation d'un algorithme de minimisation pour les wFSA acycliques, déterministes et non-pondérés. Dans cette configuration, il existe un algorithme de minimisation en temps linéaire [Revuz, 1992]. Nous avons implémenté nous-mêmes cet algorithme car il n'est pas présent dans GTN. Plus précisément nous utilisons l'algorithme *bottom-up* [Almeida and Zeitoun, 2008]. Pour pouvoir modéliser une suite de mots, nous ajoutons une fermeture de Kleene au wFSA minimisé. La minimisation a permis de mettre en commun la fin des mots du lexique (même suffixe). La Fig. 4.18b présente le résultat final. Nous notons L le wFSA qui modélise un tel lexique.

4.2.3. Intégration des modules

La transcription source $\tilde{Y}$ est tout d'abord encodée avec les symboles de $\mathcal{D}_N$. Ensuite, l'opération $N \circ \tilde{Y}$ permet d'obtenir les transcriptions hypothèses Y^* qui sont également encodées avec les symboles de $\mathcal{D}_N$. Pour restreindre ces hypothèses à des mots valides nous composons le résultat avec L, puis pour passer au système d'encodage par lettres

1. Le symbole « | » représente l'espace entre les mots



(a) Réprésentation du lexique par un arbre préfixe.

(b) Représentation optimale du lexique. Les préfixes et les suffixes sont partagés. La fermeture de Kleene est utilisée pour modéliser une suite de mots.

FIGURE 4.18. – Les deux étapes de construction du lexique L. Pour cet exemple, nous utilisons les mots **acbc|**, **abcbc|**, **abab|**, **abac|** encodés avec les symboles de la Fig. 4.17.

nous composons avec $D_{A\leftrightarrow N}$. Les hypothèses Y^* contraintes par un lexique et encodées par des lettres sont donc obtenues par $D_{A\leftrightarrow N} \circ L \circ N \circ \tilde{Y}$.

De nombreuses possibilités de fonction de coût La formulation de notre fonction de coût est très modulaire. En plus du lexique et du changement de symboles, nous pouvons mélanger les caractéristiques des deux fonctions de coût ASG et CTC. Voici les options utilisables :

- Normalisation des scores acoustiques au niveau de la trame (avec A_{CTC} et un seul terme à la fonction de coût) ou bien au niveau de la séquence (avec A_{ASG} et deux termes à la fonction de coût). Pour la suite nous noterons simplement A le wFST portant les scores acoustiques, qu'il soit normalisé ou non.
- Utilisation du modèle bi-gramme (avec B) ou non.
- Utilisation du symbole **R1** de répétition (avec R) ou non.
- Utilisation du symbole β (avec MAP_{CTC}) ou non (avec MAP_{ASG}). Pour la suite nous noterons simplement MAP le wFST qui modélise l'alignement dans les deux cas.
- Utilisation d'un modèle de bruit (avec N) ou non.
- Utilisation d'un autre système de symboles que celui des lettres pour le modèle de bruit (avec $D_{A\leftrightarrow N}$) ou non.
- Utilisation d'un lexique (avec L) ou non.

Dans le cas de la normalisation au niveau de la séquence, la fonction de coût s'exprime par

$$\text{Coût} = \text{Score(DEN)} - \text{Score(NUM)}. \tag{4.22}$$

Dans le cas de la normalisation par trame, nous avons un seul terme :

$$\text{Coût} = -\text{Score(NUM)}. \tag{4.23}$$

DEN (pour dénominateur) est un wFST qui s'exprime par

$$\text{DEN} = A \circ B, \tag{4.24}$$

et NUM (pour numérateur) est un wFST qui s'exprime par

$$\text{NUM} = A \circ \{B\} \circ \text{MAP} \circ \{R\} \circ \{D_{A \leftrightarrow N}\} \circ \{L\} \circ \{N\} \circ Y. \tag{4.25}$$

Les termes entre accolades sont les modules optionnels que nous pouvons activer ou non.

Calcul des opérations de composition La fonction de coût doit être calculée pour chaque exemple du jeu de données. Nous pouvons mettre en commun le wFST central que nous notons

$$T = \text{MAP} \circ \{R\} \circ \{D_{A \leftrightarrow N}\} \circ \{L\} \circ \{N\}. \tag{4.26}$$

Cela permet d'écrire

$$\text{NUM} = A \circ \{B\} \circ T \circ Y. \tag{4.27}$$

Le wFST T ne change pas au cours du temps. Nous initialisons la fonction de coût en calculant ce wFST une fois et nous stockons le résultat.

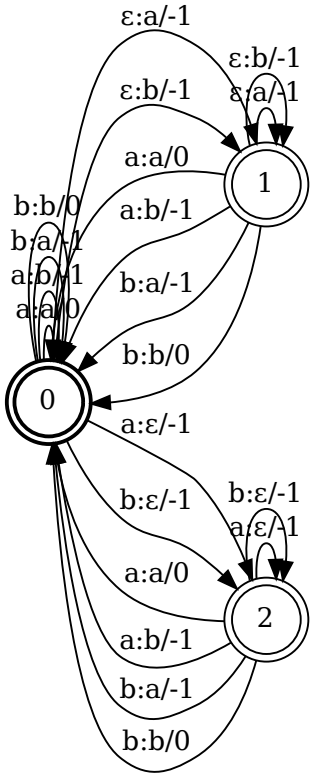
L'ordre des opérations de composition a une grande importance. Le wFST T est très grand, nous voulons éviter qu'une nouvelle opération de composition génère un nouveau wFST encore plus grand. La première opération à faire est donc $T \circ Y$ pour faire la suite des calculs avec une transcription en particulier et non pas dans un cas général. Nous composons ensuite par B puis enfin par A. Le wFST $A \circ B$ est très grand donc nous évitons de l'utiliser pour une nouvelle composition.

4.3. Le modèle de bruit de transcription N

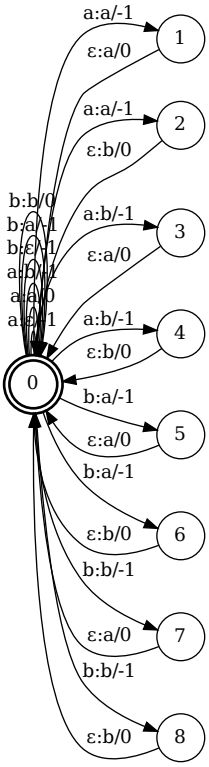
4.3.1. Structures du modèle de bruit N

Quel que soit le système de symboles $\mathcal{D}_N$ utilisé pour le modèle de bruit, nous devons choisir sa structure. Dans la littérature, il existe une méthode spectrale permettant d'inférer la structure et les poids d'un wFST inconnu à partir d'un jeu de données textuelles (voir partie 2.2.7). Nous préférons pouvoir contrôler la structure du wFST N car celle-ci a une grande influence sur l'efficacité des opérations de composition. Nous proposons différentes structures dans la Fig. 4.19. Pour l'estimation des poids, nous proposons notre propre méthode de descente de gradient en utilisant la nature différentiable des wFST.

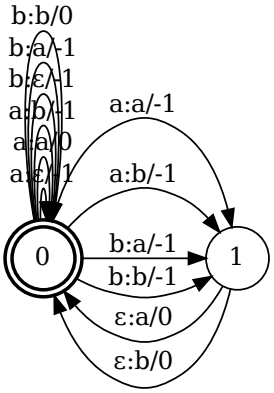
Le wFST N_1 de la Fig. 4.19a modélise les substitutions sur l'état **0**. N_1 modélise aussi les insertions avec l'état **1** et les délétions avec l'état **2**. Cette structure interdit qu'une délétion puisse suivre une insertion (ou l'inverse) car il faut revenir par l'état **0** avec une transformation de substitution. Cette propriété est importante car sinon, nous pourrions effectuer un nombre de transformations infini pour passer d'une transcription à une autre.



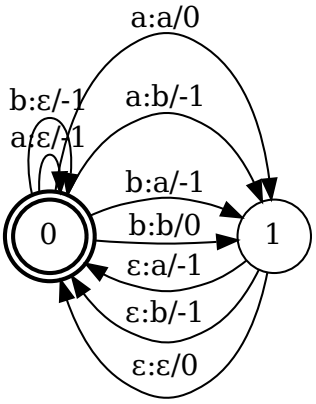
(a) Le wFST N_1.



(b) Le wFST N_2.



(c) Le wFST N_3.



(d) Le wFST N_4.

FIGURE 4.19. – Les différentes structures de wFST N que nous avons testées. Le nombre d'états et d'arcs influence la vitesse de calcul des futures opérations de composition. Nous représentons ici des modèles de bruit avec seulement deux symboles **a** et **b**. Les poids affichés sont les valeurs initiales que nous choisissons avant l'apprentissage.

Ce comportement est à interdire car cela génère des cycles dans le wFST final NUM. La fonction Score(.) ne fonctionne pas dans un tel cas. Nous pouvons tout de même effectuer plusieurs insertions ou plusieurs délétions de suite avec cette structure. Avec M le nombre de symboles de $\mathcal{D}_N$, nous comptons que cette structure a 3 états et $3 \times M^2 + 4 \times M$ arcs.

Le wFST N_2 de la Fig. 4.19b permet une modélisation plus fine des insertions. Celui-ci permet de calculer $p(\tilde{y}_1\tilde{y}_2|y_3^*\epsilon)$ avec les états **1** à **8**. L'état **0** porte les opérations de substitutions et de délétions. Cette configuration génère des grands wFST. Nous comptons en effet $M^3 + 1$ états et $2 \times M^3 + M^2 + M$ arcs.

Le wFST N_3 de la Fig. 4.19c modélise les insertions par $p(\tilde{y}_1\tilde{y}_2|y_3^*\epsilon) = p(\tilde{y}_1|y_3^*)p(\tilde{y}_2|\epsilon)$. Il ressemble à N_2 à l'exception des états **1** à **8** qui sont fusionnés. Nous comptons 2 états et $2 \times M^2 + 2 \times M$ arcs.

Le wFST N_4 de la Fig. 4.19d fusionne les opérations de substitution de N_3 en les supprimant de l'état **0**. À partir de l'état **1**, nous avons la possibilité de ne pas faire d'insertion avec la transition $p(\epsilon|\epsilon)$. Nous comptons 2 états et $M^2 + 2 \times M + 1 = (M+1)^2$ arcs. Ce modèle de bruit est en fait celui de Lead2Gold. Nous avons en effet le même nombre de poids et les insertions sont modélisées de la même manière : pour effectuer une opération d'insertion, il faut que la précédente soit une substitution.

4.3.2. Apprentissage des poids du modèle de bruit N

Nous avons besoin d'un jeu de données contenant des paires $(\tilde{Y}, Y^*)$ de transcriptions avec et sans bruit pour apprendre les poids du wFST N. Nous proposons un apprentissage par descente de gradient pour régler ces poids. Pour cela, nous construisons la fonction de coût :

$$\text{Coût}(\tilde{Y}, Y^*) = -\log p(\tilde{Y}|Y^*). \tag{4.28}$$

Nous cherchons à minimiser cette quantité. Le wFST N modélise la quantité $\log p(\tilde{Y}|Y^*)$. L'opération $N \circ \tilde{Y}$ permet d'obtenir un ensemble d'hypothèses Y^* à partir de $\tilde{Y}$ et du modèle de bruit N. Nous filtrons cet ensemble en le composant par le wFSA qui représente le vrai Y^* du jeu de données. Nous avons alors :

$$\text{Coût}(\tilde{Y}, Y^*) = -\text{Score}(Y^* \circ N \circ \tilde{Y}). \tag{4.29}$$

Considérons ici que le symbole ϵ des wFST est un symbole comme les autres. Les poids de N représentent les quantités $\log p(\tilde{y}_i|y_j^*)$ entre deux états. Mais un apprentissage par descente de gradient ne permet pas de forcer le modèle à représenter une distribution de probabilité. C'est pourquoi nous introduisons l'opération logSoftMax(.) pour un wFST qui normalise les poids. Nous posons

$$N = \text{logSoftMax}(N_{\text{score}}), \tag{4.30}$$

avec N_{score} un wFST qui a la même structure que N et dont les poids sont des scores non-normalisés que l'on veut apprendre par descente de gradient.

L'opération logSoftMax(.) Cette opération permet de normaliser les poids pour que chaque état d'un wFST ait des arcs sortants avec le même symbole d'entrée qui définit une distribution de probabilité. Soit $W_{\rightarrow e}(y^*)$ l'ensemble des poids d'un wFST qui proviennent d'arcs sortants de e et ayant le symbole d'entrée y^*. L'opération logSoftMax(.) pour chaque score $s(\tilde{y}_i|y_j^*)$ d'un état e est définie par :

$$\text{logSoftMax}(s(\tilde{y}_i|y_j^*)) = \log \frac{\exp s(\tilde{y}_i|y_j^*)}{\sum_{w \in W_{\rightarrow e}(y_j^*)} \exp w} \tag{4.31}$$

$$= s(\tilde{y}_i|y_j^*) - \underset{w \in W_{\rightarrow e}(y_j^*)}{\text{logadd}} w \tag{4.32}$$

$$= \log p(\tilde{y}_i|y_j^*). \tag{4.33}$$

Nous définissons aussi la matrice Jacobienne de cette opération pour que la méthode de rétro-propagation puisse propager les gradients jusqu'aux poids de N_{score}.

Élagage du modèle de bruit Après l'apprentissage, une grande partie des arcs du modèle de bruit porte des probabilités faibles. Pour limiter le nombre d'hypothèses Y^* générées par le modèle de bruit N, nous supprimons les arcs de N_{score} modélisant les transitions les moins probables. Nous proposons ainsi une méthode spéciale d'élagage pour N_{score}.

Pour chaque état de N_{score}, nous avons des arcs sortants dont les poids forment des ensembles de distributions de probabilités après normalisation. Au lieu de supprimer simplement les arcs portant les plus faibles probabilités, nous gardons ceux qui permettent d'atteindre des masses de probabilités supérieures à un seuil p_{seuil}. Nous obtenons ainsi un wFST N_{score} élagué que nous pouvons normaliser avec logSoftMax(.).

4.4. Expériences

Cette partie présente les expériences que nous avons menées avec la formulation modulaire de la fonction coût. Pour l'algorithme Lead2Gold, nous avons un contrôle fin de la complexité de calcul grâce à l'approximation par une recherche en faisceau. Il est cependant difficile d'anticiper cette complexité avec notre formulation utilisant des wFST. En pratique, nous avons des problèmes non-résolus d'efficacité de calcul lorsque le modèle de bruit de transcription N est activé. Nous avons pourtant des outils permettant de limiter les tailles des wFST générés avec l'utilisation du lexique et de l'élagage du modèle de bruit. Nous allons voir que cela ne suffit pas et nous envisageons des solutions.

La partie 4.4.1 présente les jeux de données que l'on utilise et l'architecture du modèle acoustique. Dans la partie 4.4.2, nous testons les fonctionnalités de la fonction de coût sans le modèle de bruit. Dans la partie 4.4.3, nous menons des expériences d'apprentissage de modèles de bruit. Enfin dans la partie 4.4.4, nous mesurons les performances avec le modèle de bruit.

4.4.1. Mise en place

Jeu de données Dans cette partie, nous voulons utiliser des jeux de données bruitées plus réalistes. Pour cela, nous utilisons un modèle acoustique dont l'apprentissage a été arrêté avant la convergence pour générer les transcriptions erronées. Cette fois-ci, nous faisons une étape de recherche en faisceau avec un lexique pour ne générer que des mots qui existent.

Pour ce modèle acoustique, nous utilisons la même architecture *Gated ConvNet* que la partie 3.4.1 mais nous nous limitons à 15 couches de convolution 1D *Gated*. Nous obtenons un petit modèle qui est composé de 4,3 millions de paramètres. Pour l'apprentissage, nous utilisons la fonction de coût ASG avec un pas d'apprentissage de 5,6 pour les paramètres du modèle acoustique et de 0,004 pour le modèle bi-gramme. La taille du *batch* est de 128 phrases réparties sur 8 GPU. Les gradients sont tronqués à 0,05 et nous utilisons l'optimiseur SGD. Ces paramètres d'apprentissage sont fournis par Flashlight. Pour les données, nous séparons l'ensemble *train-clean-360* en 36 sous-ensembles contenant environ 10 heures de parole. Un de ces sous-ensembles est utilisé pour l'apprentissage de ce modèle acoustique.

Nous enregistrons l'état du modèle acoustique aux itérations 1 200, 1 400 et 4 000, puis nous appliquons une recherche en faisceau avec le même lexique et le même modèle de langage que dans la partie 3.3.4. Nous cherchons les meilleurs paramètres du décodeur pour chaque modèle acoustique en évaluant le WER obtenu sur *dev-clean*. Enfin, nous décodons entièrement le jeu de données *train-clean-100* pour obtenir trois versions de jeux de données bruitées. Le Tab. 4.1 présente le WER obtenu pour les trois configurations. Au vu des résultats sur *train-clean-100*, nous appelons les trois jeux de données bruitées D65, D39 et D21. Le Tab. 4.2 présente un exemple de transcription pour chacune des configurations. Nous décodons également 2 fois 10 heures non utilisées de *train-clean-360* avec le modèle acoustique de l'itération 4 000 pour former $\text{D21}_{\text{N}_{\text{train}}}$ et $\text{D21}_{\text{N}_{\text{dev}}}$ qui seront utilisés pour l'apprentissage du modèle de bruit.

Apprentissage	Test dev-clean	Test avec recherche en faisceau dev-clean	train-clean-100	Paramètres du décodeur
Itération 1 200	80, 4	66, 5	65, 2	2 500 / 25 / 2 / 1 / 0
Itération 1 400	61, 7	41, 0	39, 3	2 500 / 25 / 2 / 2 / 0
Itération 4 000	38, 5	22.7	21, 2	2 500 / 50 / 3 / 2 / -1

Tableau 4.1. – WER obtenu avec les trois états du modèle acoustique. Nous reportons les résultats sur *dev-clean* avec et sans décodage. Nous reportons aussi les résultats sur *train-clean-100* avec décodage. Ces transcriptions décodées forment les nouveaux jeux de données bruitées D65, D39 et D21. La dernière colonne indique les paramètres du décodeur utilisés avec dans l'ordre : le nombre d'hypothèses à explorer, le score minimal d'une hypothèse, le poids attribué au modèle de langage, la pénalité d'insertion d'un mot et la pénalité d'insertion d'un silence.

Données	Transcription
Référence	*the well known nerve specialist in consultation with the oculist and the local practitioner in charge of the case there is a feeling of wide spread regret and sympathy in those social and artistic circles where mister dalmain was so well known and so deservedly popular*
D65	*the well known* ***mere*** *specialist in consultation with the* ***apples*** *and the* ***like but*** *in* ***art*** *the* ***pace*** *there* ***was*** *a* ***thing as brave great*** *and* ***some of the indus sunder this acute or*** *mister* ***to men with*** *so* ***non*** *and so deservedly popular*
D39	*the well known* ***mere*** *specialist in consultation with the oculist and the local practitioner in charge of the case there is a feeling a* ***fine regret*** *and* ***sent the and*** *those* ***so shall enter icicles were*** *mister* ***doman*** *was so well known and so deservedly popular*
D21	*the well known nerve specialist in consultation with the oculist and the local practitioner in charge of the case there is a feeling of* ***widespread*** *regret and sympathy and those* ***so shall in*** *artistic circles where mister* ***dolman*** *was so well known and so deservedly popular*

TABLEAU 4.2. – Exemple de transcription obtenue après décodage. Nous indiquons la transcription de référence ainsi que les versions obtenues à partir des trois configurations. En gras, nous indiquons ce qui est faux dans les transcriptions décodées.

Recherche d'un modèle acoustique performant Pour tester les fonctionnalités de la fonction de coût, nous utilisons un modèle acoustique plus performant que dans la partie 3. En 2020, l'architecture Transformer [Gulati et al., 2020, Hsu et al., 2021, Synnaeve et al., 2019] est une de celles qui obtiennent les meilleures WER et nous utilisons celle-ci dans cette partie. Nous menons une expérience de recherche d'architecture pour déterminer le nombre et la taille des couches pour le jeu de données *train-clean-100*.

Comme présenté dans la partie 2.3.2, cette architecture utilise des couches de convolution 1D *Gated* comme premières couches. Nous fixons le nombre de couches de ce type à 6 comme suggéré par Flashlight avec les couches 1 et 3 qui utilisent un *stride* de 2. Nous fixons les tailles de noyaux à 3 et utilisons un *padding*. Le *dropout* est fixé à 0,3. La dimension de sortie des couches de convolution est réglée à 1 014 à l'exception de la dernière qui s'adapte à la dimension d'entrée des couches Transformer.

Pour les couches Transformer, nous fixons le *dropout* à 0,6. Nous faisons varier la taille des couches : la taille de chaque tête d'auto-attention peut varier entre 350 et 500 avec un pas de 50, le nombre de têtes est 2 ou 4, la dimension interne du bloc FFN peut prendre les valeurs entre 800 et 2 000 avec un pas de 200. Nous faisons aussi varier le nombre de couches entre 11 et 21 avec un pas de 2.

Pour les paramètres d'apprentissage, nous gardons ceux de Flashlight. Nous utilisons l'optimiseur Adam avec un pas d'apprentissage de 0,000 1, un moment d'ordre 1 de 0,9 et un moment d'ordre 2 de 0,999. Les gradients sont tronqués à 1. La taille du *batch* est réglée à 96 phrases réparties sur 8 GPU. Pendant les 32 000 premières itérations de *batch*, le pas d'apprentissage augmente linéairement de 0 à 0,000 1. Nous effectuons 400 itérations sur le jeu de données, ce qui représente 119 200 itérations de *batch*. Il est à noter que nous activons une couche SpecAugment comme première couche de l'architecture au bout de 8 000 itérations de *batch*.

Nous réalisons l'apprentissage de toutes les configurations d'architecture sur le jeu de données *train-clean-100* et nous évaluons le WER sans modèle de langage sur *dev-clean*. La plus petite architecture a 22 millions de paramètres et la plus grande 76. La meilleure configuration trouvée a 15 couches Transformer avec une taille de tête d'auto-attention de 450, 4 têtes et une dimension interne du bloc FFN de 1 800. Le WER sur *dev-clean* est évalué à 12,56.

4.4.2. Évaluation des fonctionnalités

Dans cette partie, nous évaluons les différentes fonctionnalités de la fonction de coût à l'exception du lexique et du modèle de bruit. Nous gardons donc les lettres comme symboles pendant l'expérience. Nous utilisons l'architecture Transformer trouvée auparavant avec les mêmes configurations d'apprentissage à l'exception que nous ajoutons 20 itérations sur le jeu de données où le pas d'apprentissage est divisé par 2.

Première expérience Pour cette expérience, nous utilisons le symbole **R1** en activant le module R correspondant. Nous faisons varier les paramètres suivants :

— Normalisation des scores acoustiques au niveau de la trame ou au niveau de la séquence.

— Utilisation du symbole blanc β ou non.

— Utilisation du modèle bi-gramme B ou non.

— Apprentissage du modèle sur les transcriptions de référence de *train-clean-100* ou sur les jeux de données bruitées D39 et D21.

Le Tab. 4.3 indique le WER sur *dev-clean* de ces expériences. Nous n'utilisons pas ici de modèle de langage pour le décodage.

Pour la méthode de normalisation par séquence, nous observons que certains modes ne font pas converger le modèle. Il semble que l'utilisation du symbole blanc β provoque ce phénomène. Avec ce mode de normalisation et sans le symbole β, nous observons également une légère amélioration avec le modèle bi-gramme. Ce dernier cas correspond en fait à ASG. Il est à noter que le cas sans modèle bi-gramme est équivalent mathématiquement à une normalisation par trame.

Normalisation	Par trame					Par séquence				
β Bi-gramme	CTC	Non Non	Non Oui	Oui Non	Oui Oui	ASG	Non Non	Non Oui	Oui Non	Oui Oui
D39	36, 8	37, 2	77, 1	37, 2	78, 3	36, 1	37, 4	/	/	/
D21	21, 0	21, 2	57, 8	20, 9	59, 1	20, 1	21, 3	20, 5	/	19, 9
Référence	11, 5	11, 3	38, 4	10, 9	38, 5	10, 4	11, 3	10, 7	/	/

TABLEAU 4.3. – WER obtenus sur *dev-clean* avec des modèles acoustiques appris sur les jeux de données de référence (*train-clean-100*), D21 et D39. Nous reportons les résultats pour différentes combinaisons d'options de la fonction de coût. Ici, R est activé et nous faisons varier les options du bi-gramme, du symbole blanc β et du mode de normalisation. Nous reportons également les résultats pour les fonctions de coûts CTC et ASG classiques. Les résultats non-reportés signifient que l'expérience a échoué : le modèle ne converge pas.

Pour la méthode de normalisation par trame nous n'avons pas de problème de convergence mais nous observons que l'utilisation du modèle bi-gramme provoque une nette augmentation du WER. Nous observons également des WER similaires, que β soit activé ou non. Ici R est activé donc nous avons un mécanisme de modélisation de répétition de lettres dans les deux cas.

Seconde expérience Pour cette expérience nous n'utilisons que la méthode de normalisation par trame et nous n'activons pas le modèle bi-gramme. Nous faisons varier l'utilisation du module R de répétition de symboles et de β. Nous utilisons le même processus d'apprentissage que la première expérience. En plus du jeu de données de référence, nous reportons le WER sur *dev-clean* pour des modèles acoustiques appris avec les transcriptions de référence de *train-clean-100* ou sur les jeux de données bruitées D65, D39 et D21. Le Tab. 4.4 rapporte les résultats.

Normalisation	Par trame				
R β	CTC	Non Non	Non Oui	Oui Non	Oui Oui
D65	65, 1	65, 4	66, 6	63, 8	64, 5
D39	36, 8	42, 3	41, 9	37, 4	37, 0
D21	21, 0	28, 5	28, 3	21, 2	20, 8
Référence	11, 5	19, 3	19, 5	11, 3	10, 9

TABLEAU 4.4. – WER obtenus sur *dev-clean* avec des modèles acoustiques appris sur les jeux de données de référence (*train-clean-100*), D21 et D39 et D65. Nous utilisons un système de normalisation par trame et n'utilisons pas de modèle bi-gramme. Nous reportons les résultats en faisant varier l'utilisation du module R et de β. Nous reportons également le WER pour la fonction de coût CTC classique.

Nous observons des mauvais résultats lorsque R n'est pas activé. L'utilisation de β seul ne permet pas d'atteindre les WER obtenus avec CTC. Nous supposons que c'est dû au fait que nous ne forçons pas β à être utilisé lorsque deux symboles identiques se suivent. Pour G_{CTC}, nous pouvions le faire en modifiant directement le graphe. Notre implémentation avec la formulation de T ne permet pas cela.

De ces deux expériences, nous concluons que les options qui permettent d'obtenir les meilleurs résultats sont d'utiliser une normalisation par trame, avec R, sans β et sans modèle bi-gramme.

4.4.3. Apprentissage du modèle de bruit

Pour cette expérience, nous nous concentrons sur la structure N_1 (présentée partie 4.3.1) du modèle de bruit. Nous voulons apprendre les poids de ce wFST à partir d'un jeu de données contenant les transcriptions de référence et celles bruitées. Nous utilisons les transcriptions bruitées du jeu de données $\text{D21}_{\text{N}_{\text{train}}}$ contenant 10 heures de transcriptions décodées avec le modèle acoustique de l'itération 4 000. Pour cette expérience nous utilisons des morceaux de mots comme symboles. Cette expérience vise à déterminer quelle est la meilleure taille du dictionnaire de symboles $\mathcal{D}_N$ pour obtenir un modèle de bruit de transcription efficace.

Mesure de qualité d'un modèle de bruit Il n'est pas évident d'évaluer la qualité d'un modèle de bruit. À partir d'une transcription bruitée $\tilde{Y}$, nous pouvons songer à évaluer la meilleure transcription de référence Y^* selon le modèle de bruit en calculant

$$Y^* = \text{Viterbi}(N \circ \tilde{Y}), \tag{4.34}$$

où Viterbi(.) donne le chemin avec le meilleur score. Mais avec cette méthode, nous avons toujours $Y^* = \tilde{Y}$ car selon le modèle de bruit, il est toujours plus probable de garder la même transcription. C'est un comportement attendu : le modèle de bruit de Lead2Gold avait bien une diagonale dominante. Nous pourrions créer une mesure plus en adéquation avec l'utilisation réelle que l'on fait du modèle de bruit, par exemple en faisant intervenir les scores acoustiques. Nous gardons cette expérience simple, et nous utilisons plutôt la valeur de la fonction de coût (4.29) évaluée sur le jeu de données de développement $\text{D21}_{\text{N}_{\text{dev}}}$.

Choix des paramètres d'apprentissage Nous faisons une première expérience pour choisir les paramètres d'apprentissage. Nous utilisons simplement les lettres comme symboles pour cette étape. Nous utilisons les optimiseurs Adam ou SGD. Pour Adam, nous laissons les moments d'ordre 1 et 2 par défaut à 0,9 et 0,999. Nous faisons varier le pas d'apprentissage, la taille du *batch* et le choix de l'optimiseur et nous faisons un apprentissage pendant 50 itérations du jeu de données d'apprentissage. Selon notre mesure de qualité, la meilleure combinaison trouvée utilise l'optimiseur Adam avec un pas d'apprentissage de 0,005 et une taille de *batch* de 20.

Résultats pour différentes tailles de dictionnaires de symboles $\mathcal{D}_N$ Avec les paramètres d'apprentissage trouvés, nous faisons l'expérience en utilisant des morceaux de mots comme symboles. Nous faisons varier la taille du dictionnaire de symboles $\mathcal{D}_N$ avec les valeurs $[28, 50, 100, 150, 200, 250, 300, 400]$. La taille 28 correspond à l'utilisation de lettres (avec l'espace et l'apostrophe). Nous utilisons les transcriptions de *dev-clean* pour apprendre les modèles unigrammes. Le Tab. 4.5 reporte les résultats pour ces configurations. Il apparaît que plus le modèle de bruit est grand, plus il est difficile à apprendre. L'explication est que plus il y a de symboles, plus les probabilités faibles sont sous-estimées. Le même problème survient pour l'apprentissage de modèles de langage. Dans ce cas il existe des méthodes de lissage pour corriger cela comme la méthode de lissage de Kneser-Ney [Kneser and Ney, 1995] mais nous n'avons pas envisagé d'explorer cette direction pour l'adapter au modèle de bruit. Le meilleur résultat, selon la mesure de qualité, reste l'utilisation d'un nombre minimal de symboles, à savoir les 28 lettres. Le résultat avec 50 symboles est relativement proche de ce dernier. Les symboles trouvés pour la taille 50 sont les 28 lettres plus les morceaux de mots **e|**, **t|**, **s|**, **the|**, **y|**, **th**, **ed|**, **d|**, **and|**, **er**, **an**, **or**, **of|**, **er|**, **in**, **to|**, **re**, **ing|**, **ar**, **en**, **on**, **in|**.

La méthode unigramme pour trouver les morceaux de mots minimise le nombre de symboles nécessaires pour encoder une transcription. À la place, il faudrait imaginer un algorithme permettant de trouver un ensemble de symboles qui minimise le nombre de transformations nécessaires pour passer d'une transcription à une autre.

Nombre de symboles	28	50	100	150	200	250	300	400
Coût obtenu	0,72	0,94	1,13	1,21	1,28	1,34	1,39	1,49

TABLEAU 4.5. – Résultats d'apprentissage de modèles de bruit pour différentes tailles de dictionnaires. Nous reportons la valeur de la fonction de coût calculée sur le jeu de données de développement $\mathrm{D21}_{\mathrm{N_{dev}}}$.

4.4.4. Évaluation du coût de calcul

Dans cette partie, nous évaluons le temps de calcul de la fonction de coût. Nous nous plaçons dans un cas complexe avec l'utilisation du module R, du lexique et du modèle de bruit. Nous n'utilisons pas le modèle bi-gramme ni le symbole blanc β. Pour le lexique, nous encodons les 10 000 mots les plus utilisés de *train-clean-360*. Nous utilisons les lettres comme symboles pour le modèle de bruit. Nous utilisons le modèle de bruit appris dans la partie précédente sur $\mathrm{D21}_{\mathrm{N_{train}}}$ mais nous appliquons une opération d'élagage des arcs en choisissant $p_{\mathrm{seuil}} = 0.986$. Cela nous permet de ne garder que 110 arcs parmi 841.

Le calcul de la fonction de coût comporte 3 opérations : l'opération de composition $T\tilde{Y} = T \circ \tilde{Y}$, l'opération de composition $AT\tilde{Y} = A \circ T\tilde{Y}$ et le calcul de $\mathrm{Score}(ATY)$. Pour la transcription $\tilde{Y}$ nous considérons deux exemples : un exemple court de 1,6 seconde

et un exemple de longueur moyenne de 8,3 secondes. Ces deux exemples proviennent du jeu de données D21. Les scores acoustiques associés du wFST A viennent d'un modèle acoustique pré-entraîné sur D21 avec les mêmes options de fonction de coût mais sans le modèle de bruit et le lexique (partie 4.4.2). Le Tab. 4.6 résume les tailles des wFST communs aux deux exemples. Le Tab. 4.7 présente les tailles des wFST obtenus après les deux opérations de composition (effectuées avec GTN), et cela pour les deux exemples.

	Nombre d'états	Nombre d'arcs
MAP	30	87
R	29	812
L	7 020	15 316
N	2	110
$T = \text{MAP} \circ R \circ L \circ N$	47 181	184 253

TABLEAU 4.6. – Taille des wFST utilisés pour l'évaluation du coût de calcul. Nous reportons ici ceux qui servent pour les deux exemples.

	Phrase courte		Phrase moyenne	
	Nombre d'états	Nombre d'arcs	Nombre d'états	Nombre d'arcs
$T \circ Y$	84 377	144 781	1 568 427	2 697 107
$A \circ T \circ Y$	2 401 363	4 084 208	171 298 503	293 990 132

TABLEAU 4.7. – Taille des wFST utilisés pour le test de performance. Nous reportons ici les tailles obtenues avec l'opération de composition de GTN. Le wFST générée par $A \circ T \circ Y$ pour l'exemple de longueur moyenne utilise 5,5 Go de mémoire.

Nous observons que même sur un exemple de taille moyenne les wFST résultant des étapes de compositions deviennent très grands. Comme nous allons le voir, le calcul des opérations de composition avec GTN est très lent et ne permet pas d'utiliser la fonction de coût en pratique. C'est pourquoi nous essayons également l'algorithme de composition de K2 pour effectuer une comparaison. K2 propose deux autres algorithmes de composition que nous noterons $\circ_{\text{dense}}$ et $\circ_{\text{dense,max}}$. Ce sont en fait des implémentations d'algorithmes de recherche en faisceau. Ils effectuent la composition entre un wFST quelconque et un wFST « dense » comme celui portant les scores acoustiques A. Ils sont donc adaptés pour le calcul de $A \circ T\tilde{Y}$. Pour $\circ_{\text{dense}}$, il faut choisir un score à partir duquel les hypothèses sont élaguées. Pour $\circ_{\text{dense,max}}$, il faut indiquer en plus le nombre maximal d'hypothèses à explorer. Pour notre exemple, nous réglons ces paramètres de façon à ce que le score obtenu soit identique (à 10^{-3} près) quelle que soit la méthode de composition utilisée. Le score limite est réglé à 50 pour les deux exemples. Le nombre d'hypothèses maximal est réglé à 20 pour l'exemple court et à 100 pour l'exemple moyen. De plus, K2 fournit des implémentations sur CPU et sur GPU.

Le Tab. 4.8 indique les temps mesurés pour les différentes étapes avec les différentes implémentations de l'algorithme de composition, et cela pour les deux exemples. L'expérience est effectuée avec les interfaces Python des bibliothèques GTN et K2. Nous avons créé une fonction permettant de transformer le format d'un wFST de GTN en celui de K2. Ainsi le wFST T est créé entièrement avec GTN puis il est transformé en un format accepté par K2.

		GTN	K2 CPU			K2 GPU		
		$\circ$	$\circ$	$\circ_{\text{dense}}$	$\circ_{\text{dense,max}}$	$\circ$	$\circ_{\text{dense}}$	$\circ_{\text{dense,max}}$
Phrase	$T \circ \tilde{Y}$	0,085	0,047			0,009		
courte	$A \circ \dots$	2,260	1,780	0,070	0,002	0,013	0,022	0,010
(1,6 s)	Score(...)	0,336	0,442	2×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
Phrase	$T \circ \tilde{Y}$	1,432	1,165			0,136		
moyenne	$E \circ \dots$	240	840	—	0,048	—	—	0,048
(8,3 s)	Score(...)	44,5	54,4	—	1×10^{-5}	—	—	1×10^{-5}

TABLEAU 4.8. – Coût de calcul des opérations de composition avec GTN et K2. Le temps reporté est en secondes. Les tirets indiquent que l'opération de composition a échoué à cause d'une erreur de maximum de mémoire atteint.

Nous observons que l'algorithme de composition $\circ_{\text{dense,max}}$ de K2 permet de calculer la fonction de coût très rapidement que ce soit avec l'implémentation CPU ou GPU. Il y a un avantage à utiliser l'implémentation GPU de K2 pour le calcul de $T \circ \tilde{Y}$. De plus, $\circ_{\text{dense,max}}$ permet de limiter la taille du wFST généré et évite de saturer les capacités de mémoires. Cet algorithme répond donc à notre problème de vitesse d'exécution et permettrait d'implémenter correctement notre fonction de coût modulaire. Mais l'utilisation de K2 avec Flashlight est difficile à mettre en place avec l'interface C++, d'autant plus qu'il faut prendre en compte la rétro-propagation. La prochaine étape serait donc d'implémenter le même algorithme pour GTN.

4.4.5. Conclusion

Nous avons justifié précisément comment adapter Lead2Gold avec des wFST. La formulation est modulaire et facile à construire. Toute la difficulté se cache maintenant dans l'algorithme de composition qui agrège les différents modules. Pour Lead2Gold, nous avions anticipé la complexité exponentielle en calculant une approximation avec une recherche en faisceau différentiable. Avec les wFST, nous avons essayé de restreindre le nombre d'hypothèses en élaguant le modèle de bruit et en utilisant un lexique. L'évaluation du coût de calcul montre que cela ne suffit pas et qu'il faut un contrôle sur la complexité de l'algorithme de composition. Nous avons montré que cela est possible en utilisant un équivalent de la recherche en faisceau pour les wFST. De nombreuses difficultés ne nous ont malheureusement pas permis de mettre tous ces éléments en commun.

5. Collecte de transcriptions

Dans ce chapitre nous menons une expérience de collecte de transcriptions en conditions réelles. La partie 5.1 introduit les contraintes de l'expérience ainsi que la méthode utilisée pour collecter les données. La partie 5.2 présente comment l'expérience a été menée. La partie 5.3 analyse les données collectées. Nous y menons notamment une analyse descriptive des erreurs commises par les annotateurs. Nous conduisons également une expérience d'apprentissage d'un modèle acoustique à partir des données collectées.

5.1. Mise en place de l'expérience

5.1.1. Outils utilisés

La plateforme de *crowdsourcing* AMT Pour collecter les transcriptions, nous utilisons l'outil *Amazon Mechanical Turk* (AMT). C'est une plateforme de *crowdsourcing* permettant de mettre en relation des travailleurs avec des personnes qui proposent des tâches à effectuer. Nous appellerons « démarcheur » une personne qui propose une tâche. Les tâches prennent la forme d'une page Web accessible par le travailleur. La plupart des tâches sont simples et répétitives. Elles consistent la plupart du temps à annoter des données. Pour cette expérience, nous prenons le rôle de démarcheur pour collecter des transcriptions.

La bibliothèque Mephisto AMT propose une interface de programmation (API) complète afin d'envoyer des tâches, de gérer les droits des travailleurs ou encore de récupérer les résultats. Celle-ci est de bas niveau et son utilisation est difficile. Nous utilisons à la place la bibliothèque Mephisto [1] qui simplifie l'utilisation des plateformes de *crowdsourcing*. La gestion de l'envoi des tâches et de la réception des résultats est transparente avec Mephisto. Elle permet aussi un contrôle fin des droits des travailleurs sur les tâches, ce qui, comme nous allons le voir, va nous être utile. La bibliothèque met aussi à disposition des outils pour passer en revue les résultats et permettre de valider la tâche. Cette action entraîne alors automatiquement le paiement au travailleur. La conception de l'interface Web pour les tâches reste cependant à effectuer.

La bibliothèque React React [2] est une bibliothèque JavaScript permettant de créer des interfaces graphiques interactives. Nous l'utilisons pour créer notre interface Web. Avec cette bibliothèque, nous pouvons intégrer des éléments complexes dans une page HTML.

1. github.com/facebookresearch/Mephisto
2. fr.reactjs.org

5.1.2. Objectifs et contraintes

Objectifs de l'expérience Nous voulons collecter des transcriptions contenant des erreurs qu'un humain peut commettre. Nous ne voulons pas obtenir les meilleures transcriptions possibles. Les entreprises spécialisées dans l'annotation de données peuvent obtenir des transcriptions avec très peu d'erreurs. L'obtention de telles transcriptions demande alors l'effort de plusieurs personnes qualifiées. Pour cette expérience, nous voulons collecter des transcriptions de qualité moyenne par un procédé facilement reproductible.

L'utilisation d'une plateforme de *crowdsourcing* convient parfaitement pour cela. Les travailleurs ne sont pas spécialement qualifiés. Nous voulons même exagérer les conditions : les travailleurs ont un temps limité pour chaque tâche et ils ne sont autorisés à écouter chaque enregistrement de parole que deux fois.

Contraintes budgétaires Nous disposons d'un budget pour la collecte de données. Ce budget est mis à disposition par l'entreprise Meta qui a financé cette thèse CIFRE. Bien qu'AMT ne fixe aucune règle concernant la rémunération des travailleurs, nous fixons la rémunération à un taux horaire considéré comme excellent par rapport à la rémunération moyenne pratiquée sur AMT. Nous devons aussi prendre en compte la commission de 20% ponctionnée par AMT sur toutes les transactions.

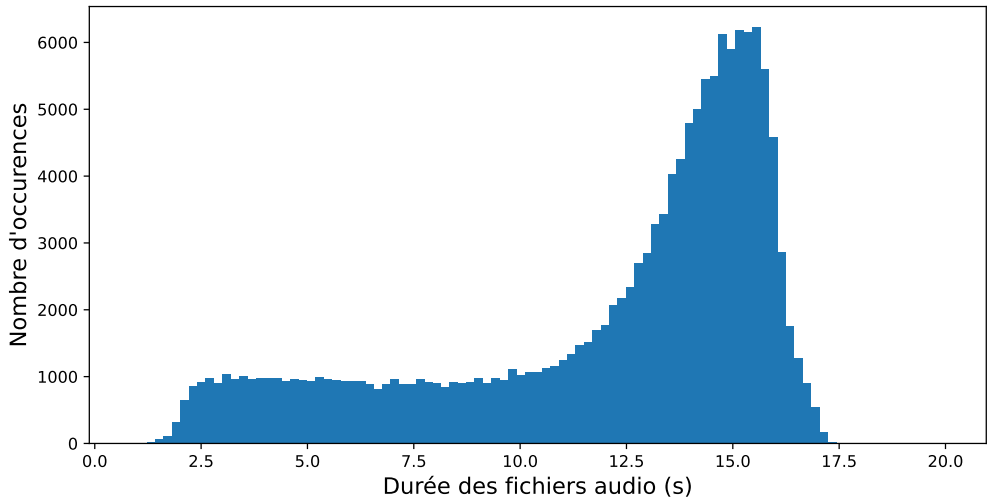
Contraintes légales Pour des raisons légales, Meta interdit d'utiliser les services de travailleurs qui résident en dehors du sol américain. Cela implique une forte contrainte sur la langue du jeu de données que nous faisons transcrire. Nous choisissons donc un jeu de données en anglais car il serait difficile de trouver des travailleurs pour d'autres langues. De plus, il faut utiliser un jeu de données avec une licence d'utilisation permissive car nous le stockons et le modifions. Enfin, les données sont stockées sur un serveur qui n'est pas administré par Meta, donc l'utilisation de données internes à l'entreprise n'est pas autorisée.

5.1.3. Le jeu de données à transcrire

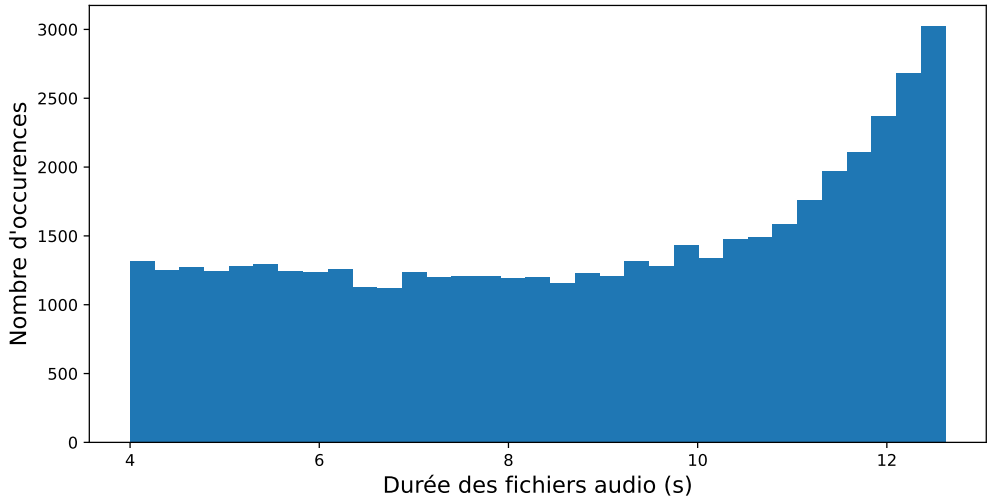
Utilisation de Librispeech Nous faisons le choix de faire transcrire aux travailleurs une partie du jeu de données Librispeech. Celui-ci possède la licence *Creative Commons attribution 4.0* (CC BY 4.0), ce qui nous laisse le droit de transformer le jeu de données et de le partager. Ce dernier contient de la parole lue en anglais donc les transcriptions fournies (en réalité, les textes lus) sont censées correspondre exactement aux mots prononcés. Nous considérons que ces transcriptions ne contiennent aucune erreur. Nous aurons ainsi un élément de référence pour analyser les transcriptions que nous collectons. Pour que la tâche soit assez difficile, nous utilisons les fichiers audio de *train-other-500* de Librispeech qui sont considérés comme plus difficiles à comprendre.

Filtrage des fichiers audio Pour faciliter la collecte des transcriptions, nous filtrons les fichiers audio les plus longs et les plus courts de *train-other-500*. Nous ne voulons pas qu'il

y ait une trop grande différence de durée des fichiers audio entre les tâches pour garder une difficulté similaire. La Fig. 5.1a représente la distribution des durées des fichiers audio de *train-other-500*. Ils durent entre 0,8 seconde et 27,9 secondes avec une moyenne de 12 secondes. Nous choisissons d'utiliser les fichiers audio de durée comprise entre 4 et 12,6 secondes. Il reste alors un peu plus de 48 000 fichiers avec une durée moyenne de 8,8 secondes, ce qui représente 118 heures de parole. Nous les regroupons aléatoirement en 24 paquets de 2 000 fichiers. Nous n'avons pas pour objectif de tout utiliser. La distribution des durées de ces fichiers est représentée dans la Fig. 5.1b.



(a) Distribution des durées des fichiers audio de *train-other-500*.



(b) Distribution des durées des fichiers audio utilisés pour l'expérience.

FIGURE 5.1. – Distribution des tailles des fichiers audio dans le jeu de données *train-other-500* avant (a) et après filtrage des fichiers trop longs et trop courts (b).

Stockage des données Les fichiers audio sont stockés sur un serveur AWS accessible par tout le monde. Il est plus facile de procéder de cette manière afin que la page Web utilisée par AMT puisse directement charger les signaux sonores à partir de ce serveur. Par contre, toutes les données produites par les travailleurs sur nos tâches sont envoyées sur un serveur interne. Il faut être vigilant sur ce point car les solutions « faciles » que propose AMT stockent les résultats sur leurs serveurs ce qui permet hypothétiquement à Amazon de les utiliser.

5.1.4. L'interface Web

Nous montrons ici l'interface Web que nous avons créée pour la collecte des données. Celle-ci est composée de deux parties. Le haut de la page Web contient les consignes que le travailleur doit suivre. La Fig. 5.2 montre cette partie. Les consignes sont les mêmes pour toutes les tâches donc le travailleur peut les lire une fois lors de sa première tâche et passer directement à la suite pour les tâches suivantes. En dessous des consignes, le travailleur a la possibilité de montrer une tâche de test (fond vert de la Fig. 5.2). Celle-ci est optionnelle et a pour but de laisser au travailleur la possibilité de se familiariser avec l'interface. Il peut ainsi s'habituer aux contraintes de l'expérience.

Le bas de la page Web contient la véritable tâche de transcription. La tâche peut commencer en appuyant sur le bouton *Start*. La Fig. 5.3 montre la tâche avant et après activation de ce bouton. Lorsque le bouton start est activé, le lecteur du fichier audio apparaît et le décompte du temps restant pour transcrire se déclenche. Nous procédons de cette manière car lors de nos premiers tests, un travailleur soumettait des résultats en trichant : il suffisait d'inspecter l'élément de la page html ou d'utiliser le clic droit sur le lecteur pour trouver où étaient stockés les fichiers audio et ainsi pouvoir transcrire sans la contrainte de temps. Comme le lecteur apparaît au moment où le chronomètre se déclenche, cette stratégie de triche n'est plus valable. Pour plus de prudence, nous désactivons aussi le clic droit sur toute la page. Nous imposons une autre contrainte que le temps : chaque fichier audio ne peut être écouté que deux fois. Il est impossible de cliquer sur le lecteur pour revenir en arrière. Le travailleur peut seulement activer le lecteur ou le mettre en pause avec le bouton *Play/Pause*. Mettre en pause le fichier audio ne met pas en pause le chronomètre.

Lorsque le bouton *Start* est activé, un champ de texte devient sélectionnable et le travailleur peut y écrire la transcription. Le travailleur peut faire une pause à tout moment pour prendre le temps d'écrire. Au lieu d'être obligé d'utiliser le bouton *Play/Pause* et de sortir du champ, nous ajoutons un raccourci clavier pour déclencher cette action.

Lorsque le temps est écoulé, le lecteur s'arrête et il n'est plus possible d'écrire. Le travailleur doit alors cliquer sur *Submit* pour passer au fichier audio suivant. Une tâche consiste à transcrire un certain nombre de fichiers audio. Lorsque le travailleur soumet sa dernière transcription, la tâche s'arrête et les données nous sont envoyées.

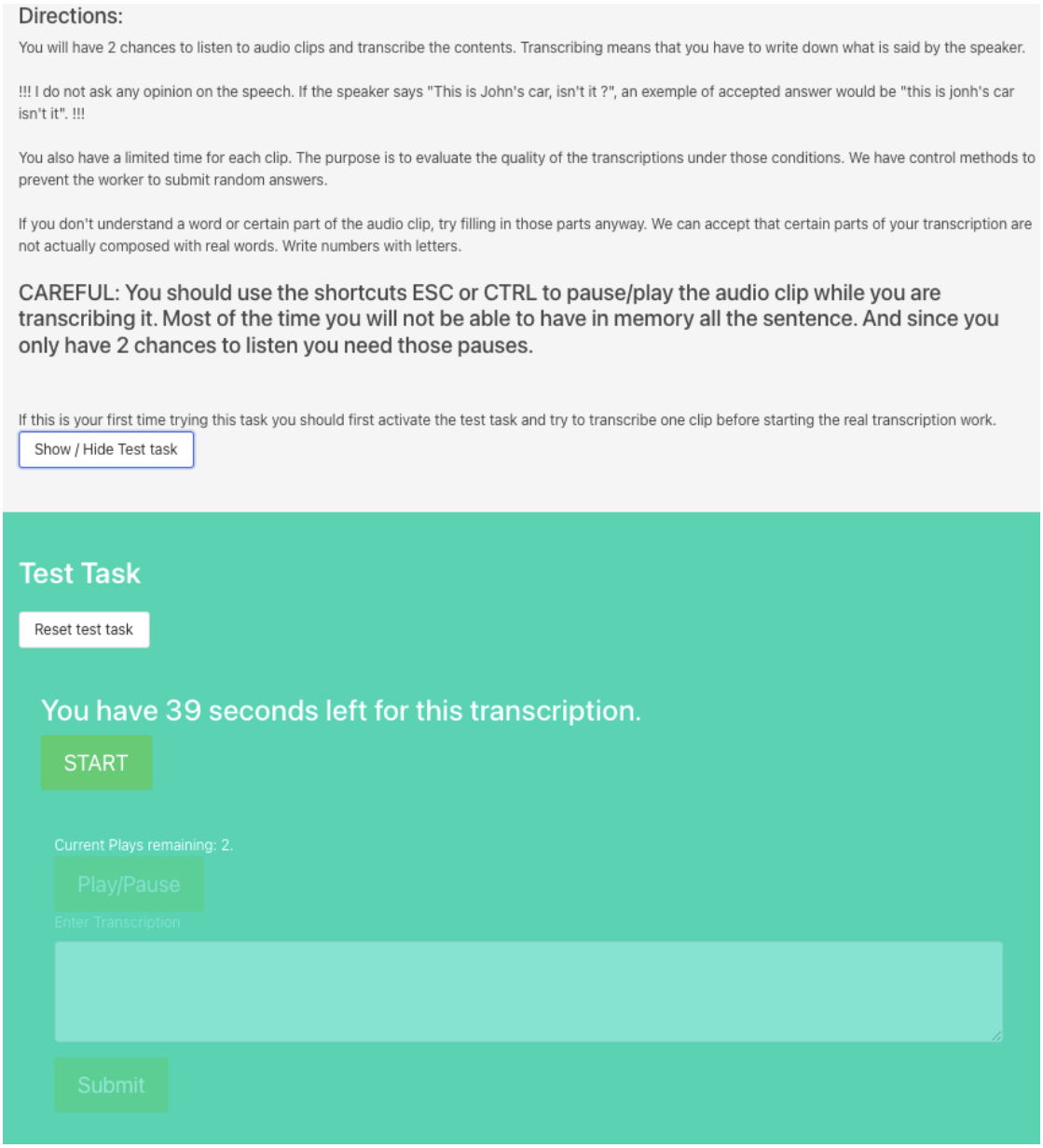
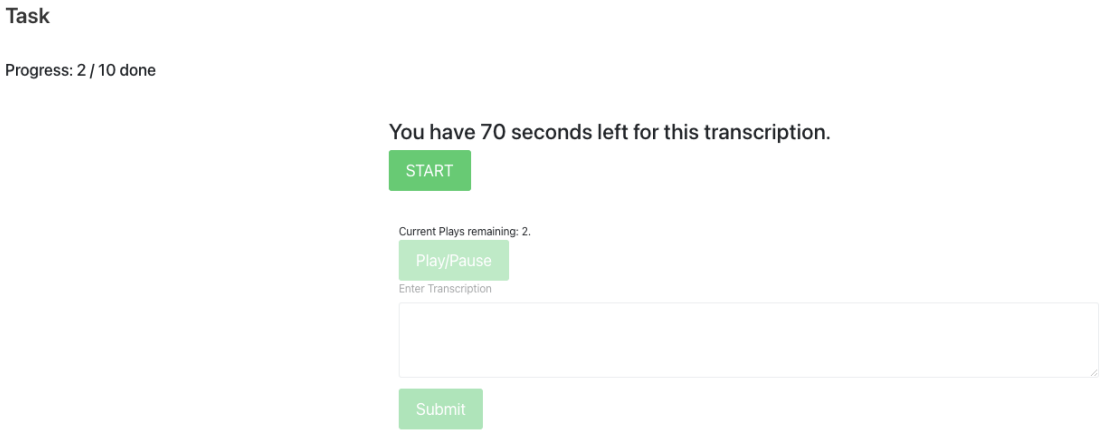
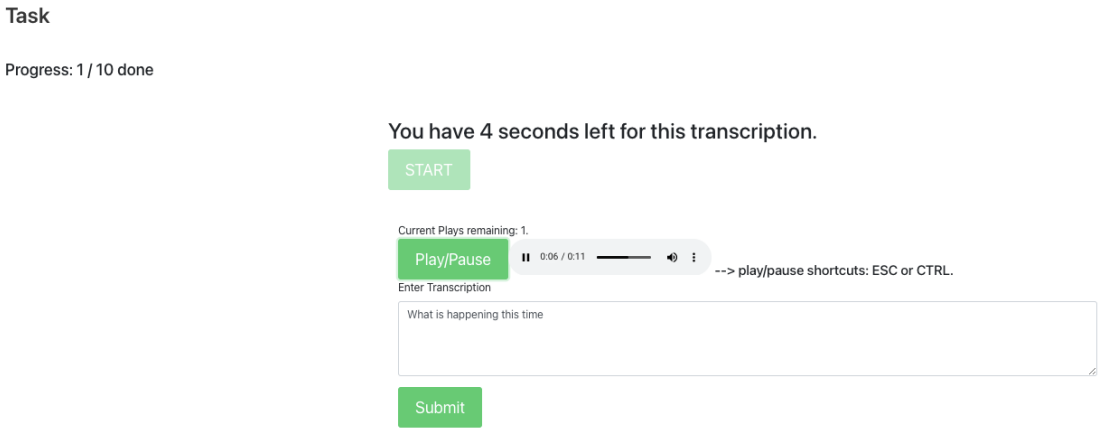


FIGURE 5.2. – Capture d'écran de la tâche de test et des consignes que les travailleurs doivent suivre.

Nous enregistrons la plupart des actions effectuées par les travailleurs. Nous savons ainsi à quel moment chaque bouton est cliqué. Nous en déduisons combien de temps un travailleur passe sur chaque transcription et combien de fois il utilise le bouton pause.



(a) Avant activation du bouton *Start*.



(b) Après activation du bouton *Start*.

FIGURE 5.3. – Captures d'écran de la tâche de transcription avant et après l'activation du bouton *Start*.

5.2. La collecte des données

5.2.1. Calibrations et tests

Première calibration Il y a plusieurs paramètres à déterminer avant d'effectuer un test avec des travailleurs. Pour cela, nous testons nous-mêmes la tâche sur 4 personnes du laboratoire. Chaque tâche nécessite d'effectuer un certain nombre de transcriptions. Nous ne voulons pas une seule transcription par tâche car alors les tâches seraient de durées trop variables. Pour commencer, nous avons testé avec 25 transcriptions par tâche sans contrainte de temps. En pratique, 20 à 30 minutes sont alors nécessaires pour compléter

une tâche. À cause de contraintes techniques, les données ne sont envoyées qu'à la fin de la tâche. Nous pensons qu'il faut un retour plus régulier pour éviter de perdre des données en cas de problème. Nous avons donc réduit à 10 le nombre de transcriptions à effectuer pour une tâche. À partir de nos résultats, nous avons déterminé combien une tâche doit être payée pour que les travailleurs atteignent le taux de rémunération horaire voulu. Nous avons également obtenu une première estimation du temps nécessaire pour transcrire un fichier audio. La difficulté est assez variable d'un fichier à un autre et certains demandent plus de temps. Nous avons ainsi estimé le rapport typique entre la durée d'un fichier audio et le temps nécessaire maximal pour le transcrire. Nous ne voulons pas que le chronomètre soit un élément trop frustrant pour les travailleurs et nous réglons ce rapport à 7 : pour un fichier de 10 secondes, nous réglons le chronomètre à 70 secondes.

Calibration en conditions réelles Comme premier test, nous avons mis à disposition des travailleurs 25 tâches contenant chacune 10 transcriptions à faire, en interdisant à un travailleur de faire plusieurs tâches. Nous avons analysé les résultats manuellement et décidé d'accepter seulement 5 de ces tâches. C'est assez décevant pour un premier test. Parmi les travailleurs non acceptés, il y a trois catégories. Une grande partie a fait preuve de mauvaise foi. Certains n'ont pas écouté les fichiers audio et ont rendu des transcriptions aléatoires. Nous le savons car nous avons enregistré les actions réalisées par chaque travailleur sur la page. Ces travailleurs n'ont pas été rémunérés. D'autres sont simplement mauvais et n'ont transcrit que quelques mots. Enfin certains n'ont pas compris la consigne : ils n'ont pas transcrit le fichier mais ont donné un avis sur ce qui est dit tel que « Je suis d'accord avec ce qui est dit » ou « la personne parle correctement ». Nous avons donc clarifié les consignes de la tâche pour la prochaine expérience. Nous considérons que les travailleurs de ces deux dernières catégories ne sont pas de mauvaise foi et nous les avons rémunérés.

Pour les 5 tâches acceptées, le rapport moyen entre la durée des fichiers et le temps nécessaire à la transcription est de 4,75. Le ratio maximum est égal à 7 ce qui veut dire que le temps imparti à été utilisé dans sa totalité. Nous gardons donc ce rapport fixé à 7.

Épreuve de qualification Nous avons réagi à ces résultats en créant une procédure de filtrage des travailleurs. Nous créons des tâches de « qualification » où les travailleurs doivent transcrire 4 fichiers audio. L'évaluation des résultats est semi-automatisée. Nous calculons le LER entre les transcriptions fournies par les travailleurs et les transcriptions de référence de Librispeech. Si le LER est inférieur à 30% alors nous validons automatiquement l'épreuve de qualification. Le travailleur peut alors passer au véritable travail de transcription sans avoir besoin de notre validation manuelle. Si le LER est entre 30% et 70% nous refusons la qualification mais nous rémunérons le travailleur. Si le LER est au-dessus de 70% nous vérifions manuellement les transcriptions fournies : si le travailleur a fait preuve de mauvaise foi alors il n'est pas rémunéré. Un travailleur ne peut tenter de

se qualifier qu'une seule fois. Nous choisissons manuellement 60 fichiers audio pour ces tâches de qualification. Les 4 fichiers audio de chaque tâche sont choisis aléatoirement parmi les 60 sélectionnés. De cette façon, nous nous assurons que les tâches de qualification sont différentes mais de difficulté similaire. Les travailleurs communiquent entre eux et nous voulons éviter toute stratégie pour être accepté.

5.2.2. Lancement de la véritable expérience

Nous avons collecté les transcriptions sur une période de deux semaines. Au fur et à mesure des besoins, nous avons laissé la possibilité à de nouveaux travailleurs de se qualifier en créant de nouvelles tâches de qualification. Nous ne voulons pas trop de travailleurs différents et voulons laisser la possibilité à un travailleur de faire un nombre important de tâches. L'objectif est d'avoir assez de transcriptions par travailleur pour analyser les erreurs caractéristiques de chacun de ces travailleurs. Nous avons tout de même imposé une limite : lorsqu'un travailleur a effectué environ 80 tâches, nous révoquons sa qualification pour qu'il ne puisse plus en faire. L'expérience se termine lorsque tout le budget a été utilisé.

Lors de l'expérience, 224 travailleurs ont essayé une tâche de qualification. 128 ont réussi, 76 n'ont pas réussi mais ont été rémunérés, 20 n'ont pas réussi et n'ont pas été rémunérés. Ces tâches représentent 3% de notre budget.

Parmi les 128 travailleurs qualifiés, 91 ont terminé au moins une tâche complète de transcription. Au total, 2 600 tâches ont été terminées. Nous n'utilisons pas d'acceptation automatique ici. Nous évaluons les tâches une par une en nous aidant du LER entre les transcriptions fournies par le travailleur et celles de Librispeech. Nous acceptons systématiquement les tâches avec un LER moyen en dessous de 30%. Nous regardons plus attentivement les autres tâches. Seuls 19 tâches ont été écartées mais rémunérées. Ces tâches ne sont souvent que partiellement complétées. Nous révoquons la qualification des travailleurs concernés. Une seule tâche a été écartée et non rémunérée car pratiquement aucun travail n'a été fourni. Nous obtenons donc un bien meilleur taux d'acceptation avec le système de qualification.

Avec les tâches acceptées, nous avons collecté 25 800 transcriptions pour 63 heures et 10 minutes de fichiers audio.

Normalisation du texte Pour l'évaluation du LER pendant la phase de collecte, nous normalisons les transcriptions avec une procédure simple : nous transformons les majuscules en minuscules, nous enlevons les espaces et les sauts de ligne et nous supprimons les caractères qui ne sont pas des lettres (sauf les apostrophes).

Pour analyser et utiliser les transcriptions collectées, nous utilisons une procédure de normalisation du texte plus complète. Les nombres cardinaux et ordinaux sont remplacés

par l'équivalent textuel. Certains symboles et abréviations courants sont également remplacés : par exemple le symbole « $ » devient « *dollar* » ou « *dollars* » selon le contexte.

5.3. Exploitation des données collectées

Nous étudions dans cette partie les transcriptions collectées. La section 5.3.1 décrit les erreurs de transcription en fonction des travailleurs. La section 5.3.2 exploite les données collectées pour l'apprentissage de modèles acoustiques.

5.3.1. Analyse descriptive

Répartition des transcriptions par travailleur La Fig. 5.4 présente le nombre d'heures de parole transcrite par travailleur. Cette distribution est inégale. Sur les 91 travailleurs qui participent à la collecte de transcription, seuls 31 ont transcrit au moins 1 heure de parole. Ces 31 travailleurs les plus productifs ont collecté ensemble 53 heures et 6 minutes sur le total de 63 heures et 10 minutes. Pour l'analyse des erreurs de transcription nous nous concentrons sur ces 31 personnes.

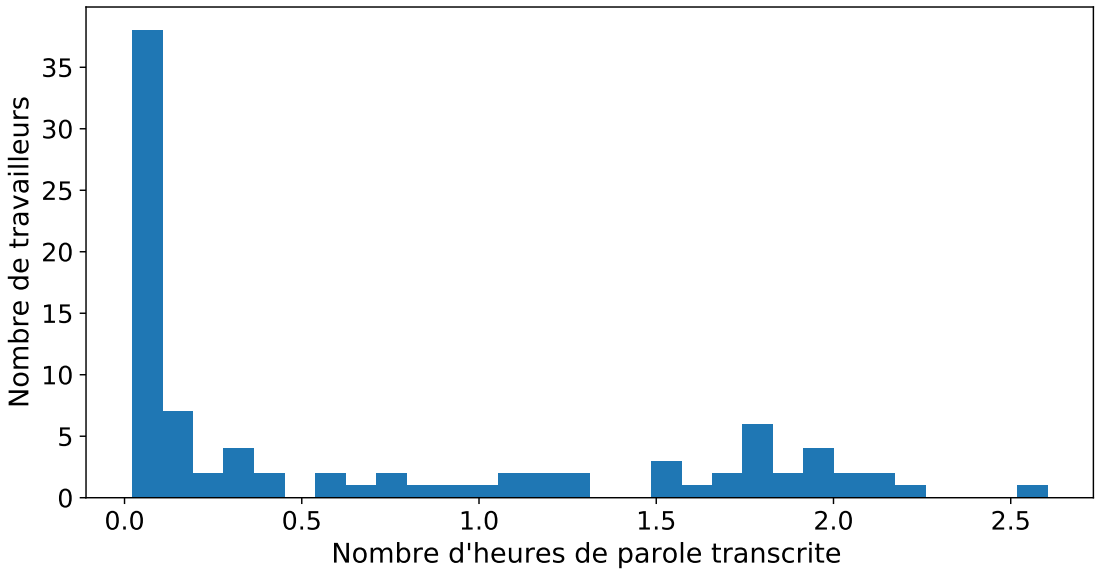


FIGURE 5.4. – Distribution du nombre d'heures de parole transcrite par les travailleurs.

Répartition des erreurs Par rapport aux transcriptions de référence de Librispeech, le LER est de 9,1% et le WER est de 15,97% sur l'ensemble des transcriptions collectées. Nous enlevons 58 transcriptions dont le LER est supérieur à 80%. Ces transcriptions sont pour la plupart vides, probablement à cause d'erreurs de manipulation avec l'interface. Un travailleur nous l'a confirmé, le bouton *Submit* est trop proche du champ de texte

dans l'interface Web. La Fig. 5.5 montre la distribution du LER pour toutes les transcriptions collectées. La médiane est de 5,32% avec le premier et troisième quartile à 1,53% et 12,28%. L'exercice de transcription est difficile et nous trouvons que les travailleurs fournissent un travail de bonne qualité.

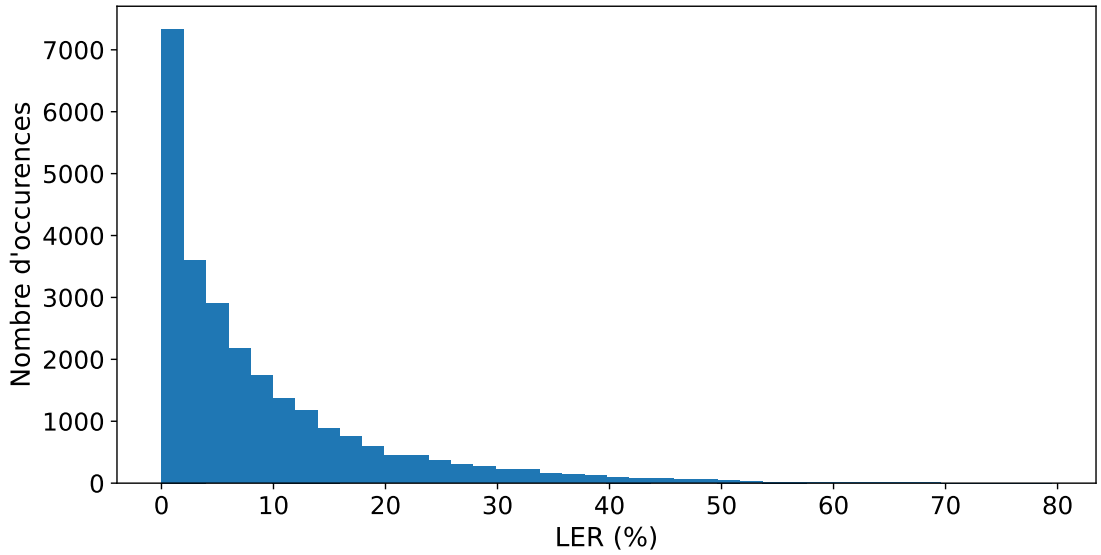


FIGURE 5.5. – Distribution du LER pour l'ensemble des transcriptions collectées.

Répartion des erreurs par travailleur La Fig. 5.6 sépare l'analyse du LER pour chacun des 31 travailleurs les plus productifs. Nous pouvons observer que la qualité des transcriptions est variable d'un travailleur à un autre. 15 travailleurs ont un LER médian inférieur à 5,2%. 13 travailleurs ont un LER médian entre 5,2% et 8,5%. Les 3 travailleurs les moins bons ont des LER médians de 11,1%, 18,5% et 23,4%. Nous pouvons remarquer que les distributions de LER sont de plus en plus étalées pour des travailleurs de moins en moins bons. Cette analyse suggère qu'il peut être pertinent de séparer la modélisation des erreurs de transcription par travailleur car la qualité des transcriptions est variable entre ces travailleurs.

Le Tab. 5.1 illustre également les différences entre les erreurs commises par les travailleurs. Il n'est pas évident de synthétiser cela mais nous proposons la méthode ci-dessous :

— Nous encodons les transcriptions de référence de Librispeech et les transcriptions des travailleurs avec un système de symboles d'un modèle unigramme (comme dans la partie 4.2). Nous choisissons la taille de ce système de symboles assez grande pour que celui-ci capture des morceaux de mots intéressants. Nous ne la choisissons pas trop grande non plus car alors la fréquence d'apparition des symboles devient trop faible. Nous réglons cette taille à 800.

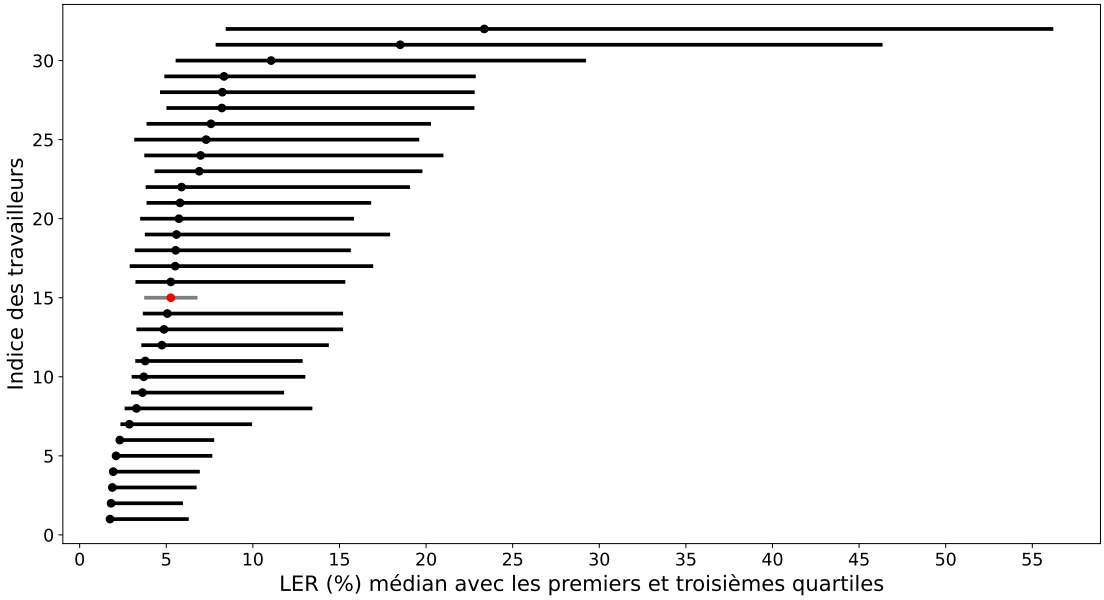


FIGURE 5.6. – Distribution du LER pour les 31 travailleurs les plus productifs. Nous indiquons la médiane ainsi que les premiers et troisièmes quartiles du LER. Nous reportons également en gris et rouge la médiane et les quartiles pour toutes les transcriptions.

— Nous déterminons l'ensemble minimal des opérations de substitution, d'insertion et de délétion pour transformer les transcriptions de référence en celles des travailleurs.

— Nous comptons le nombre d'occurrences de chaque erreur sur les symboles. Pour classer les erreurs, nous normalisons le nombre d'occurrences des erreurs de symbole par le nombre de fois où apparaissent ces symboles dans les transcriptions de référence. Nous ne classons que les erreurs qui apparaissent au moins 3 fois.

À partir de la Tab. 5.1, nous pouvons remarquer que :

— Les délétions représentent une part importante des erreurs. Pour beaucoup de travailleurs, au moins la moitié des erreurs sont des délétions. Par exemple, le travailleur 23 a 10,6% de lettres mal placées dans ses transcriptions dont 6,2% sont dues à des délétions. Les travailleurs ont donc tendance à ne pas transcrire certaines parties.

— Les insertions sont plus rares, elles représentent une part minime du total des erreurs.

— Certaines erreurs sont communes à des travailleurs comme les substitutions de *women* et *men* par *woman* et *man*. Nous avons aussi la délétion de mots *you*.

— Certaines semblent être plus spécifiques comme le remplacement de *quite* par *quiet* du travailleur 30 ou de *emperor* par *or* du travailleur 21.

Nous savions que le nombre d'erreurs est variable en fonction du du travailleur. Le Tab. 5.1 suggère que les erreurs sont également différentes d'un travailleur à un autre.

Exemples de transcriptions collectées Le Tab. 5.2 présente une comparaison de 15 transcriptions collectées choisies pour illustrer les erreurs commises par les travailleurs. Ci-dessous nous énumérons les catégories d'erreurs que nous avons identifiées :

— **Suppression de mots.** Les transcriptions 2, 8, 9, 12 et 13 illustrent des exemples où un mot au milieu de la phrase n'a pas été transcrit par le travailleur. Pour les transcriptions 6, 7 et 10, il manque aussi des mots mais en fin de phrase. La contrainte de temps de l'expérience explique ces erreurs.

— **Insertion de mots.** Cette erreur est beaucoup plus rare. Le mot *the* est en trop pour la transcription 5 tout comme le suffixe *n't* de la transcription 9.

— **Remplacement de mots.** Cette erreur est très courante. Nous avons par exemple *berries* à la place de *fairies* dans la transcription 1, *thirty* à la place de *twenty* dans la transcription 2, *office* pour *author's* dans la transcription 3, *please off your* pour *placed a few* dans la transcription 4, *the beleives* pour *her bilious* dans la transcription 5, *sometimes in a tough period* pour *symptoms of a thaw appeared* dans la transcription 6, *great* pour *grave* dans la transcription 7, *remember* pour *murmured* dans la transcription 9 et *white* pour *light* dans la transcription 9.

— **Fusion de mots.** Parfois le travailleur fusionne des mots comme avec *pertcgers breakand* pour *pictures bric a brac* dans la transcription 2 ou *delivering* pour *a living* dans la transcription 7.

— **Scission de mots.** Nous avons aussi l'inverse avec *he had speck* pour *the aspect* dans la transcription 7, *stand in the way* pour *send d'artagnan away* dans la transcription 8 et *entered and* pour *entering* dans la transcription 13.

— **Inversion de mots.** Nous avons un exemple de mots inversés avec *her cried peter* pour *here peter she cried* dans la transcription 15.

— **Problème de contexte.** Les travailleurs n'ont aucun contexte sur les phrases à transcrire. Pour chaque transcription, c'est un nouveau locuteur qui parle d'une nouvelle notion. Parfois le vocabulaire est soutenu car les exemples proviennent de la littérature. Certaines erreurs n'auraient pas été faites avec une continuité dans les exemples à transcrire. Nous avons par exemple *carpets* pour *copecks* et *rubble* pour *rouble* dans la transcription 14, ou encore *pills* à la place de *pukes* dans la transcription 5. De nombreux autres exemples contiennent aussi des noms propres difficiles à transcrire sans contexte.

— **Faux problème.** Certaines erreurs sont moins graves que d'autres pour un apprentissage d'un modèle acoustique car phonétiquement le travailleur a donné une bonne réponse. Nous avons par exemple *tonight* pour *to night* dans la transcription 11 et *silvy* pour *sylvie* dans la transcription 12.

— **Vrai problème.** Enfin, il arrive que le travailleur propose du texte sans rapport avec l'audio. Nous avons un exemple dans la transcription 10 avec *pressurizing per g*

LER	Substitutions				Délétions				Insertions				Index
25,9	6,6	whatever| → ever|	woman| → women|	quite| → it|	17,3	leaves|	there	hu	2,0	go	au	o	31
20,3	7,0	quite| → quiet|	toward| → towards|	will| → would|	10,8	count	au	sir|	2,4	go	od	wa	30
13,1	6,4	heard| → had|	men| → man|	here| → ear|	4,3	au	est	you	2,4	there	he	we	29
9,9	4,2	says| → said|	shall| → should|	seemed| → m|	4,2	wor	gu	thou|	1,5	go	ca	you	28
11,6	3,8	women| → woman|	nor| → no|	every| → y	6,3	’	several|	captain	1,6	go	m	ne	27
12,2	4,6	ours| → s|	will| → would|	may| → y	5,0	ations|	wor	off|	2,5	we	ma	may|	26
9,9	3,5	woman| → women|	every| → y	too| → to|	4,7	j	’	wh	1,6	you	od	mis	25
9,0	4,1	meet| → t|	its| → s|	used| → use|	3,1	he	th	au	1,8	mis	’	it	24
10,6	3,1	doctor| → |	there → there|	its| → s|	6,2	mor	you	’	1,3	it	pla	m	23
9,6	3,0	quite| → quiet|	ours| → s|	am| → m|	5,2	va	da	uff	1,4	thousand|	there	it	22
10,0	3,8	emperor → or	its| → s|	ards| → d|	4,9	there	course|	wil	1,3	it	au	you	21
8,2	2,7	towards| → toward|	whom| → who|	every| → y	4,3	go	besides|	wil	1,2	you	go	m	20
7,4	3,0	its| → s|	closed| → close|	ans| → s|	3,0	you	’	he	1,4	mis	it	ver	19
8,4	2,7	da → ad	whom| → who|	gra → re	4,0	he	th	an	1,6	go	po	m	18
7,0	3,4	am| → m|	ill| → l|	its| → s|	2,4	’	he	o	1,2	ver	we	m	17
8,0	3,3	there → there|	thou| → though|	women| → woman|	3,5	che	’	wi	1,2	wa	r	m	16
7,1	2,2	effect| → |	am| → m|	mis → m	3,9	man	wh	u	1,0	you	he	go	15
6,9	3,1	its| → s|	men| → man|	ill| → until|	2,2	thou|	fo	sa	1,6	you	m	it	14
7,5	2,7	ations| → ation|	re| → are|	any| → y	3,4	account|	you	ak	1,3	go	m	r	13
7,7	2,7	ll| → le|	men| → man|	their| → the|	3,8	asked|	ca	sa	1,2	m	we	he	12
7,2	2,2	towards| → toward|	am| → m|	upon| → on|	4,0	est	au	wor	1,1	m	r	ver	11
6,9	2,1	its| → s|	ious| → ous|	da → ad	3,8	du	sa	est	1,0	it	m	ver	10
6,1	2,4	doctor| → |	there → there|	every| → y	2,6	au	wi	o	1,0	m	r	dr	9
8,9	2,9	its| → s|	j → g	mis → r	1,5	there	you	j	4,5	hundred|	eight|	thousand|	8
5,3	2,4	ting| → bl	am| → m|	our| → or|	1,9	wil	fo	’	0,9	mis	bl	ju	7
4,0	1,8	ours| → s|	woman| → women|	our| → or|	1,4	ma	bi	wh	0,8	m	r	bu	6
3,8	1,7	its| → s|	ill| → l|	our| → or|	1,3	sa	au	bo	0,8	it	’	you	5
3,6	1,7	effect| → |	ill| → l|	da → ad	1,2	int	u	th	0,7	affect	he	m	4
3,8	1,5	ours| → s|	every| → y	men| → man|	1,5	au	sto	bi	0,7	m	you	r	3
2,8	1,2	towards| → toward|	thou| → though|	doctor| → |	1,0	au	every|	’	0,6	m	dr	r	2
3,2	1,6	men| → man|	re| → er|	some| → me	1,0	step	au	ma	0,7	you	m	r	1

TABLEAU 5.1. – Erreurs les plus communes pour les 31 travailleurs les plus productifs. La première colonne indique le LER moyen de chaque travailleur entre les transcriptions collectées et la référence de Librispeech. Les trois autres colonnes séparent les erreurs selon la catégorie : substitution, insertion ou délétion. Nous indiquons les 3 erreurs les plus communes pour chaque travailleur et pour chaque catégorie. Les nombres à gauche de chacune de ces 3 colonnes indiquent les taux de substitution, de délétion et d’insertion de lettres. La somme de ces taux est égale au LER. Nous classons les travailleurs dans le même ordre que dans la Fig. 5.6. Le symbole « | » représente l’espace entre les mots.

à la place de *fresh arrivals the background*. Cela est probablement dû à un manque de temps.

5.3.2. Apprentissage de modèles acoustiques

Nous menons une expérience d'apprentissage de modèles acoustiques. Nous comparons le LER atteint en utilisant le jeu de données de référence ou les transcriptions des travailleurs pour observer l'impact des erreurs dans les transcriptions sur la qualité du modèle acoustique.

Le modèle acoustique Nous utilisons le même modèle acoustique que dans la partie 4.4.1 avec l'architecture « Transformer ». Celle-ci a été choisie en fonction du LER atteint sur *train-clean-100*. Nous n'utilisons pas de couche SpecAugment et gardons le même pas d'apprentissage pendant 400 000 itérations. Pour l'évaluation, nous utilisons un jeu de données de 4 000 transcriptions (2 paquets de 2 000 fichiers) non utilisées pendant l'expérience de collecte. Nous reportons le LER et le WER atteint sur ce jeu de données sans utilisation d'un modèle de langage.

Apprentissage sur toutes les transcriptions L'apprentissage du modèle acoustique sur toutes les transcriptions collectées permet d'atteindre un LER de 10,29% et un WER de 25,46% sur le jeu de données d'évaluation. En utilisant les transcriptions de référence nous atteignons un LER de 7,03% et un WER de 18,22%. Il y a donc bien une dégradation de performance du modèle acoustique avec des transcriptions erronées.

Filtrage des transcriptions erronées Nous menons la même expérience en enlevant au fur et à mesure les transcriptions contenant le plus d'erreurs sans prendre en compte le travailleur. Cela a pour conséquence de réduire le nombre d'heures de parole dans le jeu de données d'apprentissage. Nous testons plusieurs situations pour des niveaux de filtre différents des transcriptions erronées. Nous construisons plusieurs jeux de données dont les LER des transcriptions sont au maximum de 30, 20, 15, 10, 8, 6, 4 et 2%. Ces jeux de données ont un nombre d'heures de parole qui varie entre 60,23 et 17,67. Pour chaque situation, nous reportons le LER atteint par le modèle acoustique sur le jeu de données d'évaluation. Nous effectuons également l'apprentissage sur les transcriptions de référence. La Fig. 5.7 reporte les résultats de cette expérience.

Pour des apprentissages sur les transcriptions de référence, nous observons un phénomène attendu : plus le jeu de données d'apprentissage est grand, meilleur est le LER. Pour les transcriptions collectées, cela n'est plus valable. La première observation est que les 18 minutes de parole dont les transcriptions ont un LER supérieur à 30% affectent négativement le LER du modèle acoustique. Celui-ci se détériore de 9,99% à 10,29%. Le meilleur modèle est celui qui conserve uniquement les transcriptions avec un LER inférieur à 15% pour atteindre un LER du modèle acoustique de 9,84%.

LER	Transcriptions de référence et transcriptions collectées	Index
15,0	the undertow fairies took freddy and swam with him way out to the roller fairies the underto**ne va**ries **looked rea**dy and swam with him way out to the roller **be**rries	1
26,6	pictures bric a brac and other things to the tune of twenty thousand dollars more were removed **pertcgers breakand** other things @ the tune of **thirty** thousand dollars @ were removed	2
14,3	the author's whole argument amounts to this that every opinion which differs from the code of dogmas we believe in at a given time is heresy the **office** whole argument amounts to this that every opinion which differs from **a** cod**ed amount** we believe in at a given time is heresy	3
16,3	placed a few feet from the entrance this arrangement has the merit of keeping the house warm **please off your** feet from the entrance this arrangement has the **marriage** of keeping the house warm	4
20,0	her bilious mind and scandalously muttered that in the case of patients having money the pills were sugar and the pukes were honey **the beleives** mind and **the** scandaloulsy **noted** that in the case of patients having money the pills **of** sugar and the **pills** are honey	5
32,6	and symptoms of a thaw appeared the ice began to crack here and there and jets of salt water were thrown up like fountains in an english park and **sometimes in** a **tough period** the ice began to crack here and there and **jolts** of salt water were thrown up in the **@@@@@@**	6
16,0	the general was followed by his veterans and the aspect of a grave magistrate was a living lesson to the multitude a new method of secret ballot the general was followed by his **patrons** and **he had speck** of a **great** magistrate was **delivering** lesson to the multitude a new method of secret **@**	7
19,6	this was a matter for grave consideration at all events the moment was badly chosen to send d'artagnan away this was a matter for grave consideration in all event**@** the moment was **@** chosen to **stand in the way**	8
17,7	only that would be a joke and i know i should spoil it very near though murmured the carrier with a chuckle very near the stranger only that **will** be a joke **@** i know i should**n't** spoil it very near though **remember** the carrier with **the tickle** very near the stranger	9
19,4	silently and without confusion during the past few minutes their numbers were increasing swiftly fresh arrivals packing the background silently and without confusion during the past few minutes their numbers were increasing swiftly **pressurizing per g @@**	10
7,7	the fourth day in succession without wind but overcast light snow has fallen during the day to night the wind comes from the north the fourth day in succession without wind but overcast **white** snow **is** fall**ing** during the day **tonight** the wind comes from the north	11
5,3	and he threw his arms round her neck for this novel but apparently not very painful operation it's very like kissing sylvie remarked and he threw his arms round her neck for this novel but apparently **@** very painful operation it's very like kissing **silvy** remarked	12
13,3	who acted as spokesman had on entering asked the landlord if they could sleep there who acted as **spokemen** had on **entered and** asked the **@**lord if they could sleep there	13
11,2	that makes thirty five copecks altogether so i must give you a rouble and fifteen copecks that makes thirty five **carpets** altogether so i must give you a **rubble** and fifteen **carpets**	14
10,1	they like here peter she cried they like here so much that if they can find a place to suit them for a nest they're going to stay they like **her cried peter** they like her**@** so much that if they can find a place to suit them for a nest they're going to stay	15

TABLEAU 5.2. – Exemple de 15 transcriptions fournies par 3 travailleurs. Les 10 premières transcriptions sont produites par 2 des plus mauvais travailleurs (le 29 et le 30). Les 5 dernières sont produites par un travailleur dans la moyenne (le 15). Pour chaque exemple nous montrons la transcription de référence puis celle du travailleur ainsi que le LER entre les deux. En gras nous indiquons les erreurs et le symbole @ indique qu'il manque un mot ou un morceau de mot.

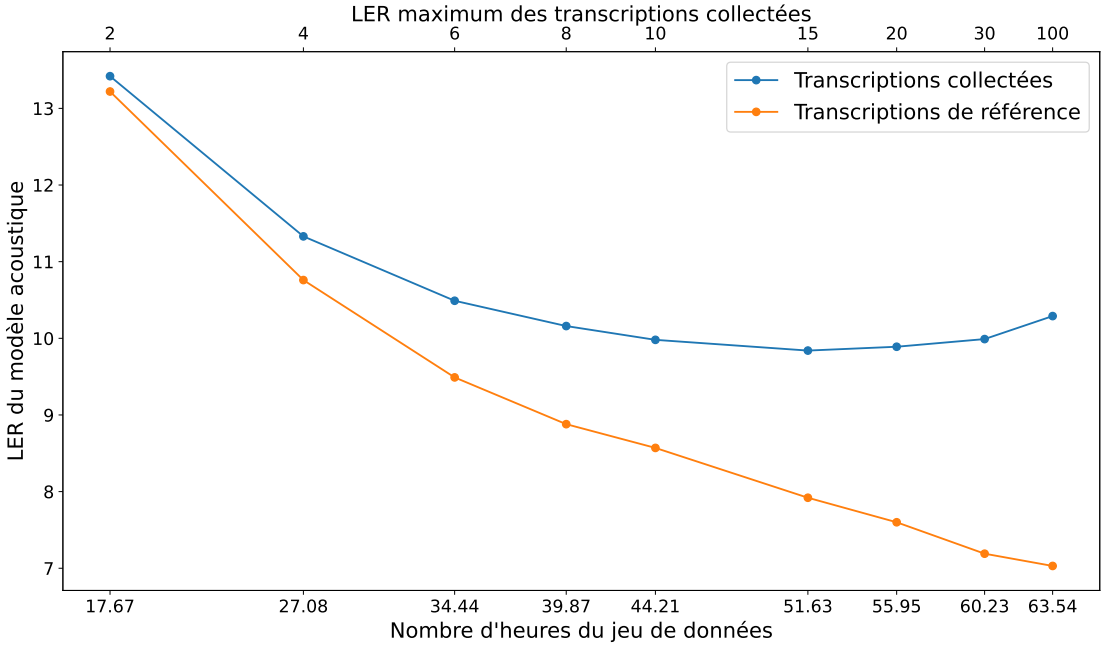


FIGURE 5.7. – LER atteint par le modèle acoustique pour des jeux de données de différentes qualité.

Nous observons globalement une stagnation des performances du modèle acoustique si l'on utilise les transcriptions les plus mauvaises. 34,44 heures de transcriptions avec un LER inférieur à 6% suffisent pour atteindre un LER sur le modèle acoustique de 10,49%. Si aucune méthode de correction des transcriptions n'est utilisée, il semble qu'il ne soit pas rentable d'embaucher les travailleurs qui font le plus d'erreurs. Pour vérifier cela, nous filtrons les travailleurs selon leur LER moyen. Nous enlevons au fur et à mesure les transcriptions des travailleurs ayant le LER moyen le plus élevé. Ainsi nous créons différents jeux de données contenant les transcriptions des travailleurs dont le LER est en dessous de 25, 12, 9, 7, 6 et 5%. Ces jeux de données utilisent respectivement le travail de 84, 69, 54, 37, 24 et 16 personnes. La Fig. 5.8 reporte les résultats de cette expérience. Pour les filtres à 25 et 12%, nous obtenons un résultat similaire de modèle acoustique. Celui avec le filtre à 12% atteint un LER de 10,11%. Nous confirmons donc que sans mécanisme de modélisation des erreurs, il n'est pas utile d'embaucher les services d'un travailleur ayant un LER moyen de plus de 12%.

5.3.3. Conclusion

Dans cette partie, nous avons identifié les contraintes et les difficultés à effectuer une collecte de données avec une plateforme de *crowdsourcing*. Nous avons mis en évidence la nécessité de choisir avec discernement les travailleurs en utilisant une procédure de qualification. L'objectif premier de cette expérience est l'analyse des erreurs des trans-

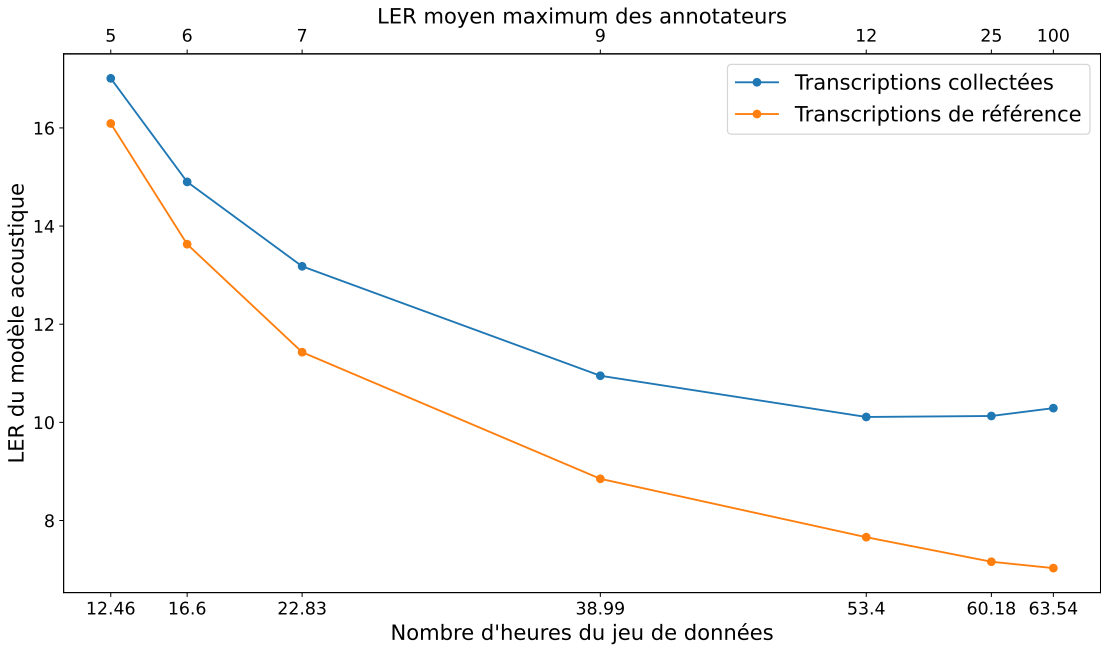


FIGURE 5.8. – LER atteint par le modèle acoustique en filtrant les travailleurs selon leur LER moyen par rapport aux transcriptions de référence.

criptions que nous avons collectées. Nous avons notamment observé les différences entre les erreurs pour différents travailleurs. Ils n'ont pas les mêmes compétences : la quantité d'erreurs varie d'un travailleur à un autre. A compétence égale, ils ne commettent pas non plus les mêmes erreurs de transcription. Cependant, pour la plupart des travailleurs, nous observons un fort taux de délétion et un faible taux d'insertion. Il est peut-être intéressant de ne pas modéliser les insertions pour la conception d'un modèle de bruit de transcription. Pratap et al. [2022] proposent d'ailleurs une méthode utilisant des wFST et qui ne modélise que les délétions.

De plus, nous avons conduit des expériences d'apprentissage de modèles acoustiques avec les transcriptions collectées. En comparant les LER obtenus en fonction de la quantité d'erreurs dans les transcriptions, nous avons mis en évidence la dégradation du modèle acoustique induite par ces erreurs. Nous avons aussi observé la nécessité de choisir des travailleurs qui font le moins d'erreurs possibles. En effet, ajouter les contributions des plus mauvais travailleurs n'a pas d'impact sur le LER du modèle acoustique.

Pour obtenir le meilleur modèle acoustique avec les moyens à disposition, il peut donc être intéressant de développer des algorithmes d'apprentissage prenant explicitement en compte les erreurs dans les transcriptions, comme nous l'avons fait dans les chapitres précédents. Ceci est une direction intéressante pour de futurs travaux.

6. Conclusion et perspectives

Cette thèse étudie l'impact des transcriptions bruitées sur l'apprentissage de modèles acoustiques. Pour mitiger cet impact, nous proposons des fonctions de coût qui utilisent une modélisation des erreurs de transcription. Dans ce chapitre, nous résumons les contributions de cette thèse dans la partie 6.1, nous synthétisons nos conclusions dans la partie 6.2 et proposons des axes de recherche possibles pour continuer le travail dans la partie 6.3.

6.1. Résumé

Le chapitre 3 propose une première solution au problème d'erreurs dans les transcriptions avec l'algorithme Lead2Gold. Pour évaluer cette approche, nous construisons des jeux de données synthétiques avec des substitutions, des insertions et des délétions de lettres. Nous proposons un modèle de bruit de transcriptions utilisant ces opérations de changement de lettres. Nous adaptons les fonctions de coûts classiques CTC et ASG pour inclure ce modèle de bruit. Pour une transcription bruitée d'un jeu de données, la fonction de coût de Lead2Gold formulée sans approximation prend en compte toutes les transcriptions possibles pouvant mener à celle-ci. Nous démontrons méthodiquement la validité mathématique de cette méthode en faisant un parallèle avec les algorithmes de programmation dynamique utilisés pour CTC et ASG. Mais cette première formulation a une complexité algorithmique exponentielle car nous prenons en compte toutes les possibilités de transcriptions. Pour remédier à cela, nous proposons de ne prendre en compte qu'un sous-ensemble de celles-ci avec une recherche en faisceau. L'approche est originale car la recherche en faisceau n'est habituellement pas utilisée dans une fonction de coût. Nous démontrons comment une telle méthode peut-être utilisée dans le cadre d'un apprentissage par descente de gradient d'un modèle acoustique. Avec les données synthétiques, nous montrons l'efficacité de Lead2Gold pour réduire l'impact des erreurs de transcriptions, notamment dans le cas où seules les substitutions sont prises en compte.

Le chapitre 4 propose une nouvelle approche d'implémentation de Lead2Gold. Nous utilisons pour cela des wFST. Ce sont des graphes pouvant se combiner entre eux et qui modélisent la transformation d'un ensemble de symboles en un autre ensemble de symboles. L'implémentation de Lead2Gold a le désavantage d'être peu adaptable car la recherche en faisceau dépend fortement des choix de modélisation postulés au départ. Nous proposons dans ce chapitre de s'affranchir de cette recherche en faisceau en la remplaçant par la composition de wFST. De nouvelles bibliothèques implémentent des algorithmes de composition afin d'utiliser les wFST dans une fonction de coût en prenant

en charge la phase de calcul des gradients qui est nécessaire à l'apprentissage d'un modèle acoustique. La formulation de Lead2Gold proposée dans ce chapitre est modulaire : elle permet d'inclure facilement de nouveaux composants comme un lexique ou un modèle de bruit qui opère sur des morceaux de mots et non des lettres. Nous justifions également notre approche mathématiquement. Notre méthode compose successivement plusieurs wFST et génère des graphes trop grands pour être utilisés en pratique dans une fonction de coût. Nous mettons en évidence la nécessité d'utiliser un algorithme de composition adapté pour élaguer les wFST de leurs composantes peu probables. Nous proposons enfin une méthode d'apprentissage pour estimer des modèles de bruit de transcription qui nécessite une version erronée des transcriptions et une version sans erreur.

Le chapitre 5 présente une expérience de collecte de transcriptions en conditions réelles. Nous utilisons une plateforme de *crowdsourcing* pour mettre en relation des travailleurs avec la tâche de transcription que nous proposons. Nous identifions les contraintes d'utilisation de la plateforme et proposons une procédure pour trouver les travailleurs les plus à même d'effectuer la tâche. L'objectif est d'analyser les erreurs de transcription commises par les travailleurs pendant l'expérience. C'est pourquoi nous imposons aux travailleurs deux contraintes : ils ont un temps limité pour chaque tâche et ne peuvent écouter l'enregistrement audio que deux fois. Nous mettons en évidence l'hétérogénéité des erreurs en fonction du travailleur et concluons à la nécessité de modéliser les erreurs de chaque travailleur de façon indépendante. Nous mesurons également l'impact des erreurs de transcription sur la qualité du modèle acoustique. Nous observons que si aucune méthode de correction des erreurs n'est utilisée, il est inutile d'utiliser les transcriptions des travailleurs commettant le plus d'erreurs.

6.2. Conclusion

Les algorithmes présentés dans cette thèse sont confrontés à plusieurs difficultés. Il y a premièrement une complexité algorithmique. L'algorithme de composition utilisé pour les wFST n'est pas adapté. Le modèle de bruit que nous avons conçu génère trop de transcriptions possibles. L'utilisation d'un lexique et l'élagage du modèle de bruit n'ont pas été suffisants pour réduire les possibilités et permettre aux opérations de composition d'être calculées en temps raisonnable, c'est-à-dire permettant d'être incorporées dans un apprentissage automatique.

Nous avons tout d'abord testé notre méthode avec des données synthétiques générées avec des opérations sur les lettres. Pour une utilisation avec un jeu de données collecté en conditions réelles, cette modélisation par lettre est trop simplificatrice. Il y a donc ici une deuxième difficulté. Il faut trouver une unité de texte adaptée à la modélisation des erreurs. Nous avons essayé avec des morceaux de mots mais il est difficile d'établir un critère permettant d'évaluer un modèle de bruit. Déterminer la structure du modèle de bruit est également un problème ouvert et intéressant. L'analyse des erreurs du chapitre 5 suggère par exemple qu'il n'est pas forcément nécessaire de modéliser les insertions.

L'estimation des probabilités de ce modèle de bruit nécessite dans notre méthode d'avoir à disposition un jeu de données contenant des transcriptions bruitées et des transcriptions sans erreur. C'est une limitation car un tel jeu de données est difficile à obtenir en pratique. Une méthode non-supervisée ne nécessitant pas les transcriptions sans erreur pourrait peut-être résoudre ce problème.

De plus, nous avons observé un fort impact des erreurs de transcriptions, collectées en conditions réelles, sur la qualité du modèle acoustique. Notre étude suggère qu'il est important de séparer la modélisation des erreurs en fonction des travailleurs les ayant produites. Avec ce constat, il devient encore plus nécessaire de trouver une bonne méthode d'apprentissage des modèles de bruit car la quantité de données disponible pour cet apprentissage est plus limitée si l'on sépare par travailleur.

Les difficultés rencontrées ne remettent pas en question la pertinence de l'intégration d'un modèle de bruit de transcription dans une fonction de coût. En effet, nous avons montré la faisabilité de cette méthode avec la première formulation de Lead2Gold dans le chapitre 3. Pour la formulation avec des wFST du chapitre 4 nous avons bien identifié où sont les problèmes et comment les résoudre, notamment avec l'utilisation d'une méthode de composition plus adaptée.

6.3. Perspectives

Concernant les perspectives de notre méthode, nous avons déjà avancé comment répondre aux difficultés. Nous présentons ici deux axes de recherche auxquels nous souhaitons nous intéresser.

6.3.1. La collecte des transcriptions est-elle vraiment une bonne approche ?

Lors de notre expérience de collecte de données, nous avons observé la difficulté et le coût financier à faire transcrire par des travailleurs 63 heures d'enregistrements audio. Les modèles acoustiques commerciaux utilisent des dizaines de milliers d'heures d'audio et de transcriptions pour les langues les plus parlées. Il n'est pas envisageable d'utiliser autant de données pour les 7 000 langues parlées dans le monde. Il y a donc un réel déséquilibre d'accès à la technologie de reconnaissance vocale.

Des travaux proposent de meilleures méthodes de collecte de transcriptions. L'apprentissage actif est une technique qui consiste à identifier quels doivent être les enregistrements audio à transcrire en priorité pour améliorer au maximum un modèle acoustique. Combinée avec une méthode semi-supervisée, cette approche peut réduire les coûts de transcription de 70% [Drugman et al., 2016, Yu et al., 2010].

Mais ce paradigme d'utiliser des transcriptions collectées n'est pas forcément le seul. La technique dite de « pseudo-transcription » dans un contexte d'apprentissage semi-supervisé [Kahn et al., 2020, Likhomanenko et al., 2020, Xu et al., 2020] consiste à utiliser un modèle acoustique initial appris avec un petit jeu de données classique contenant des transcriptions produites par des humains. Ensuite, ce modèle acoustique est utilisé pour produire des pseudo-transcriptions d'un grand jeu de données contenant uniquement les enregistrements audio. Les pseudos-transcriptions sont souvent générées à l'aide d'un modèle de langage. Ensuite, elles sont utilisées pour raffiner le modèle acoustique. On peut alors recommencer le cycle et générer de meilleures pseudo-transcriptions. Nous avons fait des expériences préliminaires avec Lead2Gold pour modéliser les erreurs des pseudo-transcriptions et permettre d'améliorer un peu plus le modèle acoustique à chaque étape.

6.3.2. Les transcriptions sont-elles vraiment nécessaires ?

L'utilisation de transcriptions pour l'apprentissage d'un modèle acoustique semble assez naturelle. Après tout, c'est l'objectif final : à partir d'un signal acoustique nous voulons trouver la transcription correspondante. Mais ce n'est pas parce que l'objectif est d'obtenir une transcription que l'on doit forcément en utiliser pour l'apprentissage d'un modèle acoustique. Les transcriptions sont-elles finalement un bon support de l'information acoustique ? En tant qu'êtres humains, nous savons lire une transcription et émettre le son correspondant. Mais si nous pouvons le faire c'est parce que nous avons appris toutes les règles permettant de combiner des lettres et des mots pour connaître la prononciation. D'ailleurs, il suffit de changer de langue et les règles peuvent changer. Elles peuvent changer un peu, si l'on compare par exemple l'anglais et le français. Mais elles peuvent être aussi complètement différentes, par exemple dans le cas d'un autre système de représentation de l'écriture comme les hiragana et katakana japonais. Un système d'écriture est d'ailleurs peut-être meilleur qu'un autre pour l'apprentissage d'un modèle acoustique. La méthode pour contourner ce problème a longtemps été d'utiliser un système commun d'écriture en transformant toutes les transcriptions en phonèmes : ce sont des symboles qui représentent uniquement le son. L'alphabet phonétique international (IPA) a été créé « à la main » par des phonéticiens français et anglais pour essayer de couvrir tous les sons produits dans toutes les langues. Les règles de lecture sont alors encodées dans un lexique qui contient toutes les prononciations possibles d'un mot. Mais cette étape de transformation en phonèmes a aussi un coût : les erreurs commises lors de la transformation en phonèmes réduisent la qualité du modèle acoustique. Dans tous les cas, nous imposons au modèle acoustique de comprendre les règles d'écriture des transcriptions et cela n'est en soi pas utile pour modéliser l'acoustique.

L'apprentissage auto-supervisé est une technique émergente permettant de ne pas utiliser de transcriptions. Avec cette technique, les travaux de Schneider et al. [2019] et de Baevski et al. [2020] s'affranchissent des transcriptions pour le pré-apprentissage de modèles acoustiques. À partir d'une représentation initiale d'un morceau de signal audio,

la méthode employée est de donner pour objectif au modèle acoustique de prédire la représentation des trames suivantes. Une fois le modèle pré-appris, une étape d'ajustement est nécessaire en utilisant un petit jeu de données de signaux audio et de transcriptions (10 minutes peuvent suffire). La méthode de Baevski et al. [2021] effectue cet ajustement de façon totalement non-supervisée et ne nécessite aucune transcription.

6.3.3. Le mot de la fin

La RAP a le potentiel de réduire drastiquement les barrières linguistiques. Combinée avec un modèle de traduction, elle pourrait fournir un moyen de communication efficace entre tous et pour n'importe quelle langue. Contrairement à ce qui est souvent annoncé, la RAP est encore loin de tenir toutes ces promesses. Pour l'instant, son champ d'application est limité à un nombre restreint de langues et de situations. C'est à nous, concepteurs de modèles acoustiques, d'aller au-delà des paradigmes habituels pour enfin répondre à ces promesses. C'est ce que nous avons fait dans cette thèse et c'est pourquoi nous espérons avoir contribué au domaine de la RAP pour atteindre cet objectif.

A. La fonction logadd

Définition La fonction logadd est aussi appelée « Log-Sum-Exp » ou fonction LSE. Nous l'utilisons car c'est l'opération $\oplus$ du demi-anneau Log utilisée pour les poids des wFST. Soit $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^{\mathbb{n}}$ un ensemble de scalaires. La fonction logadd est une fonction scalaire définie par :

$$\begin{array}{rccc} \text{logadd} & : & \mathbb{R}^{\mathbb{n}} & \to & \mathbb{R} \\ & & X & \mapsto & \log\left(e^{x_1} + \cdots + e^{x_n}\right) \end{array} . \tag{A.1}$$

On peut écrire

$$\text{logadd}(x_1, x_2, \ldots, x_n) = \log\left(e^{x_1} + \cdots + e^{x_n}\right). \tag{A.2}$$

Nous pourrons aussi utiliser une autre notation. Avec $g : \mathbb{R} \to \mathbb{R}$ une fonction arbitraire. On note :

$$\underset{x \in X}{\text{logadd}}\ f(x) = \log\left(e^{f(x_1)} + \cdots + e^{f(x_n)}\right) = \log \sum_{i=1}^{n} e^{f(x_i)}. \tag{A.3}$$

Nous pouvons aussi combiner deux fonctions logadd. Avec $f : \mathbb{R}^2 \to \mathbb{R}$ et $Y \in \mathbb{R}^{\mathbb{m}}$:

$$\underset{x \in X}{\text{logadd}}\ \underset{y \in Y}{\text{logadd}}\ g(x, y) = \log \sum_{i=1}^{n} \exp\left[\log \sum_{j=1}^{m} e^{g(x_i, y_j)}\right] \tag{A.4}$$

$$= \log \sum_{i=1}^{n} \sum_{j=1}^{m} e^{g(x_i, y_j)} \tag{A.5}$$

$$= \log \sum_{\substack{i \in \{1 \ldots n\} \\ j \in \{1 \ldots m\}}} e^{g(x_i, y_j)} \tag{A.6}$$

$$= \underset{\substack{x \in X \\ y \in Y}}{\text{logadd}}\ g(x, y) \tag{A.7}$$

Propriété 1 En plus d'être commutative et associative, la fonction logadd a des propriétés intéressantes. Pendant l'exécution d'un algorithme nous n'avons pas forcément accès à tout l'ensemble X sur lequel nous voulons appliquer logadd. Nous pouvons agréger les scores au fur et à mesure :

$$\begin{aligned}
\text{logadd}(\text{logadd}(x_1, x_2), x_3) &= \log(e^{\text{logadd}(x_1,x_2)} + e^{x_3}) && \text{(A.8)}\\
&= \log(e^{\log(e^{x_1}+e^{x_2})} + e^{x_3}) && \text{(A.9)}\\
&= \log(e^{x_1} + e^{x_2} + e^{x_3}) && \text{(A.10)}\\
&= \text{logadd}(x_1, x_2, x_3). && \text{(A.11)}
\end{aligned}$$

Propriété 2 Soit $C \in \mathbb{R}$ une constante. Nous utiliserons souvent la propriété suivante :

$$\begin{aligned}
\underset{x\in X}{\text{logadd}}\ [C + f(x)] &= \log \sum_{i=1}^{n} e^{f(x_i)+C} && \text{(A.12)}\\
&= \log \sum_{i=1}^{n} e^{C} e^{f(x_i)} && \text{(A.13)}\\
&= \log e^{C} \sum_{i=1}^{n} e^{f(x_i)} && \text{(A.14)}\\
&= \log e^{C} + \log \sum_{i=1}^{n} e^{f(x_i)} && \text{(A.15)}\\
&= C + \underset{x\in X}{\text{logadd}}\ f(x) && \text{(A.16)}
\end{aligned}$$

Dérivé partielle La fonction logadd est une fonction scalaire dont on peut calculer ses dérivés partielles par rapport à chacun des x_i de X :

$$\frac{\partial\ \text{logadd}}{\partial x_i}(X) = \frac{e^{x_i}}{\sum_j e^{x_j}}. \qquad \text{(A.17)}$$

Implémentation Pour implémenter la fonction logadd correctement il faut faire attention aux problèmes liés à la représentation des nombres flottants en machine. Les nombres flottants ont une précision limitée et ne peuvent pas être trop grands. Cela est appelé l'*underflow* et l'*overflow*. Soit $C \in \mathbb{R}$, avec la propriété 2 nous avons :

$$\begin{aligned}
\text{logadd}(X) &= C - C + \log\left(e^{x_1} + \cdots + e^{x_n}\right) && \text{(A.18)}\\
&= C + \log\left(e^{x_1-C} + \cdots + e^{x_n-C}\right). && \text{(A.19)}
\end{aligned}$$

Pour éviter d'utiliser la fonction exponentielle sur un nombre trop grand, on remplace la constante C par :

$$c = \max\{x_1, \ldots, x_n\}. \qquad \text{(A.20)}$$

B. Exemple d'utilisation des wFST différentiables

La Fig. B.1 présente un exemple complet de l'opération $y = \text{Score}(A_1 \circ A_2)$. Le but est de calculer y puis ses dérivés partielles par rapport aux poids de A_1 et de A_2.

La fonction de transformation des poids pour la composition $b = f_2(a)$ agrège avec l'opération $\otimes$ (l'addition dans le demi-anneau Log) un poids de A_1 noté a_j et un poids de A_2 noté a_k pour obtenir chaque b_i. a_j et a_k sont portés par les transitions t_j et t_k. b_i est porté par la transition t_i. Donc pour tous les appariements $t_i = (t_j, t_k)$ du résultat de la composition on a

$$b_i = a_j + a_k \tag{B.1}$$

$$J_{f_2}(i,j) = \frac{\partial b_i}{\partial a_j} = 1,\ J_{f_2}(i,k) = \frac{\partial b_i}{\partial a_k} = 1 \text{ et } J_{f_2}(i,l) = 0 \text{ si } l \notin \{j,k\}. \tag{B.2}$$

De même, la fonction Score(.) produisant le poids $y = f_1(b)$ peut être écrite

$$y = \log \sum_{\pi \in B} \exp \left(\sum_{k=1}^{n} b_j \right). \tag{B.3}$$

Donc pour le calcul de $\nabla_y = J_y^T$, on a

$$\nabla_y(j) = \frac{\partial y}{\partial b_j} \tag{B.4}$$

$$= \frac{1}{\exp(y)} \frac{\partial}{\partial b_j} \sum_{\pi \in B} \exp\left(\sum_{k=1}^{n} b_k\right) \tag{B.5}$$

$$= \frac{1}{\exp(y)} \sum_{\pi \in B} \frac{\partial}{\partial b_j} \exp\left(\sum_{k=1}^{n} b_k\right) \tag{B.6}$$

$$= \frac{1}{\exp(y)} \sum_{\pi \in B} \exp\left(\sum_{k=1}^{n} b_k\right) \frac{\partial}{\partial b_j} \sum_{k=1}^{n} b_k \tag{B.7}$$

$$= \frac{1}{\exp(y)} \sum_{\pi \in B} \exp\left(\sum_{k=1}^{n} b_k\right) \mathbb{1}_{b_j \in \pi} \tag{B.8}$$

$$= \frac{1}{\exp(y)} \sum_{\substack{\pi \in B \\ b_j \in \pi}} \exp\left(\sum_{k=1}^{n} b_k\right). \tag{B.9}$$

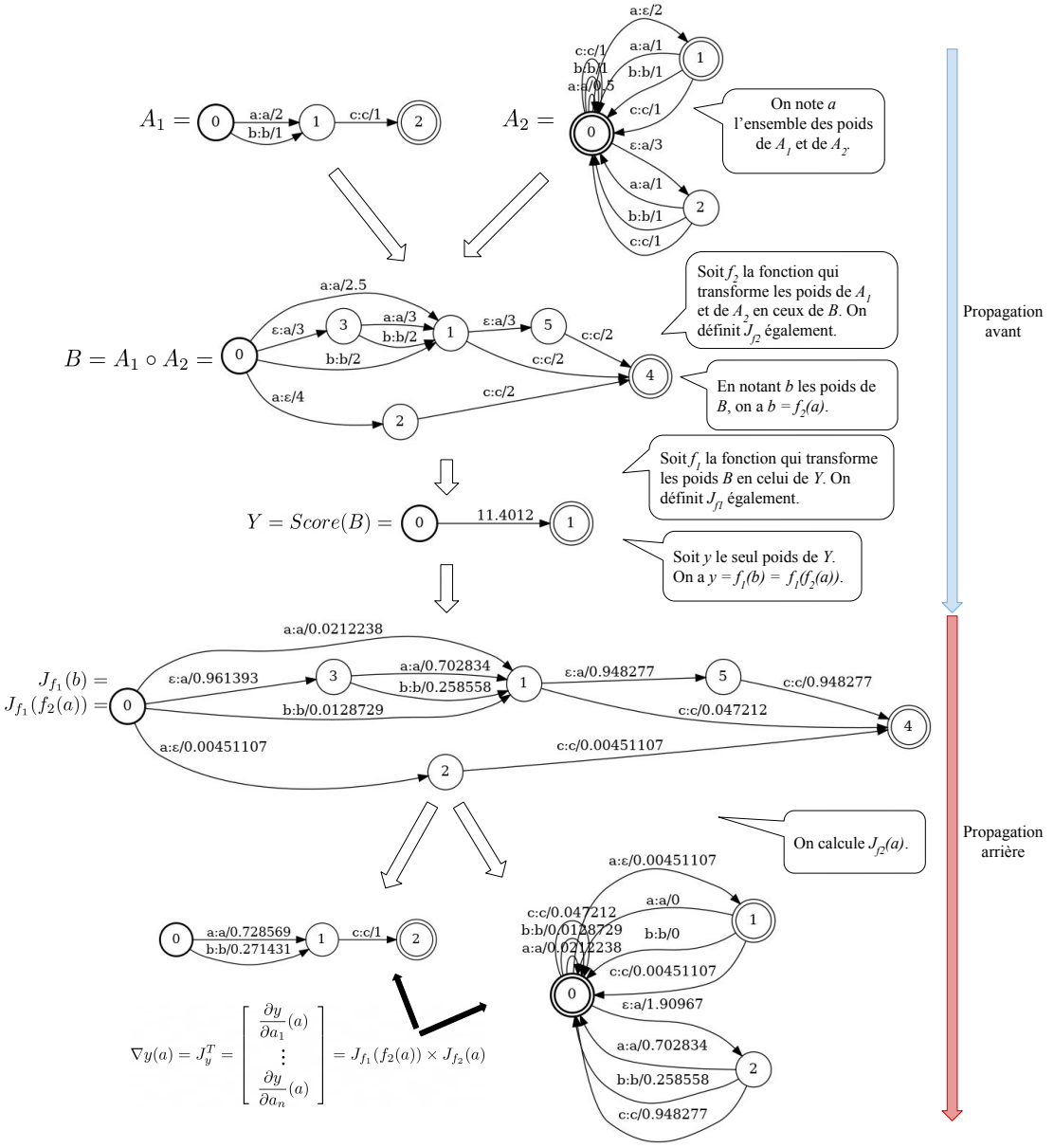


FIGURE B.1. – Exemple de calcul de gradient sur des wFST. On définit deux wFST A_1 et A_2 puis on calcule $y = \text{Score}(A_1 \circ A_2)$ pendant la phase de propagation avant. Pendant la phase de propagation arrière, on propage le gradient avec le théorème de dérivation des fonctions composées jusqu'à obtenir le gradient de y par rapport aux poids de A_1 et A_2.

C. Observation de $R(t)$

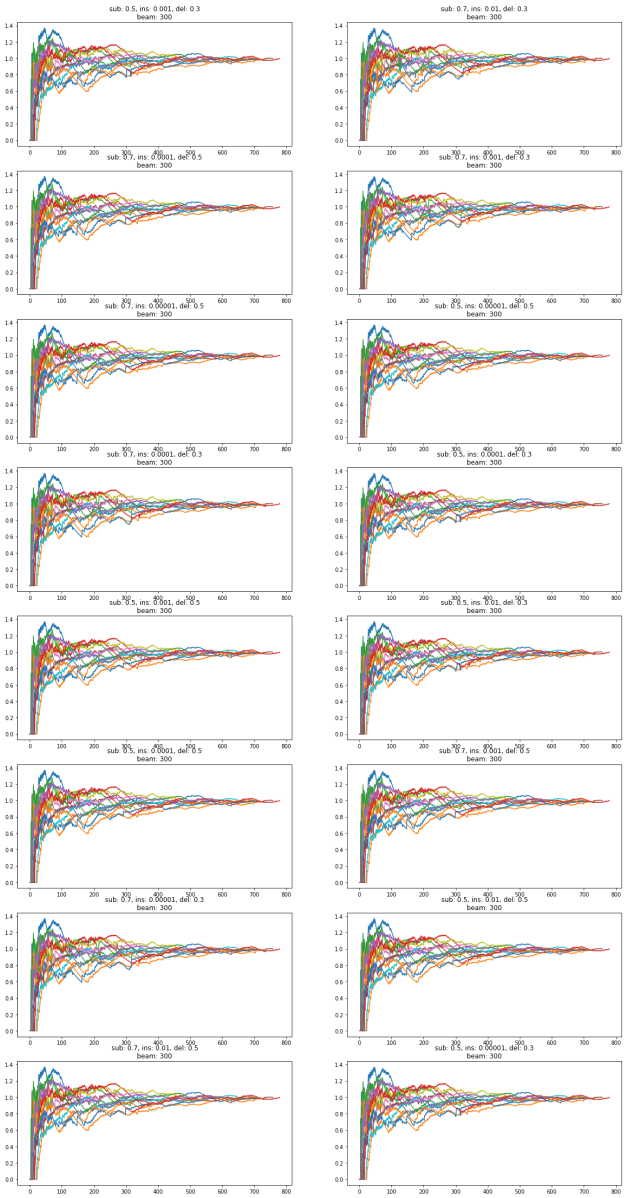


FIGURE C.1. – Observation de $R(t)$ en fonction des trois paramètres α_{bruit} choisis pour les transformations de substitution, d'insertion et de délétion.

Bibliographie

Allauzen, C., Riley, M., and Schalkwyk, J. (2009). A generalized composition algorithm for weighted finite-state transducers. In *Interspeech*, pages 1203–1206.

Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). OpenFst : A general and efficient weighted finite-state transducer library. In *International Conference on Implementation and Application of Automata*, pages 11–23.

Almeida, J. and Zeitoun, M. (2008). Description and analysis of a bottom-up DFA minimization algorithm. *Information Processing Letters*, 107(2) :52–59.

Arrivault, D., Benielli, D., Denis, F., and Eyraud, R. (2017). Sp2Learn : A toolbox for the spectral learning of weighted automata. In *International Conference on Grammatical Inference*, pages 105–119.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv :1607.06450*.

Baevski, A., Hsu, W.-N., Conneau, A., and Auli, M. (2021). Unsupervised speech recognition. *Advances in Neural Information Processing Systems*, 34.

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0 : A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33 :12449–12460.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*.

Bahl, L., Brown, P., De Souza, P., and Mercer, R. (1986). Maximum mutual information estimation of hidden Markov model parameters for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 49–52.

Bahl, L. R., Jelinek, F., and Mercer, R. L. (1983). A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2) :179–190.

Bailly, R., Carreras Pérez, X., and Quattoni, A. J. (2013). Unsupervised spectral learning of FSTs. In *Advances in Neural Information Processing Systems*, volume 26, pages 1–13.

Balle, B., Carreras, X., Luque, F., and Quattoni, A. (2013). Spectral learning of weighted automata a forward-backward perspective. *Machine Learning*, 96 :33–63.

Balle, B. and Mohri, M. (2015). Learning weighted automata. In *International Conference on Algebraic Informatics*, pages 1–21.

Balle, B. and Mohri, M. (2018). Generalization bounds for learning weighted automata. *Theoretical Computer Science*, 716(C) :89–106.

Balle, B., Quattoni, A., and Carreras, X. (2011). A spectral learning algorithm for finite state transducers. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 156–171.

Bassil, Y. and Semaan, P. (2012). ASR context-sensitive error correction based on microsoft n-gram dataset. *arXiv preprint arXiv :1203.5262*.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731) :34–37.

Berkson, J. (1944). Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227) :357–365.

Billa, J. (2017). Improving LSTM-CTC based ASR performance in domains with limited training data. *arXiv preprint arXiv :1707.00722*.

Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). YOLOv4 : Optimal speed and accuracy of object detection. *arXiv preprint arXiv :2004.10934*.

Bourlard, H. and Wellekens, C. J. (1990). Links between Markov models and multilayer perceptrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(12) :1167–1178.

Bourlard, H. A. and Morgan, N. (1994). *Connectionist Speech Recognition : A Hybrid Approach*. Kluwer Academic Publishers.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1) :5–32.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, volume 28, pages 577–585.

Collobert, R., Puhrsch, C., and Synnaeve, G. (2016). Wav2letter : an end-to-end convnet-based speech recognition system. *arXiv preprint arXiv :1609.03193*.

Cosi, P., Bengio, Y., and De Mori, R. (1990). Phonetically-based multi-layered networks for acoustic property extraction and automatic speech recognition. *Speech Communication*, 9(1) :15–30.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International Conference on Machine Learning*, pages 933–941.

Davis, K. H., Biddulph, R., and Balashek, S. (1952). Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6) :637–642.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4) :357–366.

Deng, L. and Yu, D. (2014). Deep learning : methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4) :197–387.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT : Pre-training of deep bidirectional transformers for language understanding. In *2019 Annual Conference of the North American Chapter of the ACL - Human Language Technologies*, pages 4171–4186.

Drugman, T., Pylkkönen, J., and Kneser, R. (2016). Active and semi-supervised learning in asr : Benefits on the acoustic and language models. In *Interspeech*, pages 2318–2322.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61) :2121–2159.

Dufraux, A., Vincent, E., Hannun, A., Brun, A., and Douze, M. (2019). Lead2gold : Towards exploiting the full potential of noisy transcriptions for speech recognition. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop*, pages 78–85.

Eberhard, D. M., Simons, G. F., and Fennig, C. D., editors (2021). *Ethnologue : Languages of the World*, volume 24.

Eilenberg, S. (1974). *Automata, Languages, and Machines*. Academic Press.

Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *40th Annual Meeting of the ACL*, pages 1–8.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2) :179–211.

Errattahi, R., El Hannani, A., and Ouahmane, H. (2018). Automatic speech recognition errors detection and correction : A review. *Procedia Computer Science*, 128 :32–37.

Fedus, W., Zoph, B., and Shazeer, N. (2021). Switch transformers : Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv :2101.03961*.

Forney, G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3) :268–278.

Franzini, M., Lee, K.-F., and Waibel, A. (1990). Connectionist Viterbi training : a new hybrid method for continuous speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 425–428.

Frénay, B. and Verleysen, M. (2013). Classification in the presence of label noise : a survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5) :845–869.

Furui, S. (2005). 50 years of progress in speech and speaker recognition research. *ECTI Transactions on Computer and Information Technology*, 1(2) :64–74.

Gales, M. and Young, S. (2008). The application of hidden Markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3) :195–304.

Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., and Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2494–2498.

Ghorbani, S. and Hansen, J. H. (2018). Leveraging native language information for improved accented speech recognition. In *Interspeech*, pages 2449–2453.

Ghosh, A., Kumar, H., and Sastry, P. (2017). Robust loss functions under label noise for deep neural networks. In *AAAI Conference on Artificial Intelligence*, volume 31, pages 1919–1925.

Gibson, M. and Hain, T. (2006). Hypothesis spaces for minimum Bayes risk training in large vocabulary speech recognition. In *Interspeech*, volume 6, pages 2406–2409.

Gillick, L. and Cox, S. J. (1989). Some statistical issues in the comparison of speech recognition algorithms. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 532–535.

Glenn, M., Strassel, S., Lee, H., Maeda, K., Zakhary, R., and Li, X. (2010). Transcription methods for consistency, volume and efficiency. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification : labelling unsegmented sequence data with recurrent neural networks. In *23rd International Conference on Machine Learning*, pages 369–376.

Grézl, F., Karafiát, M., Kontár, S., and Cernocky, J. (2007). Probabilistic and bottleneck features for LVCSR of meetings. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4, pages IV–757–760.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., et al. (2020). Conformer : Convolution-augmented Transformer for speech recognition. In *Interspeech*, pages 5036–5040.

Guo, J., Sainath, T. N., and Weiss, R. J. (2019). A spelling correction model for end-to-end speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5651–5655.

Hadian, H., Sameti, H., Povey, D., and Khudanpur, S. (2018). End-to-end speech recognition using lattice-free MMI. In *Interspeech*, pages 12–16.

Haffner, P., Waibel, A., Sawai, H., and Shikano, K. (1989). Fast back-propagation learning methods for large phonemic neural networks. In *Eurospeech*, pages 2553–2556.

Han, B., Yao, Q., Yu, X., Niu, G., Xu, M., Hu, W., Tsang, I., and Sugiyama, M. (2018). Co-teaching : Robust training of deep neural networks with extremely noisy labels. *Advances in Neural Information Processing Systems*, 31.

Hannun, A. (2017). Sequence modeling with CTC. *Distill*. https ://distill.pub/2017/ctc.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech : Scaling up end-to-end speech recognition. *arXiv preprint arXiv :1412.5567*.

Hannun, A., Pratap, V., Kahn, J., and Hsu, W.-N. (2020). Differentiable weighted finite-state transducers. *arXiv preprint arXiv :2010.01003*.

Hasegawa-Johnson, M. A., Jyothi, P., McCloy, D., Mirbagheri, M., Di Liberto, G. M., Das, A., Ekin, B., Liu, C., Manohar, V., Tang, H., et al. (2016). ASR for under-resourced languages from probabilistic transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1) :50–63.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Heafield, K. (2011). Kenlm : Faster and smaller language model queries. In *6th Workshop on Statistical Machine Translation*, pages 187–197.

Heafield, K., Pouzyrevsky, I., Clark, J. H., and Koehn, P. (2013). Scalable modified Kneser-Ney language model estimation. In *51st Annual Meeting of the ACL*, pages 690–696.

Hendrycks, D., Mazeika, M., Wilson, D., and Gimpel, K. (2018). Using trusted data to train deep networks on labels corrupted by severe noise. In *Advances in Neural Information Processing Systems*, volume 31, pages 10477–10486.

Hermansky, H., Ellis, D. P., and Sharma, S. (2000). Tandem connectionist feature extraction for conventional HMM systems. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1635–1638.

Hinton, G. E., Srivastava, N., and Swersky, K. (2012). Lecture 6d - a separate, adaptive learning rate for each connection. In *Slides of Lecture Neural Networks for Machine Learning*. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). Introduction to automata theory, languages, and computation. *ACM SIGACT News*, 32(1) :60–65.

Hsu, D., Kakade, S. M., and Zhang, T. (2012). A spectral algorithm for learning hidden Markov models. *Journal of Computer and System Sciences*, 78(5) :1460–1480.

Hsu, W.-N., Bolte, B., Tsai, Y.-H., Lakhotia, K., Salakhutdinov, R., and Mohamed, A. (2021). HuBERT : Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29 :3451–3460.

Iwamida, H., Katagiri, S., and McDermott, E. (1991). Speaker-independent large vocabulary word recognition using an LVQ/HMM hybrid algorithm. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 553–556.

Jaitly, N. and Hinton, G. (2011). Learning a better representation of speech soundwaves using restricted boltzmann machines. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5884–5887.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J., and Fei-Fei, L. (2018). Mentornet : Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2304–2313.

Juang, B. H. and Rabiner, L. R. (1991). Hidden Markov models for speech recognition. *Technometrics*, 33(3) :251–272.

Kahn, J., Lee, A., and Hannun, A. (2020). Self-training for end-to-end speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7084–7088.

Kaneko, T., Ushiku, Y., and Harada, T. (2019). Label-noise robust generative adversarial networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2467–2476.

Kapadia, S., Valtchev, V., and Young, S. J. (1993). MMI training for continuous phoneme recognition on the TIMIT database. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 491–494.

Khetan, A., Lipton, Z. C., and Anandkumar, A. (2018). Learning from noisy singly-labeled data. In *International Conference on Learning Representations*.

Killer, M., Stüker, S., and Schultz, T. (2003). Grapheme based speech recognition. In *Eurospeech*, pages 3141–3144.

Kingma, D. P. and Ba, J. (2014). Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184.

Knight, K. and May, J. (2009). Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pages 571–596.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105.

Kudo, T. (2018). Subword regularization : Improving neural network translation models with multiple subword candidates. In *56th Annual Meeting of the ACL*, pages 66–75.

Kuich, W. and Salomaa, A. (2012). *Semirings, Automata, Languages*, volume 5. Springer Science and Business Media.

Lang, K. J. (1988). The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie Mellon University.

LeCun, Y. (1985). Une procédure d'apprentissage pour un réseau à seuil asymétrique. *Proceedings of Cognitiva*, 85 :599–604.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553) :436–444.

Lee, K.-H., He, X., Zhang, L., and Yang, L. (2018). Cleannet : Transfer learning for scalable image classifier training with label noise. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5447–5456.

Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710.

Levin, E. (1990). Word recognition using hidden control neural architecture. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 433–436.

Li, J., Wong, Y., Zhao, Q., and Kankanhalli, M. S. (2019). Learning to learn from noisy labeled data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5051–5059.

Li, Y., Yang, J., Song, Y., Cao, L., Luo, J., and Li, L.-J. (2017). Learning from noisy labels with distillation. In *IEEE International Conference on Computer Vision*, pages 1910–1918.

Likhomanenko, T., Xu, Q., Kahn, J., Synnaeve, G., and Collobert, R. (2020). slimipl : Language-model-free iterative pseudo-labeling. *arXiv preprint arXiv :2010.11524*.

Ling, S., Shen, C., Cai, M., and Ma, Z. (2021). Improving pseudo-label training for end-to-end speech recognition using gradient mask. *arXiv preprint arXiv :2110.04056*.

Lippmann, R. P. (1989). Review of neural networks for speech recognition. *Neural Computation*, 1(1) :1–38.

Lippmann, R. P. (1997). Speech recognition by machines and humans. *Speech communication*, 22(1) :1–15.

Liptchinsky, V., Synnaeve, G., and Collobert, R. (2017). Letter-based speech recognition with gated ConvNets. *arXiv preprint arXiv :1712.09444*.

Lowerre, B. T. (1976). *The Harpy Speech Recognition System*. PhD thesis.

Lüscher, C., Beck, E., Irie, K., Kitza, M., Michel, W., Zeyer, A., Schlüter, R., and Ney, H. (2019). RWTH ASR systems for LibriSpeech : Hybrid vs attention-w/o data augmentation. *arXiv preprint arXiv :1905.03072*.

Ma, X., Huang, H., Wang, Y., Romano, S., Erfani, S., and Bailey, J. (2020a). Normalized loss functions for deep learning with noisy labels. In *International Conference on Machine Learning*, pages 6543–6553.

Ma, X., Wang, Y., Houle, M. E., Zhou, S., Erfani, S., Xia, S., Wijewickrema, S., and Bailey, J. (2018). Dimensionality-driven learning with noisy labels. In *International Conference on Machine Learning*, pages 3355–3364.

Ma, Y., Narayanaswamy, B., Lin, H., and Ding, H. (2020b). Temporal-contextual recommendation in real-time. In *26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2291–2299.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2) :269–311.

Mohri, M. (2009). Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer.

Mohri, M., Pereira, F., and Riley, M. (2000). The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1) :17–32.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech and Language*, 16(1) :69–88.

Mohri, M., Pereira, F., and Riley, M. (2005). Weighted automata in text and speech processing. *arXiv preprint cs/0503077*.

Mohri, M., Pereira, F., and Riley, M. (2008). Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer.

Moradi, M. and Samwald, M. (2021). Evaluating the robustness of neural language models to input perturbations. In *Conference on Empirical Methods in Natural Language Processing*.

Morgan, N. and Bourlard, H. (1990). Continuous speech recognition using multilayer perceptrons with hidden Markov models. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 413–416.

Mori, D. et al. (1989). BPS : A learning algorithm for capturing the dynamic nature of speech. In *International Joint Conference on Neural Networks*, pages 417–423.

Mozer, M. C. (1993). Neural net architectures for temporal sequence processing. In *Santa Fe Institute Studies in the Sciences of Complexity*, volume 15, pages 243–243.

Nair, V. and Hinton, G. (2010). Rectified linear units improve restricted Boltzmann machines. In *International Conference on Machine Learning*, volume 27, pages 807–814.

Namysl, M., Behnke, S., and Köhler, J. (2020). NAT : Noise-aware training for robust neural sequence labeling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1501–1517.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1) :31–88.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech : an ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5206–5210.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., and Le, Q. V. (2019). SpecAugment : A simple data augmentation method for automatic speech recognition. In *Interspeech*, pages 2613–2617.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch : An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037.

Patrini, G., Rozza, A., Krishna Menon, A., Nock, R., and Qu, L. (2017). Making deep neural networks robust to label noise : A loss correction approach. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1944–1952.

Paul, D. B. and Baker, J. M. (1992). The design for the Wall Street Journal-based CSR corpus. In *Workshop on Speech and Natural Language*, pages 357–362.

Piktus, A., Edizel, N. B., Bojanowski, P., Grave, E., Ferreira, R., and Silvestri, F. (2019). Misspelling oblivious word embeddings. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3226–3234.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al. (2011). The Kaldi speech recognition toolkit. In *IEEE Workshop on Automatic Speech Recognition and Understanding*.

Povey, D., Hannemann, M., Boulianne, G., Burget, L., Ghoshal, A., Janda, M., Karafiát, M., Kombrink, S., Motlíček, P., Qian, Y., et al. (2012). Generating exact lattices in the WFST framework. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4213–4216.

Povey, D., Kuang, F., and Qiu, H. (2020). k2. https://github.com/k2-fsa/k2. Consulté le 09/09/2021.

Pratap, V., Hannun, A., Synnaeve, G., and Collobert, R. (2022). Star Temporal Classification : Sequence classification with partially labeled data. *arXiv preprint arXiv :2201.12208*.

Pratap, V., Hannun, A., Xu, Q., Cai, J., Kahn, J., Synnaeve, G., Liptchinsky, V., and Collobert, R. (2019). Wav2letter++ : A fast open-source speech recognition system. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6460–6464.

Rabiner, L. (1993). *Fundamentals of Speech Recognition*. PTR Prentice Hall.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–286.

Rao, K., Sak, H., and Prabhavalkar, R. (2017). Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer. In *IEEE Automatic Speech Recognition and Understanding Workshop*, pages 193–199.

Revuz, D. (1992). Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1) :181–189.

Richard, M. D. and Lippmann, R. P. (1991). Neural network classifiers estimate Bayesian a posteriori probabilities. *Neural Computation*, 3(4) :461–483.

Robinson, T. (1992). A real-time recurrent error propagation network word recognition system. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 92, pages 617–620.

Rolnick, D., Veit, A., Belongie, S., and Shavit, N. (2017). Deep learning is robust to massive label noise. *arXiv preprint arXiv :1705.10694*.

Rosenblatt, F. (1958). The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6) :386.

Salimans, T. and Kingma, D. P. (2016). Weight normalization : A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 29, pages 901–909.

Saon, G., Kurata, G., Sercu, T., Audhkhasi, K., Thomas, S., Dimitriadis, D., Cui, X., Ramabhadran, B., Picheny, M., Lim, L.-L., Roomi, B., and Hall, P. (2017). English Conversational Telephone Speech Recognition by Humans and Machines. In *Interspeech*, pages 132–136.

Sarma, A. and Palmer, D. D. (2004). Context-based speech recognition error detection and correction. In *Proceedings of HLT-NAACL : Short Papers*, pages 85–88.

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec : Unsupervised pre-training for speech recognition. *arXiv preprint arXiv :1904.05862*.

Sengupta, S., Pratap, V., and Hannun, A. (2021). Parallel composition of weighted finite-state transducers. *arXiv preprint arXiv :2110.02848*.

Song, H., Kim, M., Park, D., Shin, Y., and Lee, J.-G. (2020). Learning from noisy labels with deep neural networks : A survey. *arXiv preprint arXiv :2007.08199*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1) :1929–1958.

Stolcke, A. and Droppo, J. (2017). Comparing human and machine errors in conversational speech transcription. *arXiv preprint arXiv :1708.08615*.

Stolcke, A., Grezl, F., Hwang, M.-Y., Lei, X., Morgan, N., and Vergyri, D. (2006). Cross-domain and cross-language portability of acoustic features estimated by multilayer perceptrons. In *IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages 321–324.

Sukhbaatar, S., Bruna, J., Paluri, M., Bourdev, L., and Fergus, R. (2014). Training convolutional networks with noisy labels. *arXiv preprint arXiv :1406.2080*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27, pages 3104–3112.

Synnaeve, G. (2021). WER are we ? https://github.com/syhw/wer_are_we. Consulté le 14/09/2021.

Synnaeve, G., Xu, Q., Kahn, J., Likhomanenko, T., Grave, E., Pratap, V., Sriram, A., Liptchinsky, V., and Collobert, R. (2019). End-to-end ASR : from supervised to semi-supervised learning with modern architectures. *arXiv preprint arXiv :1911.08460*.

Szymański, P., Żelasko, P., Morzy, M., Szymczak, A., Żyła-Hoppe, M., Banaszczak, J., Augustyniak, L., Mizgajski, J., and Carmiel, Y. (2020). WER we are and WER we think we are. *arXiv preprint arXiv :2010.03432*.

Tanaka, D., Ikami, D., Yamasaki, T., and Aizawa, K. (2018). Joint optimization framework for learning with noisy labels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5552–5560.

Tjandra, A., Sakti, S., and Nakamura, S. (2017). Attention-based wav2text with feature transfer learning. In *IEEE Automatic Speech Recognition and Understanding Workshop*, pages 309–315.

Touvron, H., Vedaldi, A., Douze, M., and Jégou, H. (2020). Fixing the train-test resolution discrepancy : Fixefficientnet. *arXiv preprint arXiv :2003.08237*.

Trentin, E. and Gori, M. (2001). A survey of hybrid ANN/HMM models for automatic speech recognition. *Neurocomputing*, 37(1-4) :91–126.

Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer.

Tóth, L., Frankel, J., Gosztolya, G., and King, S. (2008). Cross-lingual portability of MLP-based tandem features - a case study for English and Hungarian. In *Interspeech*, pages 2695–2698.

Vahdat, A. (2017). Toward robustness against label noise in training deep discriminative neural networks. In *Advances in Neural Information Processing Systems*, volume 30.

Vapnik, V. (2013). *The Nature of Statistical Learning Theory*. Springer Science and Business Media.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008.

Veit, A., Alldrin, N., Chechik, G., Krasin, I., Gupta, A., and Belongie, S. (2017). Learning from noisy large-scale datasets with minimal supervision. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 839–847.

Vergyri, D., Lamel, L., and Gauvain, J.-L. (2010). Automatic speech recognition of multiple accented english data. In *Interspeech*, pages 1652–1655.

Veselý, K., Ghoshal, A., Burget, L., and Povey, D. (2013). Sequence-discriminative training of deep neural networks. In *Interspeech*, pages 2345–2349.

Waibel, A. (1989). Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1(1) :39–46.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3) :328–339.

Wang, Y., Ma, X., Chen, Z., Luo, Y., Yi, J., and Bailey, J. (2019). Symmetric cross entropy for robust learning with noisy labels. In *IEEE/CVF International Conference on Computer Vision*, pages 322–330.

Warwick, K. and Shah, H. (2016). Can machines think ? A report on Turing test experiments at the Royal Society. *Journal of Experimental and Theoretical Artificial Intelligence*, 28(6) :989–1007.

Xiao, T., Xia, T., Yang, Y., Huang, C., and Wang, X. (2015). Learning from massive noisy labeled data for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2691–2699.

Xie, L., Wang, J., Wei, Z., Wang, M., and Tian, Q. (2016). Disturblabel : Regularizing CNN on the loss layer. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4753–4762.

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M. L., Stolcke, A., Yu, D., and Zweig, G. (2017a). Toward human parity in conversational speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25 :2410–2423.

Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M. L., Stolcke, A., Yu, D., and Zweig, G. (2017b). Toward human parity in conversational speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(12) :2410–2423.

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., and Stolcke, A. (2018). The Microsoft 2017 conversational speech recognition system. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5934–5938.

Xu, Q., Likhomanenko, T., Kahn, J., Hannun, A., Synnaeve, G., and Collobert, R. (2020). Iterative pseudo-labeling for speech recognition. *arXiv preprint arXiv :2005.09267*.

Yi, K. and Wu, J. (2019). Probabilistic end-to-end noise correction for learning with noisy labels. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7017–7025.

Young, S. (2021). *Hey Cyba : The Inner Workings of a Virtual Personal Assistant*. Cambridge University Press.

Yu, D., Varadarajan, B., Deng, L., and Acero, A. (2010). Active learning and semi-supervised learning for speech recognition : A unified framework using the global entropy reduction maximization criterion. *Computer Speech & Language*, 24(3) :433–444.

Zeghidour, N. (2019). *Learning Representations of Speech from the Raw Waveform*. PhD thesis.

Zeiler, M. D. (2012). Adadelta : an adaptive learning rate method. *arXiv preprint arXiv :1212.5701*.

Zhang, Z. and Sabuncu, M. R. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. In *Advances in Neural Information Processing Systems*, volume 31, pages 8792–8802.

Zolnay, A., Schlüter, R., and Ney, H. (2003). Extraction methods of voicing feature for robust speech recognition. In *Eurospeech*, pages 497–500.