MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

Segmentation Automatique de la Langue des Signes de Belgique Francophone à l'aide de Réseaux Neuronaux Récurrents

POITIER, Pierre

Award date:
2022

Awarding institution:
Universite de Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

Segmentation Automatique de la Langue des Signes de Belgique Francophone à l'aide de Réseaux Neuronaux Récurrents

Pierre Poitier

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Remerciements

Je tiens à remercier toutes les personnes grâce à qui la réalisation de ce mémoire a été rendue possible. Je voudrais dans un premier temps adresser toute ma gratitude au promoteur de ce mémoire, Benoît Frenay, ainsi qu'à son co-promoteur, Jérôme Fink, pour leur patience, leur disponibilité et leurs nombreux conseils. Leur supervision a été des plus précieuses tant pour mon stage que pour la réalisation d'un article et pour mon mémoire. J'ai énormément appris durant ces derniers mois en grande partie grâce à eux.

J'aimerais également remercier Laurence Meurant ainsi que l'entièreté de l'équipe du LSFB-Lab de m'avoir permis de travailler sur une cause aussi noble. C'est un honneur de pouvoir participer, à ma mesure, à une entreprise aussi humaniste. Ma gratitude va également aux membres du personnel de la faculté d'informatique de l'université de Namur pour leur accueil chaleureux dont j'ai bénéficié en tant que stagiaire.

Finalement, j'aimerais exprimer ma plus profonde gratitude à ma famille et à mes amis pour leur présence et leur indéfectible support durant toutes ces années d'études. Je leur serai éternellement reconnaissant.

Abstract

Linguistic research on sign languages has led to the creation of substantial corpora. Indeed, one of the most important ones tackles the French Belgian Sign Language (LSFB) and is the result of the work of the LSFB-Lab at the University of Namur. This corpus has motivated the development of a contextual dictionary from LSFB into French. Such a system is challenging, but it would help and improve the inclusion of the deaf community in society. A mandatory step to automate the continuous translation of sign language is segmentation to distinguish between signs within videos of ongoing discussions. Such a system could also be used to facilitate the annotation of the videos. Indeed, increasing the amount of annotated data is key to improve the quality of the automated translation. This master thesis presents and evaluates a sign language segmentation system developed based on Recurrent Neural Networks (RNNs). This system is incrementally improved and evaluated through different experiments that highlight the importance of taking into account the coarticulation phenomenon, as well as a better transition modelling within RNNs. In parallel, new datasets are created from the available LSFB videos, and an assessment of the importance of the signers' joints in the segmentation is conducted.

Résumé

La recherche linguistique sur la langue des signes a abouti à la création de corpus conséquents. En effet, un des plus importants concerne la langue des signes de Belgique francophone (LSFB) et est issu du travail du LSFB-Lab de l'université de Namur. Ce corpus a motivé le développement d'un dictionnaire contextuel de la LSFB vers le français. L'enjeu d'un tel système est important. Cela permettrait d'aider et d'améliorer l'inclusion des personnes muettes ou malentendantes au sein de la société. Une étape préliminaire à l'automatisation de la traduction de la langue des signes consiste en une segmentation afin de distinguer les signes entre eux au sein de vidéos de discussions continues. Un tel système pourrait également être utilisé afin de faciliter le travail d'annotations de ces vidéos. En effet, l'augmentation de la quantité de données annotées est un facteur clé afin d'améliorer la qualité de la traduction automatique. Ce mémoire présente et évalue un système de segmentation de la langue des signes développé à l'aide de réseaux neuronaux récurrents (RNNs). Ce système est amélioré et évalué de façon incrémentale dans différentes expériences qui mettent en avant l'importance de la prise en compte du phénomène de coarticulation, ainsi que d'une meilleure modélisation des transitions au sein des RNNs. En parallèle, de nouveaux datasets sont créés à partir des vidéos LSFB disponibles, tandis qu'une évaluation de l'importance des articulations des signeurs dans la segmentation est menée.

Table des matières

Chapitre 1

Introduction

La reconnaissance automatique de signes à partir de vidéos en langue des signes (vidéos LS) est un sujet actif au sein de la communauté scientifique [1], notamment en ce qui concerne la reconnaissance de mouvements [2]. De tels systèmes de reconnaissance pourraient améliorer l'inclusion des personnes malentendantes ou muettes, par exemple, au sein du système éducatif [3]. Un objectif dans ce sens est l'élaboration d'outils en ligne afin d'aider à comparer le français et la langue des signes francophone belge (LSFB) [4].

L'élaboration d'un **corpus** entre le français et la LSFB [5], par le LSFB-Lab de l'université de Namur [1], constitue un premier outil disponible. Celui-ci a pour ambition de mettre à disposition un dictionnaire contextuel entre le français et la LSFB. Le corpus [5] contient 50 heures de vidéos en langue des signes annotées manuellement. Ces ressources sont décrites dans un article [6] présentant une première étape de traduction d'un signe isolé vers le français (traduction LS).

Ce mémoire aborde une étape préliminaire à la traduction LS : la **segmentation automatique** des vidéos continues en langue des signes (segmentation LS). En effet, la traduction LS ainsi que tous les processus opérant sur des signes individuels nécessitent de distinguer les signes entre eux. Or, le dataset LSFB_CONT [6] fournit des vidéos de discussions continues en langue des signes. Il est donc nécessaire de les isoler à travers une étape de segmentation LS (voir Figure 1.1), également appelée tokenization. Contrairement à la reconnaissance LS, celle-ci est peu étudiée dans la littérature.

De façon analogue à la tokenization des mots dans le traitement du langage naturel (NLP), une segmentation LS efficace serait une contribution notable qui mènerait à de nouvelles perspectives dans la recherche sur le traitement automatique de la langue des signes. Cela permettrait de simplifier le travail d'annotation des vidéos LS (actuellement complètement manuel), mais également d'incorporer une étape de tokenization au sein de systèmes plus complexes.

L'intérêt envers la segmentation LS et le fait que peu de recherches aient été réalisées sur le sujet placent la segmentation automatique de la langue des signes au centre de ce mémoire. Celui-ci est structuré en plusieurs chapitres distincts.

1. https ://www.corpus-lsfb.be/

Les premiers chapitres présentent l'état d'art tandis que les suivants détaillent les expériences réalisées afin de constituer un système de segmentation LS automatique.

Premièrement, le chapitre 2 décrit l'état d'art en segmentation LS. Une description du dataset LSFB_CONT [6] y est également présente. Celui-ci a été choisi car il est un des plus grand datasets de vidéos continues en langue des signes et est constitué de discussions non-contraintes en LSFB. La segmentation LS réalisée dans ce mémoire l'utilise comme source de données et comme principal outil d'évaluation des résultats.

Ensuite, le chapitre 3 décrit les techniques de machine learning utilisées afin de constituer un système de segmentation LS automatique. Une description détaillée des modèles séquentiels et des réseaux de neurones récurrents (RNN) y est donnée. En effet, la segmentation LS dans ce mémoire utilise un type spécifique de RNN, le Long Short-Term Memory (LSTM, voir section 3.3.2).

Après ces états de l'art, le chapitre 4 est dédié aux nouveaux datasets créés durant la recherche effectuée sur la segmentation SL automatique. Ceux-ci contiennent des squelettes (voir Figure 1.1) extraits des vidéos du dataset LSFB_CONT [6]. Bien que tous ces datasets ne soient pas utilisés dans les expériences, ceux-ci constituent des sources de données potentielles pour de futures recherches.

Les chapitres 5, 6 et 7 sont, quant à eux, dédiés à la présentation des expériences réalisées dans le cadre de ce mémoire. Le chapitre 5 concerne l'importance de la coarticulation et de sa considération afin d'améliorer les résultats de la segmentation. Ensuite, le chapitre 6 se concentre sur des méthodes permettant de meilleures modélisations des transitions au sein des RNNs. Des extensions du LSTM y sont présentées ainsi que leur apports pour la segmentation SL. Ces chapitres 5 et 6 ont d'ailleurs mené à la réalisation d'un article scientifique soumis à la conférence internationale ESANN [2]. Celui-ci est en annexe de ce mémoire. Le chapitre 7, lui, présente des améliorations diverses afin d'optimiser les résultats du système de segmentation LS.

Le chapitre 8, quant à lui, se concentre sur l'importance des différentes données par rapport aux résultats obtenus par le système de segmentation SL. L'importance des différents points de repères de la main et du squelette y est évaluée.

Finalement, le chapitre 9 récapitule les conclusions des expériences menées ainsi que les résultats finaux du systèmes de segmentation LS automatique. Des perspectives et améliorations y sont également présentées.

2. https ://www.esann.org/

FIGURE 1.1 – Exemple de vidéo de conversation continue en langue des signes avec la segmentation associée. La ligne du temps représente la présence (gris) ou l'absence (blanc) de signes pour chaque frame de la vidéo.

Résumé des objectifs

L'objectif principal de ce mémoire est de mettre en avant des pistes de solution aux difficultés rencontrées dans la segmentation automatique des vidéos en langue des signes. Pour cela, divers expériences et benchmarks sont réalisés. Une attention particulière est donnée aux courtes périodes entre deux signes, nommées coarticulations.

Ensuite, un second objectif est la création d'un tokenizer afin de faciliter et d'accélérer l'annotation des vidéos LS. Cela serait réalisable grâce à une étape de segmentation automatique réalisée au préalable.

En outre, un troisième objectif est une description détaillées des nouveaux datasets constitués à partir des vidéos LS munies de leurs annotations. Cela est réalisé afin qu'ils puissent servir à de futures recherches.

Finalement, un dernier objectif est d'évaluer l'importance des données utilisées comme entrées dans les modèles réalisés. Cela afin d'améliorer la confiance que l'on peut avoir à l'égard du système de segmentation LS, mais également pour potentiellement soulever de nouvelles questions de recherche en linguistique.

Chapitre 2

État de l'art en segmentation de signes

Ce chapitre résume l'état de l'art en segmentation automatique de la langue des signes. Bien que, au sein de vidéos, la reconnaissance de signes et la segmentation des signes soient deux problématiques distinctes, ces deux champs de recherche utilisent des techniques similaires. Un court état de l'art quant à la reconnaissance automatique de signes est donc réalisé avant l'état de l'art sur la segmentation automatique des signes. Ensuite, le dataset LSFB_CONT est décrit section 2.3 et est mis en relation avec ces champs de recherche.

2.1 La reconnaissance de signes

La reconnaissance de signes (reconnaissance LS) est un des problèmes les plus étudiés en reconnaissance de mouvements [7, 2]. Il y a plusieurs raisons à cela. Premièrement, il s'agit d'une tâche complexe et d'un problème toujours ouvert. C'est donc un choix pertinent pour mettre à l'épreuve des systèmes de reconnaissance de mouvements. Ensuite, il y a un enjeu sociétal important à mieux intégrer la communauté de personnes mal-entendantes ou muettes dans la société via, par exemple, des interfaces mieux adaptées [8].

La reconnaissance de signes n'est pas un problème nouveau. En effet, dès les années 90 [1], de nombreuses études traitent le sujet. Cependant, de multiples changements de paradigmes ont eu lieu au cours du temps.

La reconnaissance de mouvement exploite une information spatiale, mais également une information temporelle. L'approche la plus courante est l'utilisation de vidéos en langue des signes à travers des techniques de reconnaissance d'images [2]. Celles-ci ont l'avantage de ne nécessiter qu'un support vidéo capté avec une simple caméra, par exemple, grâce à un smartphone. Une autre problématique abordée est l'utilisation de périphériques spécialisés qui apportent des données supplémentaires. Par exemple, des bracelets permettant de localiser les mains, ou encore des caméras 3D telles que la Kinect (Figure 2.1) [9, 10] qui permettent de fournir un tracking ainsi qu'une information de profondeur. Ce-

pendant, de tels dispositifs sont peu accessibles [2]. En effet, ces périphériques pourraient freiner l'adoption au grand public, car ils représentent un certain coût et nécessitent une certaine expertise. En parallèle, l'évolution fulgurante des techniques de reconnaissance de mouvement à partir de vidéos a d'autant plus encouragé la communauté scientifique à suivre cette voie. Dès lors, depuis 2005, peu de recherches en reconnaissance de signes utilisent des périphériques spécialisés [1].

FIGURE 2.1 – Tracking à l'aide d'une Kinect [10].

En ce qui concerne l'exploitation des informations de mouvements et de l'aspect séquentiel des vidéos en langue des signes, traditionnellement, les techniques statistiques ont eu du succès à partir des années 90 jusqu'aux années 2010 [11, 2, 1]. L'utilisation de processus stochastiques tels que les Hidden Markov Models (HMMs, voir section 3.3.1) était très répandue [11]. L'utilisation des HMMs a été en grande partie remplacée par les réseaux récurrents tel que le LSTM (voir section 3.3.2) [12]. De 2015 à 2020, la grande majorité des systèmes de reconnaissance de signes utilisent des réseaux récurrents (RNN, voir section 3.3.2) couplés à des réseaux convolutifs (CNN) [1]. Plus récemment, les réseaux transformer gagnent en popularité et concurrencent les réseaux récurrents en reconnaissance LS [13].

Pour conclure, à l'heure actuelle, la communauté scientifique utilise essentiellement des réseaux récurrents et des supports vidéo afin de constituer des systèmes de reconnaissance automatique de la langue des signes. Ces techniques sont également pertinentes dans la segmentation automatique de la langue des signes, étant donné que celle-ci est fortement liée à la reconnaissance LS.

2.2 La segmentation automatique de signes

Ces dernières années, des avancées majeures ont été réalisées dans le domaine du traitement du langage naturel (NLP) [14] avec des applications telles que l'analyse des sentiments, les traducteurs, la synthèse vocale, la détection des erreurs grammaticales et même des systèmes de dialogue [15].

Un élément fondamental dans le NLP est une étape de preprocessing qui consiste à diviser un texte en phrases ou mots, appelés tokens. Cette étape est appelée

la tokenization et est critique pour une bonne exploitation du texte dans les systèmes. En effet, ceux-ci appliquent une série d'opérations à chaque token.

Alors qu'il est facile de trouver de bons tokenizers [16] pour du texte en français ou en anglais, ce n'est pas le cas pour la langue des signes (LS). En effet, dans un texte, les espaces ou les signes de ponctuation permettent, respectivement, de bien délimiter les mots ou les phrases. En langue des signes, l'absence de frontière claire entre les signes rend leur différenciation complexe au sein d'une vidéo.

À cette difficulté s'en ajoutent d'autres. Premièrement, l'annotation des signes est une tâche manuelle, lente et très chronophage. En conséquence, le nombre de datasets disponibles est limité [1]. Ensuite, l'activité d'un signeur dans une période entre deux signes est fortement dépendante de ceux-ci. Ce phénomène s'appelle la coarticulation [17] et rend la frontière entre deux signes imprécise et contextuelle. Une autre difficulté est le fait qu'un signeur utilise ses deux mains et que la fin d'un signe est susceptible de se superposer avec le début d'un autre. Finalement, là ou un mot possède une orthographe définie, un signe est sujet à des variations suivant, par exemple, sa vitesse d'exécution ou encore les habitudes du signeur.

La segmentation automatique des signes consiste donc en un système capable d'isoler et de distinguer les signes entre eux au sein de vidéos de discussions en langue des signes (vidéos LS). Une segmentation performante ferait office de tokenizer pour la langue des signes (tokenizer LS). La création d'un tokenizer efficace adapté à la LS serait fort utile pour la traduction de vidéos continues en langue des signes (traduction LS). En effet, tout comme un tokenizer est critique pour les systèmes NLP, il pourrait ouvrir de nouvelles perspectives au sein des systèmes de traitement automatique de la langue des signes. Cela pourrait également accélérer l'annotation de vidéos. En conséquence, cela pourrait aboutir à la création d'un cercle vertueux où un plus grand nombre de vidéos annotées permet la création de modèles plus performants, permettant ainsi d'accélérer davantage l'annotation.

Une particularité de ce mémoire est le fait que l'approche abordée n'utilise pas la sémantique des signes dans la segmentation. Dès lors, celle-ci ne se restreint pas à un vocabulaire précis et se veut plus générale. D'autres études [18, 19] sur la segmentation LS utilisent un vocabulaire précis dans un contexte de traduction.

Bien que la reconnaissance de signes (reconnaissance LS) soit un sujet populaire au sein de la communauté scientifique, la segmentation de la langue des signes (segmentation LS), elle, est peu étudiée. Néanmoins, plusieurs articles s'intéressent à la segmentation des signes et aux problématiques évoquées. Trois d'entre eux sont retenus dans cet état de l'art [18, 19, 20].

Le premier est l'article de *Elakkiya* et *Selvamani* [18] qui s'intéresse explicitement à la segmentation des signes au sein de vidéos continues. Celui-ci se base sur la reconnaissance des mains avec des HMMs (voir section 3.3.1) et utilise un framework dans lequel la coarticulation est implicitement modélisée. Cet article soulève comme problématique la difficulté d'identifier les coarticulations au sein des données d'entraînement. Cette tâche peut être réalisée manuellement, mais elle est fastidieuse et très longue. Dès lors, une modélisation implicite est préférée.

Ensuite, l'article de *Yang*, *Sarkar* et *Loeding* [19], lui, s'intéresse aux ambiguïtés dans la segmentation des mains au sein d'une vidéo continue. En effet, les mains peuvent se superposer et la détection des mouvements des mains peut être imprécise. Un élément important est l'accent mis sur l'épenthèse des mouvements entre les mains [17]. C'est-à-dire l'intérêt porté aux mouvements intermédiaires entre deux mouvements principaux. Cette problématique rejoint également la difficulté de l'identification de la coarticulation. En effet, la coarticulation correspond à un phénomène d'épenthèse dans les mouvements du signeur.

Finalement, l'étude menée par *Bull*, *Gouiffès* et *Braffort* [20] s'intéresse explicitement à la segmentation automatique de la langue des signes. L'objectif est de constituer des unités sous-titrables au sein de vidéos continues en langue des signes. Cette problématique est équivalente à celle de la tokenization LS. En effet, au même titre que l'étude menée dans ce mémoire, cet article ne s'intéresse pas à la sémantique des signes dans la segmentation. Cependant, il n'aborde pas la problématique de la coarticulation. Un élément important au sein de l'article est l'utilisation de séquences de squelettes au lieu de vidéos. C'est en nous inspirant de ce travail que nous avons décidé d'utiliser des squelettes (voir chapitre 4) dans les expériences de ce mémoire (voir chapitres 5, 6, et 7).

2.3 Le dataset LSFB_CONT

Afin de construire un système de segmentation LS automatique, il est important de disposer d'une grande quantité de données annotées de qualité. Plusieurs datasets fournissent des vidéos continues de discussions en langue des signes annotées, notamment :

- **RWTH Phoenix Weather 2014** [21] est un dataset contenant des enregistrements en langue des signes de prédictions météo publiques en Allemagne ;
- **Greek Sign Language** (GSL) [22] est un dataset contenant des discussions (scriptées) entre des agents publiques dans la langue des signes Grec ;
- **LSFB_CONT** [6] est un dataset contenant des discussions sans contraintes entre deux signeurs dans la langue des signes francophone de Belgique.

Dans ce mémoire, le dataset LSFB_CONT [6] est systématiquement utilisé pour plusieurs raisons. Premièrement, celui-ci est un des plus grands dataset de vidéos continues en langue des signes au monde (voir Table 2.1) contenant plus de 100 signeurs différents et plus de 90 heures de discussions. Ensuite, il a été constitué dans le cadre du corpus LSFB [5] dont l'objectif est de fournir un dictionnaire contextuel entre la LSFB et le français [4]. Pour remplir cet objectif, beaucoup de soin a été apporté à l'annotation de ce corpus, ce qui le rend très fiable. Un premier système de traduction, à partir de signes isolés, est d'ailleurs décrit dans l'article original du dataset [6]. L'expertise présente à Namur en ce qui concerne la LSFB motive, aussi, l'utilisation de ce dataset. Finalement, les discussions entre signeurs sont sans contraintes de vocabulaire et sans contrainte de vitesse d'exécution des signes. Par conséquent, la diversité des signes rencontrés dans les discussions (6883 signes différents) rend le dataset plus réaliste. Cela permet de constituer un système de segmentation LS plus robuste.

FIGURE 2.2 – Exemples de vidéos présentes dans le dataset LSFB_CONT

dataset	n. de signes	n. de signeurs	heures	FPS
LSFB_CONT	6883	100	90	50
Phoenix Wheater	1558	9	10.73	25
GSL	310	7	9.6	30

TABLE 2.1 – Comparaison des datasets de vidéos continues en langue des signes

Le dataset LSFB_CONT [6] contient différentes données :

- des vidéos continues en langue des signes (vidéos LS) ;
- des annotations relatives aux vidéos ;
- des métadonnées quant au signeur, à la session d'enregistrement, etc. ;
- des traductions continues.

Les vidéos LS sont au format MP4, en couleur (RGB), leur résolution est de 720 sur 576 et leur taux de rafraîchissement est de 50 images par seconde (FPS). La durée de chaque vidéo varie de 1 ou 2 minutes à plus de 30 minutes. Celles-ci contiennent un seul signeur. Chaque discussion est donc séparée en deux vidéos chacune contenant un signeur. La position de celui-ci et la position de la caméra sont contrôlées. L'arrière-plan uniforme est contrôlé également.

Chaque vidéo possède des annotations séparées entre la main droite et la main gauche. Celles-ci sont au format de fichier ELAN (XML) [23]. Chaque annotation contient comme information :

- un intervalle de temps au sein de la vidéo ;
- une main associée ;
- un gloss, c'est-à-dire un identifiant du signe permettant de distinguer les différents signes entre eux.

2.4 Récapitulatif

De nos jours, les recherches en reconnaissance de signes utilisent essentiellement des supports vidéos ainsi que des algorithmes de machine learning tels que les réseaux récurrents [1]. Un exemple d'initiative incorporant un système de reconnaissance de signes est le dictionnaire contextuel réalisé avec le corpus LSFB [5].

Une étape préliminaire dans le traitement de vidéos continues en langue des signes est l'isolation des signes au sein de celles-ci. Cette étape est appelée segmentation automatique de la langue des signes (segmentation LS) ou encore tokenization de la langue des signes (tokenization LS). Bien que la reconnaissance de signes soit un sujet actif au sein de la communauté scientifique, peu d'études s'intéressent au problème de la segmentation LS.

Finalement, le dataset utilisé dans ce mémoire est LSFB_CONT [6]. Celui-ci est un des plus grands datasets de vidéos continues en langue des signes au monde. Il est également directement lié au corpus LSFB [5], et est sans contraintes de vocabulaire ou de vitesse d'exécution. Ces éléments permettent, respectivement, la consultation d'une expertise quant à la LSFB ainsi qu'une réalisation d'un système de segmentation LS plus robuste.

Chapitre 3

État de l'art dans les modèles séquentiels

Afin d'implémenter un système de segmentation automatique de la langue des signes (segmentation LS), ce mémoire utilise des modèles séquentiels et plus spécifiquement des réseaux neuronaux récurrents (RNNs). Afin d'aborder ce concept, ce chapitre introduit des notions nécessaires dans l'ordre suivant : le machine learning, le deep learning et finalement les modèles séquentiels.

3.1 Le machine learning

Le machine learning est une branche d'un domaine plus large, l'intelligence artificielle. À l'origine, afin de constituer des systèmes capables de prendre des décisions, on faisait appel à des experts. Cette consultation avait pour but de créer manuellement des algorithmes capables de résoudre des tâches avec l'expertise apportée par les spécialistes du domaine dans lequel le système serait utilisé (systèmes experts) [24]. Cependant, constituer de tels systèmes s'avère très coûteux. En effet, des experts ne sont pas forcément disponibles, le monde de l'informatique ne leur est pas forcément familier et ceux-ci peuvent également se contredire. Constituer un système de décision de la sorte est donc devenu compliqué. Une nouvelle approche a donc émergé. Il s'agit de dériver des règles à partir de données. Ces systèmes capables d'effectuer une telle tâche sont appelés algorithmes de machine learning [25]. Les règles apprises, elles, sont appelées un *modèle*.

Dans ce mémoire, la notion de modèle est fondamentale. Il peut être défini comme un outil détectant des régularités dans un ensemble de données (dataset) afin de les distinguer. Dans le cas d'un algorithme de machine learning dit classique, ces données sont généralement tabulaires. Les tâches que peuvent résoudre les algorithmes de machine learning sont des prédictions de différentes natures. Celles-ci sont la classification (ex. classifier des chiens et des chats), la régression (ex. prédire le prix d'un voyage en taxi) ou le clustering (ex. regrouper entre eux les utilisateurs se ressemblant). Pour ce faire, le modèle reçoit des données, appelées instances ou observations. Dans le cas d'un tableau, il s'agit

de lignes. Ensuite, il réalise une prédiction à partir de celles-ci. La Figure 3.1 représente le fonctionnement d'un modèle.

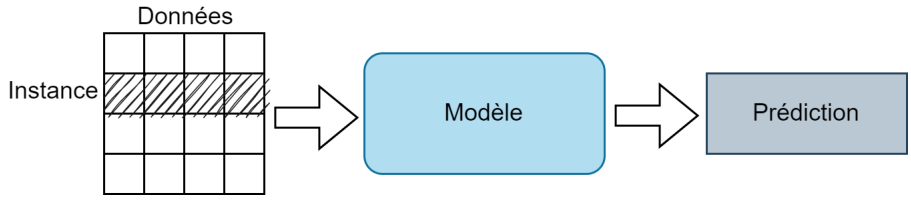


FIGURE 3.1 – Une prédiction en machine learning.

Une notion à éclaircir est la notion d'apprentissage. Le comportement d'un modèle est caractérisé par ses paramètres. Ceux-ci définissent quelle prédiction réalise le modèle à partir d'une certaine observation. En machine learning, l'apprentissage supervisé consiste en un processus réalisé automatiquement à travers un algorithme d'apprentissage. Celui-ci utilise un ensemble de données (dataset d'entraînement) dont les valeurs cibles, c'est-à-dire celles que l'on veut prédire, sont connues à l'avance. Dès lors, les prédictions du modèle sont comparées à ces valeurs cibles afin de mieux ajuster les paramètres de celui-ci. Cette comparaison s'effectue avec une fonction qui calcule à quel point le modèle se trompe. Elle est appelée *loss function*. La Figure 3.2 représente les composants essentiels à l'apprentissage d'un modèle.

À ce stade, deux ensembles de données (datasets) distincts sont utilisés. Le premier consiste en un dataset d'entraînement. Celui-ci possède des valeurs cibles par rapport auxquelles le modèle compare ses prédictions et ajuste ses paramètres. Ensuite, le dataset de test, lui, a un rôle d'évaluation. C'est sur celui-ci que la performance du modèle est évaluée (voir métriques, section 3.4). Dans un dataset classique, les instances sont considérées indépendantes entre elles. La section 3.3 traite de datasets dont les observations sont séquentielles et, donc, dépendantes entre elles.

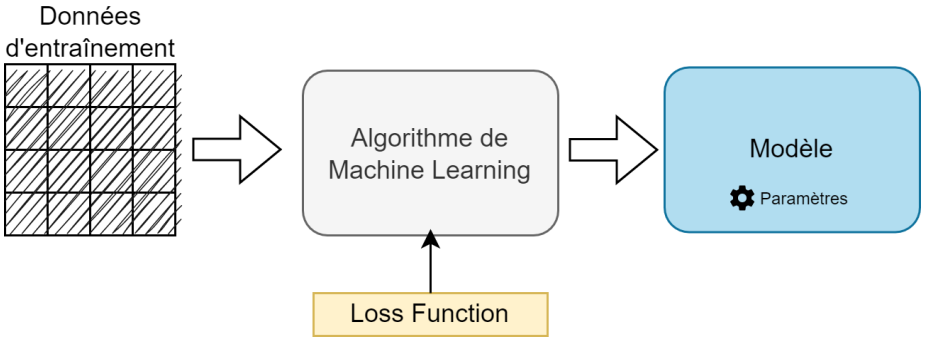


FIGURE 3.2 – Exemple d'apprentissage supervisé.

Un deuxième point à éclaircir est la nature des prédictions réalisées par le modèle. Dans le cas de la classification, chaque valeur cible (classe) est associée à un label. Des exemples de labels sont : *chien* et *chat*. La prédiction du modèle est donc l'appartenance d'une observation à une classe à laquelle est associé un

label. Un exemple est une prédiction où l'observation est une suite de valeurs relatives à un animal tandis que la prédiction est le niveau d'appartenance de cet animal aux classes dont les labels sont *chien* ou *chat*. Une prédiction peut également être une valeur numérique, par exemple le prix d'un voyage en taxi. Dans ce cas, l'algorithme de machine learning effectue une tâche de régression. Dans le cadre de ce mémoire, seules des tâches de classification sont réalisées.

3.1.1 Un exemple d'apprentissage supervisé

Afin d'assurer une meilleure compréhension des concepts abordés précédemment, un exemple simpliste d'apprentissage supervisé est présenté. Celui-ci est tiré du livre de Bishop, *Pattern Recognition and Machine Learning* [26]. À des fins pédagogiques, cet exemple présente une tâche de régression au lieu d'une tâche de classification. Cependant, les concepts abordés dans cet exemple sont également valables dans le cadre de la classification.

Soit un ensemble de données d'entraînement :

$$\mathbf{X} \equiv (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$$

où $\mathbf{x}_i, 1 \leq i \leq N$ sont des instances.

Soit les valeurs cibles associées à ces instances :

$$\mathbf{T} \equiv (t_1, \ldots, t_N)^T.$$

Si $y(\mathbf{x}_i)$ est la prédiction (régression) réalisée par un modèle, celle-ci doit idéalement être proche de la valeur cible t_i : $y(\mathbf{x}_i) \approx t_i$.

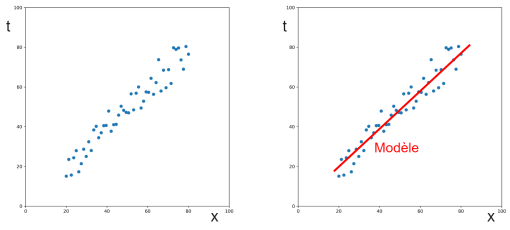


FIGURE 3.3 – Exemple de modèle de régression linéaire [26].

On émet l'hypothèse selon laquelle un modèle linéaire (voir Figure 3.3) modéliserait bien la relation entre $\mathbf{X}$ et $\mathbf{T}$. Ce modèle est défini comme suit :

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d = \sum_{j=0}^{d} w_j x_j$$

où d est la dimension de l'instance et le vecteur $\mathbf{w}$ constitue les paramètres du modèle, appelés poids. La loss function utilisée dans cet exemple est l'erreur quadratique moyenne (EQM). Elle mesure la somme des écarts (au carré) entre les valeurs cibles et les prédictions. Plus cette somme est faible, plus les

prédictions sont proches des valeurs cibles :

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=0}^{N}(t_i - y(\mathbf{x}_i, \mathbf{w}))^2$$

L'objectif est donc de définir les paramètres optimaux $\mathbf{w}^*$ tels que la loss $E(\mathbf{w})$ soit minimisée. L'équation pour les calculer est la suivante [26] :

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}) = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T}$$

Dans cette situation, il existe une solution analytique. Cependant, ce n'est pas toujours le cas. D'autres méthodes d'optimisation des paramètres doivent donc être utilisées. Un exemple courant d'une telle méthode (voir Figure 3.4) est la descente de gradient [27]. Il s'agit d'une méthode itérative qui consiste en la modification des poids en fonction du gradient de la loss function. Cela permet de converger vers un minimum local de la loss function. Comme la EQM est convexe, il s'agit d'un minimum global. Dans ce cas, les paramètres $\mathbf{w}$ convergent donc vers les paramètres optimaux $\mathbf{w}^*$.

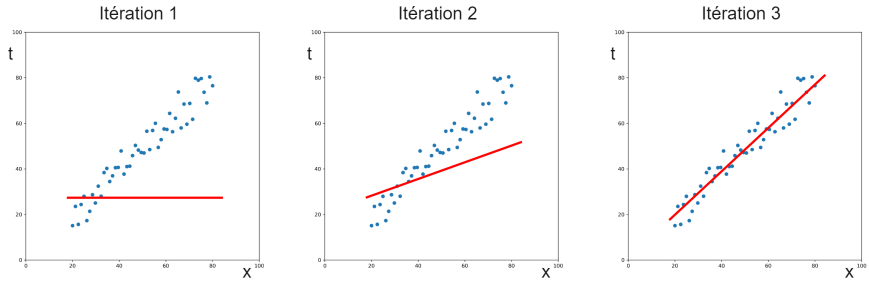


FIGURE 3.4 – Méthode itérative de la descente de gradient

Une fois le modèle entraîné. Celui-ci peut maintenant être utilisé afin de prédire la valeur cible d'une instance $\mathbf{x}$ n'ayant jamais été observée.

Afin de rigoureusement évaluer les performances du modèle, un dataset de test doit être utilisé avec des métriques appropriées (voir section 3.4). Cette étape n'est pas présentée dans l'exemple, mais est systématiquement réalisée dans les expériences dans la suite du mémoire (chapitres 5, 6 et 7).

3.1.2 La malédiction de la dimension

Cette section introduit brièvement une problématique liée à la dimension des instances, c'est-à-dire le nombre de colonnes dans les données. Plus la dimension d'une instance est importante, plus le volume de l'espace dans lequel repose cette instance est grand. Cela signifie qu'une instance aura plus tendance à se retrouver isolée dans un espace en hautes dimensions.

Les conséquences sont que les instances sont donc plus éloignées les unes des autres au sein de ce volume. Il est donc plus difficile pour le modèle de les mettre en relation.

Afin d'améliorer les performances d'un modèle, il est donc courant de se restreindre aux composantes (features) essentielles de l'instance. Cela revient à diminuer le nombre de colonnes dans les données. Pour cela, il est nécessaire d'évaluer le rôle de chaque feature dans le résultat final du modèle. Cela peut se faire en utilisant des méthodes d'interprétabilité de modèles. Celles-ci sont abordées au chapitre 8.

3.2 Le deep learning

Le deep learning constitue un sous-ensemble des modèles de machine learning. Ceux-ci sont appelés réseaux de neurones. Afin d'appréhender cette notion, un neurone isolé est décrit à travers le perceptron, section 3.2.1, pour ensuite décrire un réseau de neurones, le multi-layer perceptron, section 3.2.2.

De nombreuses architectures de réseaux de neurones existent, dont les réseaux récurrents (voir section 3.3.2) qui sont utilisés comme modèles pour la segmentation LS (voir chapitres 5, 6 et 7).

3.2.1 Le perceptron

Afin de comprendre le fonctionnement des réseaux de neurones, il est nécessaire de s'intéresser d'abord à un neurone isolé. Celui-ci est appelé un perceptron [28] et constitue à lui seul un modèle de machine learning. En effet, le perceptron, ou single layer perceptron, peut être entraîné pour des tâches de régression ou de classification.

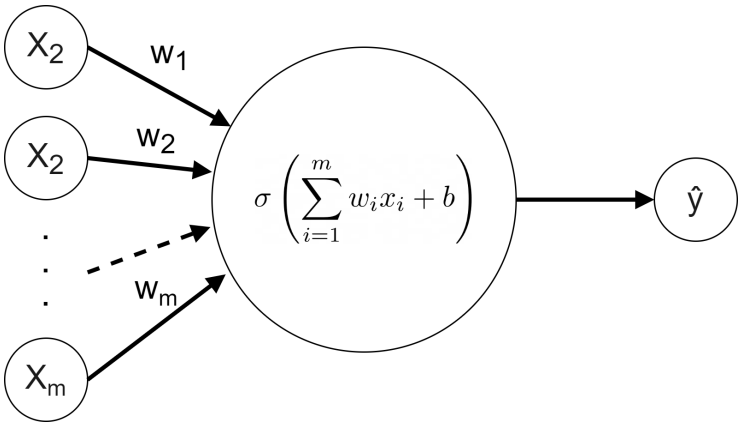


FIGURE 3.5 – Schéma d'un perceptron.

Un perceptron possède des entrées $x_1, x_2, \ldots, x_m$, des paramètres que sont les poids $w_1, w_2, \ldots, w_m$ ainsi que le biais b, et finalement une sortie y. Entre les entrées et la sortie se trouve un noyau central. Celui-ci applique une fonction σ, appelée fonction d'activation, à une combinaison linéaire des entrées avec leurs poids respectifs.

Au sein d'un perceptron, une prédiction est donc réalisée en traversant le modèle de gauche à droite, depuis l'entrée jusqu'à la sortie. Ensuite, si l'on est dans une

phase d'entraînement, les poids $w_1, w_2, \ldots, w_m$ sont mis à jour en fonction de leur responsabilité individuelle dans l'erreur moyenne commise par le neurone mesurée par la loss function. L'entraînement est itératif et utilise des méthodes telles que la descente de gradient ou la descente de gradient stochastique [27].

Afin de construire un perceptron, il est nécessaire de définir le nombre de données en entrées mais également une fonction d'activation, par exemple, la fonction sigmoïde : $\sigma(a) = 1/(1 + e^{-a})$.

3.2.2 Les réseaux de neurones

Un perceptron seul n'est pas capable de représenter n'importe quelle fonction continue [28]. En effet, celui-ci a un pouvoir de modélisation assez limité. Cependant, il est possible de combiner plusieurs perceptrons pour en faire un réseau de neurones. Cela permet de constituer des modèles plus puissants appelés multi layer perceptrons (MLPs) [29]. Ceux-ci permettent d'approximer des fonctions arbitrairement complexes [29].

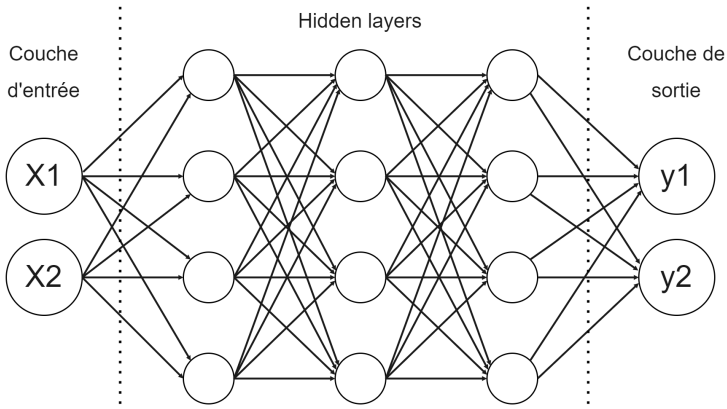


FIGURE 3.6 – Schéma d'un multilayer perceptron.

Tel que représenté dans la Figure 3.6, un MLP possède une couche d'entrée, une couche de sortie et un nombre arbitraire de couches intermédiaires appelées hidden layers. Chaque noeud est appelé neurone et est connecté à l'ensemble des noeuds de la couche suivante. Cela forme un réseau de neurones de la famille des fully connected artificial neural networks (fully connected ANNs). Un tel réseau de neurones forme un graphe acyclique dans lequel l'information navigue dans une seule direction (feed-forward neural network). Les MLPs sont la catégorie de réseaux de neurones artificiels (ANNs) la plus classique. Cependant, bien d'autres architectures de réseaux de neurones existent. Deux exemples connus sont les convolutional neural networks (CNNs) [30] et les recurrent neural networks (RNNs) [31]. Les RNNs sont décrits dans la section 3.3.2.

Pour réaliser une prédiction, un MLP reçoit des données en entrée, celles-ci traversent le réseau jusqu'à obtenir les valeurs de sortie. Cette traversée de gauche à droite (Figure 3.7) est appelée forward propagation. Ensuite, la mise à jour des poids s'effectue dans le sens inverse (Figure 3.8) en démarrant de la sortie et en remontant le réseau tout en mettant à jour les poids de celui-ci

en fonction de leur responsabilité dans l'erreur mesurée par la loss function. Cela s'appelle la backpropagation et est réalisé grâce à l'algorithme du même nom [32].

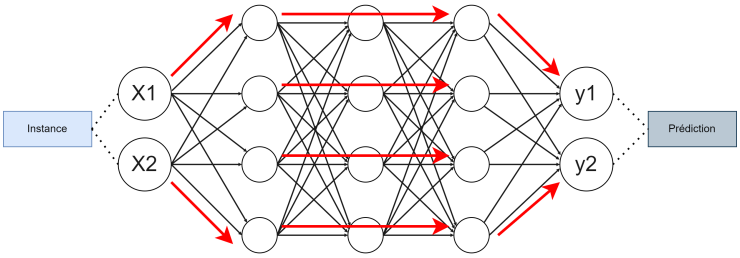


FIGURE 3.7 – Forward propagation dans un fully connected ANN.

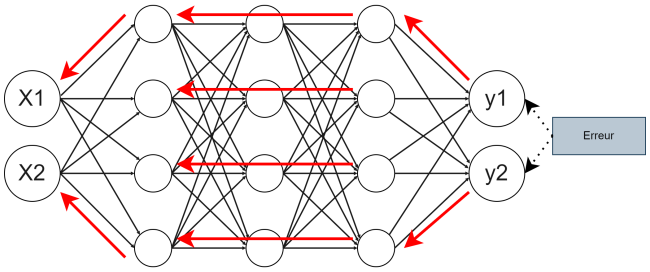


FIGURE 3.8 – Backpropagation de l'erreur dans un fully connected ANN.

Les ANNs sont des modèles qui nécessitent un nombre important d'itérations afin d'être entraînés. Cependant, l'explosion du volume de données accumulées [33] et l'explosion de la puissance de calcul des ordinateurs en font des modèles extrêmement populaires [30]. Ceux-ci se sont avérés être des modèles très puissants. Un exemple est le modèle AlexNet [34] qui a battu les anciens records dans le concours de classification d'images ImageNet en 2012 [35]. La puissance de modélisation des réseaux de neurones en fait des modèles très génériques. Cette généricité liée à l'ajout de multiple hidden layers forme une nouvelle branche de l'intelligence artificielle nommée le **deep learning**.

3.3 Les modèles séquentiels

La segmentation automatique d'une vidéo en langue des signes (segmentation LS) peut être vue comme une tâche de prédiction pour chaque instant de la vidéo. Or, une vidéo est fondamentalement une succession d'images appelées frames. Pour tout instant t de la vidéo, la frame associée est notée F_t.

La prédiction associée à cette une frame F_t est notée $y(F_t) \in \{0, 1, 2\}$. Si $y(F_t) = 0$, cela signifie que le signeur est en train d'effectuer un signe, si $y(F_t) = 1$, le signeur est en train de faire une transition entre deux signes et si $y(F_t) = 2$, le signeur est inactif. On obtient donc un problème de classification ternaire avec 3 classe (0, 1, et 2), associées respectivement à trois labels : *signing*, *coarticulation* et *waiting*. Une exception est l'expérience chapitre 5 dans laquelle un problème de classification binaire est considéré en fusionnant les classes *coarticulation* et *waiting*.

Le machine learning a pour objectif d'approximer cette fonction de classification $y(F_t)$ à travers un modèle. Celui-ci, dans ce cas de figure, est entraîné à partir des données que sont les frames extraites des vidéos, soit environ 16 millions de frames [6] pour tout le corpus LSFB [5, 6].

Dans un modèle de classification classique (voir section 3.1), les observations sont considérées indépendantes entre elles. Dans notre cas, l'état d'une frame F_t est dépendante des autres frames. En effet, si le signeur effectue un signe à l'instant t, il est vraisemblable que celui-ci le poursuive à l'instant $t+1$. L'aspect séquentiel de la vidéo est donc une propriété dont le modèle devrait tenir compte. Sinon, cela voudrait dire que la nature séquentielle des vidéos, dans ce cas une nature temporelle, est ignorée.

Afin de pallier à ce problème, des modèles dits *séquentiels* [36] ont été développés. Ceux-ci ne font pas cette hypothèse d'indépendance entre les observations. Dans le cas de la segmentation des vidéos en langue des signes, la décision associée à une frame est donc $y(F_t|F_{t-1}, F_{t-2}, \ldots, F_1)$ soit la décision pour la frame à l'instant t sachant les prédictions pour les frames précédentes F_1 à F_{t-1}.

3.3.1 Approches statistiques

Une grande famille de modèles séquentiels en machine learning sont des processus stochastiques [37]. Ceux-ci ont un gros bénéfice : ils se basent sur des fondements théoriques solides que sont les statistiques. En effet, ces modèles manipulent des variables aléatoires qui représentent des concepts précisément définis.

Il existe de nombreux modèles séquentiels basés sur des approches statistiques. Les plus connus sont les Hidden Markov Models (HMM) [38]. Il en existe d'autres tels que les Autoregressive Moving Average (ARMA) [39] ou encore les Conditional Random Fields (CRF) [40], mais ils ne seront pas abordé dans la suite de ce mémoire. Une extension du HMM, le Hidden semi-Markov Model (HSMM) [41] est brièvement décrite en section 3.3.1. Les concepts relatifs aux HMMs et HSMMs sont utilisés dans l'expérience du chapitre 6 ainsi que dans l'article soumis à la conférence internationale ESANN (voir annexes).

HMM

Les Hidden Markov Models (HMMs) [38] sont parmi les premiers modèles séquentiels. Les HMMs sont populaires dans des tâches telles que la reconnaissance vocale [42], le traitement de séquences ADN [43], etc. [44]. Un des avantages des HMMs est que le modèle est facile à comprendre.

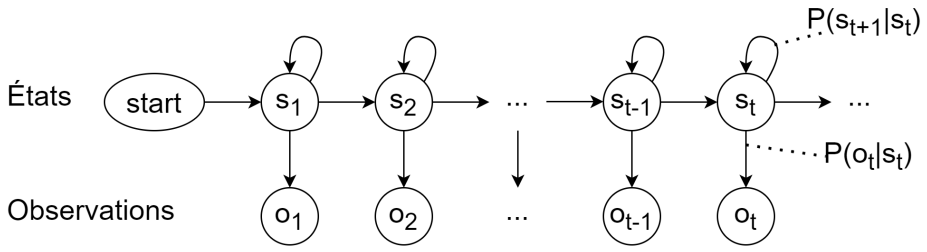


FIGURE 3.9 – Hidden Markov Model.

On retrouve deux séquences dans un HMM :

1. une séquence d'états cachés : $s_1, s_2, \ldots, s_{t-1}, s_t$;
2. une séquence d'observations : $o_1, o_2, \ldots, o_{t-1}, o_t$.

Les états cachés ne sont pas directement observables. C'est-à-dire qu'ils sont inconnus au moment où la séquence (par exemple une vidéo LS) est observée. Dans notre cas, l'état $s_t \in \{0, 1, 2\}$ représente la prédiction $y(F_t)$ à l'instant t. Les observations, quant à elles, sont directement observables et sont corrélées aux états cachés. Leur nature peut varier, mais dans l'exemple présent il s'agit d'une information relative aux frames (squelette ou position des mains, voir chapitre 4).

À un instant t d'une séquence, le HMM distingue deux probabilités. En premier lieu, la probabilité de transition d'un état vers un autre $P(s_{t+1}|s_t, s_{t-1}, \ldots, s_1)$ et ensuite la probabilité d'émission d'une observation à partir de l'état à l'instant courant $P(o_t|o_{t-1}, \ldots, o_1, s_t, s_{t-1}, \ldots, s_1)$. On peut également distinguer la probabilité initiale des états, π, pour l'état s_1. La Figure 3.9 représente ces probabilités par des flèches.

Deux hypothèses simplificatrices sont formulées dans les HMMs :

1. **l'hypothèse de Markov** : La probabilité de l'état courant ne dépend que de l'état précédent, soit la probabilité de transition $P(s_{t+1}|s_t)$;
2. **l'indépendance des observations** : La probabilité d'une observation o_t ne dépend que de l'état s_t, soit la probabilité d'émission $P(o_t|s_t)$.

À partir de cette formulation des HMMs, il est possible de calculer la meilleure séquence d'états cachés (la séquence la plus vraisemblable) avec un algorithme dit de décodage tel que l'algorithme de *Viterbi* [45] ou l'algorithme de *Baum-Welch* [46]. Dans le cadre de ce mémoire, trouver une telle séquence revient donc à segmenter la vidéo étant donné qu'un état caché représente l'état du signeur à l'état t et que l'observation associée est l'information de la frame concernée.

HSMM

Dans un HMM, la durée d'un état au sein d'une séquence est implicitement modélisée par une distribution géométrique/exponentielle [38] (voir Figure 3.10). En effet, la probabilité de rester dans l'état courant ne dépend que de l'état précédent. Il s'agit de l'hypothèse de Markov. Or, une distribution exponentielle de la durée des états favorise les états courts. Il arrive que cette modélisation soit insuffisante et qu'une autre distribution corresponde mieux à la durée des états (par exemple une distribution gamma, voir Figure 3.10).

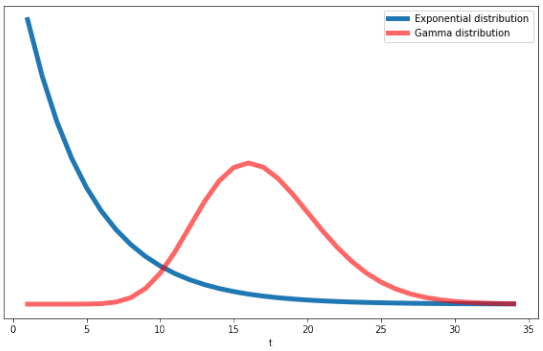


FIGURE 3.10 – Comparaison entre une distribution exponentielle et une distribution gamma.

Les Hidden semi-Markov Model (HSMM) [41] sont une extension des HMMs relaxant l'hypothèse de Markov. Dans cette extension, la durée des états est explicitement modélisée. Dès lors, un état possède une variable aléatoire modélisant sa durée ainsi que plusieurs observations, voir Figure 3.11.

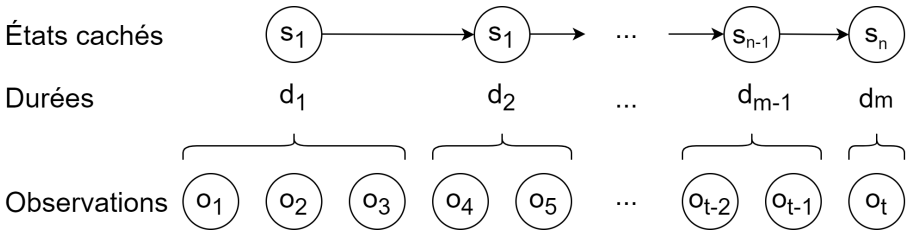


FIGURE 3.11 – Hidden semi-Markov Model.

Dans un HSMM, notre séquence d'observations devient une ligne du temps. La séquence d'états, quant à elle, n'est plus de la même taille que notre séquence d'observations. En effet, dans un HSMM, un état possède une durée et durant celui-ci, plusieurs observations sont émises dépendamment de sa durée associée. La durée d est une variable aléatoire discrète telle que : $d \in \{1, 2, \ldots, D\}$. La transition d'un état i vers un état j devient donc la transition d'un état i à la durée d vers un état différent j à la durée d'. Soit la transition : $(i, d) \rightarrow (j, d')$ avec $i \neq j$. Par exemple, dans la figure 3.11, durant l'état s_1, 3 observations sont émises aux temps t_1, t_2 et t_3. La transition vers l'état s_2 s'effectue quand $d_1 = 3$, soit $(s_1, 3) \rightarrow (s_2, 1)$.

On définit l'état d'un instant t à un instant k comme étant $s_{t:k} = s_t, s_{t+1}, \ldots, s_k$. La probabilité de transition, quant à elle, devient :

$$a_{(i,d)(j,d')} = P(s_{t+1:t+d} = j | s_{t-d'+1:t} = i)$$

où cette probabilité de se retrouver dans un état à une certaine durée dépend dans quel état on se situe avant la transition et depuis combien de temps.

De façon similaire, les observations d'un instant t à un instant k sont notées : $o_{t:k} = o_t, o_{t+1}, \ldots, o_k$. La probabilité d'émission, dans le HSMM, dépend de l'état courant, mais également de la durée de cet état. On a donc :

$$b_{i,d}(o_{t+1:t+d}) = P(o_{t+1:t+d} | s_{t+1:t+d} = i)$$

où il s'agit de la probabilité d'observer une séquence d'observation d'une certaine durée suivant l'état dans lequel on est et depuis combien de temps.

Diverses hypothèses simplificatrices peuvent être formulées au sein des HSMM [41]. Par exemple, on peut considérer que la distribution de la durée d'un état est indépendante de l'état précédent. Ensuite, de façon générale, le calcul de la séquence la plus probable utilise des algorithmes similaires à ceux employés pour les HMM.

3.3.2 Réseaux récurrents

Bien que les modèles séquentiels basés sur des approches statistiques sont encore régulièrement utilisés dans la littérature [11], dans le cas de la reconnaissance ou de la segmentation de vidéos en langue des signes, des modèles plus adaptés existent [2] tels que les recurrent neural networks (RNN) [31]. Ces réseaux neuronaux sont largement utilisés dans le cas de la langue des signes [1].

Dans le cas de la segmentation des signes, une vidéo est une séquence de n frames. L'objectif du mémoire est de définir un label pour chaque frame. Dès lors, il s'agit de prédire une séquence à partir d'une séquence d'entrée. Une modèle prédisant une séquence à partir d'une séquence est appelé un modèle *seq2seq*. Afin d'uniformiser les notations dans la suite du mémoire, une séquence d'entrée est notée $x_1, x_2, \ldots, x_n$ tandis que la séquence prédite est notée $y_1, y_2, \ldots, y_n$.

RNN simple

Le réseau récurrent simple (simple RNN) [47, 48] est la base sur laquelle repose la famille des réseaux récurrents (RNN). Des réseaux récurrents plus puissants et plus utilisés existent, tels que les Long Short-Term Memory (LSTM) [12]. Cependant, ceux-ci sont des extensions des RNN simples. Afin de décrire le fonctionnement d'un RNN, il est utile de démarrer par les réseaux de neurones classiques que sont les MLP (voir section 3.2.2).

Afin de prédire une séquence, une approche envisageable est de passer chaque élément de la séquence d'entrée dans un réseau de neurones et de collecter les prédictions. La Figure 3.12 illustre ce scénario. Bien qu'une telle approche fonctionne, chaque élément de la séquence est considéré comme étant indépendant des autres. Dès lors, l'information séquentielle est ignorée et les performances du modèle en sont impactées.

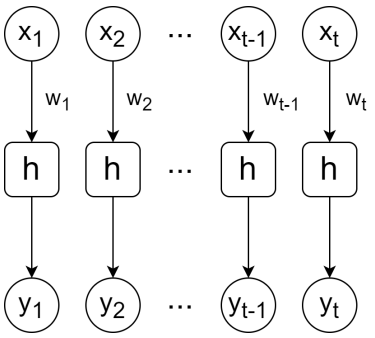


FIGURE 3.12 – Prédiction seq2seq à travers un réseau de neurones classique (MLP).

Comme décrit dans la section 3.3, afin d'avoir de meilleures performances, un modèle séquentiel devrait tenir compte des dépendances entre les éléments au sein d'une séquence. À cette fin, les RNNs sont des réseaux de neurones classiques auxquels sont ajoutées des connexions dites récurrentes (voir Figure 3.13). Ces nouvelles connexions lient les hidden layers du réseau à elles-mêmes et possèdent des paramètres (poids) **U** en supplément des poids des connexions classiques **W**. De la même façon que les poids **W**, les poids **U** sont mis à jour lors de la phase d'entraînement. Dès lors, **U** est mis à jour à chaque passage d'un élément de la séquence. De cette façon, la prédiction d'un élément x_t utilise des paramètres entraînés sur les observations $x_{t-1}, x_{t-2}, \ldots, x_1$. Une hidden layer avec des connexions récurrentes est appelée recurrent layer. Un réseau récurrent tire donc parti de l'aspect séquentiel des données et fournit donc de meilleures prédictions.

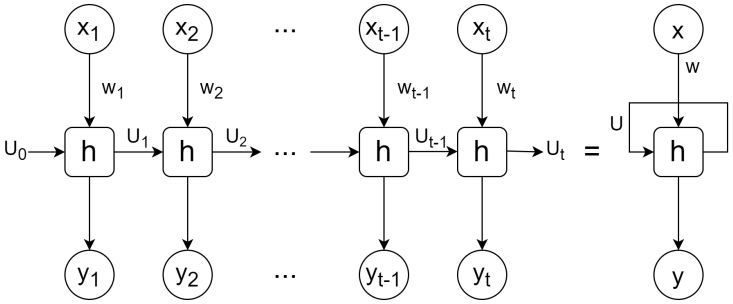


FIGURE 3.13 – Prédiction seq2seq à travers un réseau de neurones récurrents (RNN). Représentation déroulée à gauche et représentation enroulée à droite.

Un exemple de RNN simple est le réseau de Elman [48] où un noyau récurrent calcule un vecteur intermédiaire avant de calculer la sortie du noyau. Ce vecteur intermédiaire h_t est appelé hidden vector. Une recurrent layer du réseau de Elman est définie comme suit :

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

où

- x_t est l'élément la séquence d'entrée à l'instant t ;
- y_t est la prédiction à l'instant t ;
- h_t est l'hidden vector à l'instant t ;
- W_h, U_h, b_h, W_y, et b_y sont les paramètres du modèle ;
- σ_h et σ_y sont des fonctions d'activation.

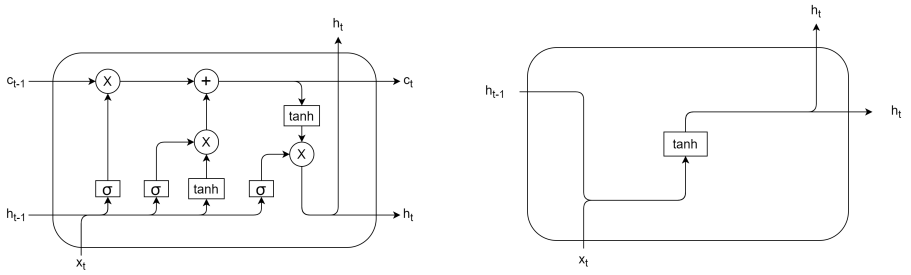
Le réseau de Elman souffre de limitations qui dégradent ses performances, notamment en ce qui concerne les dépendances entre des données éloignées au sein d'une séquence. Dès lors, un autre RNN plus puissant est privilégié dans la segmentation LS, le LSTM [12]. Celui-ci est une extension du réseau de Elman et est décrit dans la section 3.3.2.

LSTM

Les RNNs ont des limitations [12]. Premièrement, les RNNs simples subissent le problème des vanishing gradients. Il s'agit d'une conséquence du fait que l'entraînement d'un réseau de neurones utilise des méthodes basées sur le gradient [27, 32]. En effet, dans ces méthodes, l'actualisation des poids est réalisée à partir de la dérivée partielle de la loss function par rapport aux poids aux différents endroits dans le réseau. Or, il est possible que ces dérivées soient de plus en plus petites au fur et à mesure que l'on remonte dans les couches précédentes. Cela a comme conséquence que la valeur des poids ne se met plus à jour lorsque les dérivées sont quasiment nulles. Plus un réseau possède de couches, plus le phénomène est important. Les réseaux récurrents sont particulièrement sujets à ce problème. En effet, les connexions récurrentes prolongent artificiellement la longueur du réseau.

Une seconde limitation est le fait qu'un RNN simple peut avoir du mal à modéliser un décalage temporel entre la séquence d'entrée et la séquence de sortie [12, 48]. Or, un tel décalage peut apparaître dans les données. De façon plus générale, plus deux éléments d'une séquence sont éloignés, plus le RNN a du mal à modéliser les relations entre ceux-ci.

Une amélioration des RNNs simples est l'architecture Long short-term memory (LSTM) [12]. Les LSTMs sont supérieurs aux RNNs simples en plusieurs points. Premièrement, ceux-ci atténuent le problème des vanishing gradients et sont donc plus rapides à entraîner. Ensuite, le calcul du hidden vector est plus complexe au sein d'une cellule LSTM qu'au sein d'une cellule de RNN simple. La cellule LSTM est représentée Figure 3.14a, tandis que la cellule d'un réseau de Elman est représentée Figure 3.14b à titre de comparaison. Le LSTM possède des connexions supplémentaires (feedback connections), des portes logiques (gates) ainsi qu'un hidden vector supplémentaire noté c_t et appelé le vecteur d'état de la cellule (cell state vector). Ces ajouts permettent de filtrer l'information utilisée, de partiellement mettre à jour les hidden states, etc. Cela permet une modélisation plus sélective et une meilleure représentation temporelle sur de longues séquences.



(a) Représentation d'une cellule LSTM avec des feedback connections et des gates.

(b) Représentation d'une cellule d'un réseau de Elman.

FIGURE 3.14 – Représentation de cellules RNN

Une description de la cellule LSTM, plus précise que sur la figure 3.14a, consiste en la définition mathématique de la mise à jour des hidden vectors :

$$
\begin{aligned}
f_t &= \sigma(\mathbf{W}_t x_t + \mathbf{U}_f h_{t-1} + b_f) \\
i_t &= \sigma(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i) \\
o_t &= \sigma(\mathbf{W}_o x_t + \mathbf{U}_o h_{t-1} + b_o) \\
\widetilde{c}_t &= \tanh(\mathbf{W}_c x_t + \mathbf{U}_c h_{t-1} + b_c) \\
c_t &= f_t \times c_{t-1} + i_t \times \widetilde{c}_t \\
h_t &= o_t \times \tanh(c_t)
\end{aligned}
$$

où

- $\times$ représente le produit entre éléments, ou produit d'Hadamard [49] ;
- σ représente la fonction sigmoïde ;
- f_t, i_t, et o_t s'appellent les vecteurs d'activation pour, respectivement, la forget gate, l'input gate et l'output gate ;
- c_t est le cell state vector ;
- h_t est le hidden state vector qui est aussi la sortie de la cellule.

Dans ce mémoire, les LSTMs sont les réseaux récurrents qui sont utilisés afin d'effectuer la segmentation automatique des signes dans les vidéos en langue des signes. Des extensions des LSTMs sont également abordées dans l'expérience sur une meilleure modélisation des transitions d'états dans le chapitre 6.

3.4 Les métriques

Afin d'évaluer les performances de la segmentation LS, des métriques sont nécessaires. Cette section décrit différentes métriques [50] utilisées pour évaluer les techniques de machine learning du mémoire. Étant donné que la segmentation LS est une tâche de classification, les métriques utilisées ne concernent que celle-ci et non la régression ou le clustering. Ces métriques sont : l'accuracy, le recall et la balanced accuracy.

3.4.1 L'accuracy

Une métrique populaire en machine learning est l'accuracy [50]. Celle-ci est le taux de prédictions correctes réalisées par le modèle :

$$\text{accuracy} = \frac{\text{\# prédictions correctes}}{\text{\# total de prédictions}}.$$

Cette métrique est représentative des performances du modèle seulement si les valeurs cibles sont uniformément réparties au sein du dataset de test. Par exemple, dans le cas de la segmentation on aurait 33.33% de frames avec le label *signing*, 33.33% avec le label *coarticulation* et 33.33% avec le label *waiting*. Un dataset respectant une distribution uniforme des labels est dit **balancé**.

Cependant, dans les expériences de ce mémoire, les données sont fortement **non-balancées** (voir chapitre 5). Dès lors, l'accuracy n'est pas suffisante pour évaluer correctement le modèle. En effet, dans un cas extrême où 99% des données appartiennent à la même classe, un modèle pourrait atteindre une accuracy de 99% en ne prédisant que cette classe. D'autres métriques (par exemple la balanced accuracy, voir section 3.4.3) sont nécessaires afin de pallier au non-balancement des données. Le chapitre 5 traite, plus en détail, de la problématique du non-balancement des données dans le cadre de la segmentation LS.

3.4.2 Le recall

Le recall est une métrique définie comme étant le taux de true positives (TP) pour une classe donnée [50]. Par exemple, soit une classe possédant la valeur cible c, le nombre de TP pour cette classe est le nombre de prédictions correctes telles que les instances prédites ont c comme valeur cible. En ce qui concerne le recall, il s'agit du taux de TP par rapport au total de prédictions :

$$\text{recall}(c) = \frac{\text{\# true positives pour la classe } c}{\text{\# total de prédictions}}.$$

Par exemple, dans le cas de données fortement non-balancées avec 95% de labels à 0 et 5% à 1, si le modèle prédit à chaque fois 0, l'accuracy est de 95%, le recall de la classe 0 est de 100% tandis que le recall de la classe 1 est de 0%. En effet, le modèle a des performances parfaites pour prédire le label 0 mais est très peu performant pour le label 1. Le recall est donc important dans ce genre de situation.

3.4.3 La balanced accuracy

L'accuracy est un bon indicateur de performance si les données sont balancées [50]. Cependant, comme le montre l'exemple pour le recall, celle-ci est biaisée si les données sont non-balancées. Dans ce cas de figure la balanced accuracy peut être utilisée. Celle-ci consiste en la moyenne des recalls de chaque classe divisée par le nombre de classes. De cette façon, une bonne balanced accuracy est synonyme d'un modèle qui a de bonnes performances pour toutes les classes, quelles que soient leurs proportions au sein des valeurs cibles.

La balanced accuracy est définie comme suit :

$$\text{balanced accuracy} = \frac{\text{moyennes des recalls de toutes les classes}}{\text{nombre de classes}}.$$

Dans les expériences menées dans ce mémoire, la balanced accuracy est l'indicateur de performance préféré afin de comparer les modèles. Cependant, l'accuracy et les recalls sont également gardés comme métriques pertinentes afin de pouvoir interpréter plus en détail les résultats des modèles.

Chapitre 4

Création et traitement des datasets

Dans ce mémoire, la segmentation automatique des signes est réalisée à l'aide de techniques de machine learning (voir section 3.1). Il s'agit d'algorithmes dérivant les paramètres d'un modèle à partir de données. Ces données peuvent être de natures multiples et sont regroupées au sein d'ensembles de données (datasets). La section 4.2 présente de nouveaux datasets contenant des squelettes (points de repère) extraits à partir des images (chaque frame) des signeurs dans les vidéos du dataset LSFB_CONT (voir section 2.3). Ceux-ci sont les suivants :

- les articulations des mains (section 4.2.2) ;
- le squelette supérieur (section 4.2.1) ;
- le maillage du visage du signeur (section 4.2.3).

La sortie est simplifiée afin de correspondre au cas d'étude : la segmentation des signes (voir section 2.2). En effet, les valeurs cibles de nos modèles sont l'activité du signeur (*signing*, *waiting* et *coarticulation*). La Figure 4.1 montre un exemple de segmentation pour une vidéo quelconque. Le traitement des annotations et leur transformation en données exploitables sont décrits dans la section 4.1.

En machine learning, les performances des modèles dépendent fortement de la quantité, mais aussi de la qualité des données que ceux-ci utilisent lors de son entraînement. Dès lors, un soin particulier doit être apporté à la préparation des données. Il est nécessaire de les nettoyer et d'éliminer le plus d'éléments perturbateurs possible. Cette étape de préparation des points de repères est décrite dans la section 4.3.

Ce chapitre décrit plusieurs datasets dont certains ne sont pas utilisés dans les expériences décrites dans la suite du mémoire (chapitre 5, 6 et 7). Cependant, ceux-ci sont disponibles [1] [51] au profit de la recherche, pour de futurs travaux.

1. `https://pypi.org/project/lsfb-dataset/`

4.1 Traitement des annotations

Initialement, le dataset LSFB-CONT [6] met à disposition des vidéos en langue des signes accompagnées d'annotations au format ELAN [23]. Ces annotations sont extraites de ces fichiers et mises sous forme de données tabulaires dont les lignes fournissent chacune comme information :

1. la main utilisée pour faire le signe ;
2. un intervalle (en millisecondes) durant lequel l'annotation apparaît dans la vidéo (début, fin) ;
3. un identifiant du signe, appelé gloss. Par exemple, *CHANGER.DEVENIR* ;
4. une traduction en français de l'annotation (pas toujours présente).

Cependant, la segmentation des signes n'utilise pas l'information relative à la sémantique des signes. Dès lors, l'information effectivement utilisée, dans notre cas, ne concerne que la main utilisée (1) et l'intervalle (2).

Étant donné que les modèles réalisés font des prédictions pour chaque frame, une vidéo de n frames nécessite un vecteur de dimension n indiquant une valeur cible pour chaque frame. Celle-ci est typiquement 0, 1 ou 2 avec comme labels respectifs *signing*, *coarticulation* et *waiting* afin d'identifier l'activité du signeur à cette frame (voir chapitre 5). La présence d'un signe est donc une succession de 0. Ce vecteur est appelé **segmentation** et peut être représenté sur une ligne du temps, comme la Figure 4.1, où le noir représente la présence d'un signe et le blanc des moments où le signeur est inactif ou effectue une transition entre deux signes.

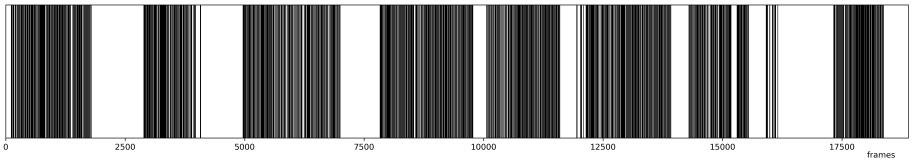


FIGURE 4.1 – Exemple de segmentation. Il s'agit d'une ligne du temps où pour chaque frame (axe x) le noir représente la classe *signing* (présence d'un signe) tandis que le blanc représente les classes *coarticulation* et *waiting* (absence de signes).

Lorsqu'une vidéo possède de nombreux signes consécutifs, il peut être difficile de les distinguer dans la segmentation. Les séparations entre ceux-ci (espaces blancs) sont parfois très courtes. C'est pourquoi une autre représentation de la segmentation distingue les signes consécutifs avec des couleurs différentes. Ces couleurs ne représentent aucune classe particulière. Leur seule utilité est de bien identifier les séparations entre les signes. La Figure 4.2 montre, avec des couleurs, la segmentation de la même vidéo que la Figure 4.1.

L'expérience du chapitre 5 nécessite d'isoler le phénomène de coarticulation entre les signes. Celui-ci est décrit dans l'état de l'art sur la segmentation des signes (voir section 2.2). Les annotations fournies par les fichiers ELAN ne distinguent pas les coarticulations entre les signes. C'est pourquoi une hypothèse

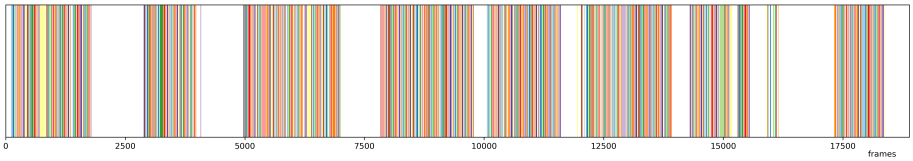


FIGURE 4.2 – Exemple coloré de segmentation. La segmentation est identique à celle de la Figure 4.1, mais avec des couleurs pour distinguer les signes entre eux.

est utilisée afin de distinguer la coarticulation des simples périodes d'inactivité : toute période d'inactivité (absence de signe) entre deux signes de moins d'une seconde est une coarticulation. Avec cette hypothèse, il est possible de labelliser les coarticulations à partir de la segmentation initiale. Il s'agit des espaces blancs de moins d'une seconde (moins de 50 frames dans le cas d'une vidéo en 50FPS) entre les signes. La Figure 4.3 représente la segmentation avec, en rouge, la coarticulation.

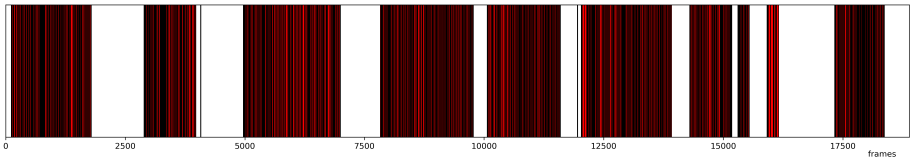


FIGURE 4.3 – Exemple de segmentation identique à celle de la Figure 4.1 où la *coarticulation* (en rouge) est distinguée de la classe *waiting*.

Les exemples de segmentation (voir Figures 4.1, 4.2 et 4.3) concernent une vidéo complète (plus de 18000 frames). Il est parfois utile de visualiser un segment plus court (voir Figure 4.4) de la vidéo afin de mieux distinguer les signes et la coarticulation.

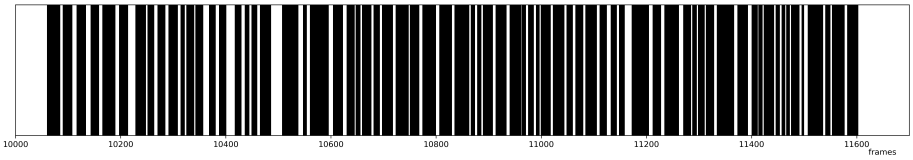


FIGURE 4.4 – Segmentation identique à la figure 4.1, où seules les frames 10000 à 11700 sont représentées.

En résumé, pour chaque vidéo, le traitement des annotations suit les étapes suivantes :

1. extraction des annotations depuis les fichiers ELAN ;
2. filtrage des données inutiles dans la segmentation LS ;
3. mise sous forme de segmentation (figure 4.1) ;
4. (optionnel) isolation de la coarticulation dans la segmentation.

4.2 Les repères

Le dataset LSFB_CONT (voir section 2.3) [6] fournit un ensemble de vidéos de discussions en langue des signes. Sur chacune de ces vidéos se trouve un unique signeur. Afin d'évaluer automatiquement si celui-ci parle ou non, il est supposé que seules les informations relatives au signeur sont pertinentes. Dans cette section, trois caractéristiques du signeur sont retenues :

1. la configuration des mains du signeur ;
2. la pose générale du signeur ;
3. l'expression du visage du signeur.

Afin d'extraire ces caractéristiques des vidéos, la librairie MediaPipe [52] est utilisée. Celle-ci fournit un ensemble de repères (voir Figure 4.5) :

1. les articulations des mains ;
2. le squelette, ou la pose ;
3. le visage.

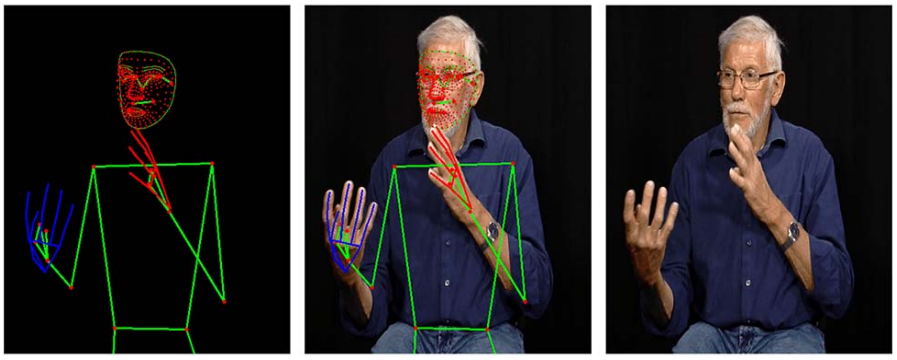


FIGURE 4.5 – Repères extraits avec MediaPipe [52].

Les informations des points de repères sont moins riches que les images brutes. Néanmoins, dans la segmentation LS, on fait l'hypothèse que les positions des différentes articulations du signeur peuvent être suffisantes pour les prédictions. Dans l'hypothèse où les points de repères sont pertinents et correctement positionnés, ils fournissent plusieurs avantages décrits ci-dessous.

Entraînement plus rapide

Entraîner un réseau de neurones est très gourmand en ressources. Cela est d'autant plus vrai lorsque l'on traite des images [30]. Utiliser des repères au lieu des images réduit la quantité d'information fournie au modèle et permet d'utiliser des architectures moins gourmandes en ressources et en énergie.

Moins biaisé par les couleurs

Dans la mesure où les points de repères sont correctement positionnés, les différences de contrastes, de couleurs ou d'arrière-plans n'ont pas d'incidence sur les performances du modèle. En effet, dans les repères, aucune de ces informations n'est retenue. Cependant, l'extraction des repères menée par MediaPipe [52] est toujours potentiellement impactée par ces contrastes ou changements de couleurs. Le problème que sont ces perturbations est donc délégué à l'extracteur de repères. Dans son code de conduite, MediaPipe est conscient de ces problématiques et alloue énormément d'efforts afin de diversifier les couleurs et arrière-plans afin de minimiser ces perturbations.

Plus éthique

Le dataset LSFB_CONT [6] et les autres datasets en langue des signes (voir section 2.3) ne représentent pas forcément les minorités. Par exemple, dans le cas où l'entraînement de la segmentation automatique des signes est réalisé directement à partir des vidéos du dataset LSFB_CONT, le modèle résultant sera biaisé par la sous-représentation des personnes de couleur de peau noire au sein du dataset. Utiliser un modèle intermédiaire tel que MediaPipe [52] permet de mitiger ce problème. En effet, MediaPipe est pré-entrainé avec comme objectif d'être inclusif (cf. code de conduite [52]).

4.2.1 Premier dataset : les mains

Le premier dataset réalisé concerne les mains. L'utilisation explicite des mains dans la langue des signes rend pertinente l'utilisation d'informations liées à celles-ci.

Ce mémoire se concentre essentiellement sur des données sous forme de repères (voir Figure 4.5). Cela induit un nouveau critère : le nombre de repères devant être utilisés afin de représenter un concept physique, dans ce cas, les mains. Ce paramètre influe sur le niveau de précision avec lequel les mouvements seront conservés. S'il y a trop peu de repères, cela induit une représentation trop simpliste des mains tandis que trop de repères dégradent les performance du modèle (voir section 3.1.2). Dans ce mémoire, afin d'assurer de bonnes performances du modèle, les repères ciblent particulièrement : les articulations de la main, les phalanges et la paume. Selon ces critères, la Figure 4.6 montre les différents repères sélectionnés et extraits à partir des vidéos, pour chaque frame et pour chaque main.

Une main possède donc 21 repères (voir Figure 4.6). Ceux-ci sont considérés en 2-dimensions (x et y). L'ensemble de l'information relative aux mains sauvegardée pour une vidéo de n frames est donc une séquence de longueur n où chacun de ses éléments consiste en 2 mains avec, respectivement, 21 coordonnées (x, y). Soit un tableau de n lignes et 84 colonnes. Le dataset complet consiste donc en tableaux au format *CSV* correspondant aux vidéos du dataset LSFB_CONT [6].

L'extraction des repères des mains utilise MediaPipe [52]. Bien que celui-ci fournisse généralement de très bons résultats, des imprécisions et des repères

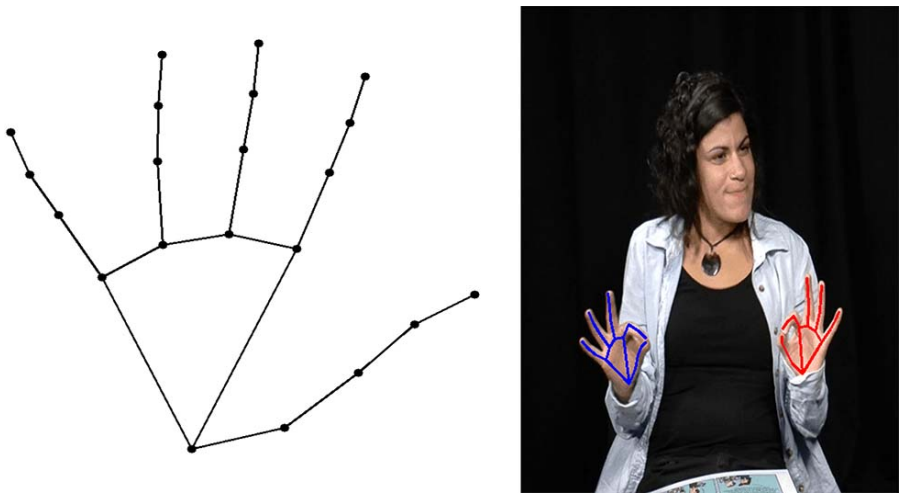
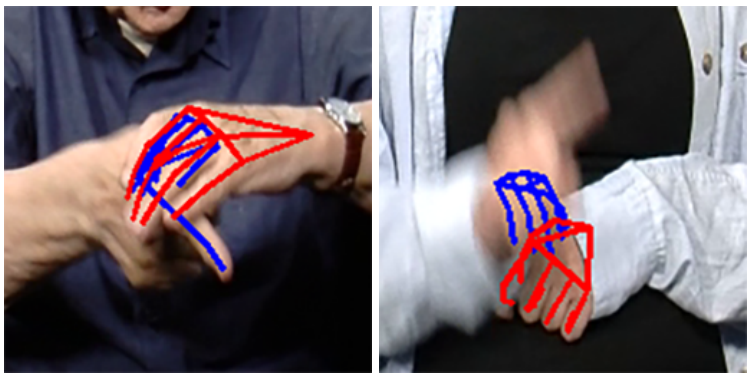


FIGURE 4.6 – Repères de la main.

manquants subsistent. Deux exemples de situations où la détection est plus susceptible d'échouer sont : lorsqu'une main obstrue l'autre (voir Figure 4.7a) et lorsqu'apparaît un flou de mouvement trop important (voir Figure 4.7b).



(a) Mains qui se superposent. (b) Flou de mouvement.

FIGURE 4.7 – Exemples de problèmes dans la détection des repères de la main.

Afin d'améliorer la qualité des données, une étape de post-traitement est appliquée. Celle-ci est similaire pour tous les datasets de ce mémoire. Elle est décrite dans la section 4.3 après que ceux-ci soient introduits.

4.2.2 Second dataset : le squelette supérieur

Un second dataset est construit après celui concernant les mains (voir section 4.2.1). Ce nouveau dataset se concentre sur la pose du signeur. En effet, l'hypothèse selon laquelle la pose du signeur est corrélée à son activité et aux signes qu'il réalise est étudiée dans l'expérience du chapitre 7.

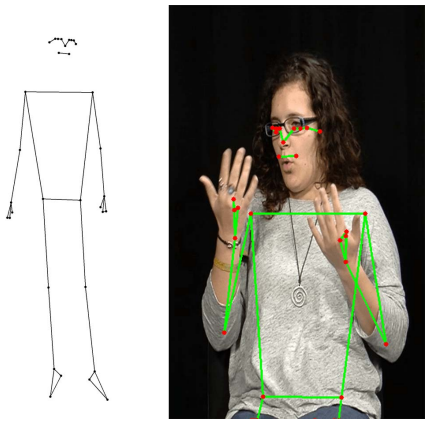
De la même manière que pour le dataset sur les mains, il est nécessaire de définir

un niveau de granularité quant à l'information fournie par les repères. Pour ce faire, plusieurs éléments sont ciblés :

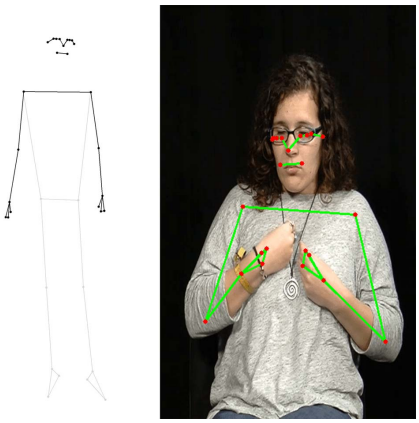
- les mouvements des bras ;
- les mouvements des épaules ;
- l'orientation de la tête ;
- l'orientation (sommaire) des mains.

Par défaut, *MediaPipe* [52] fournit 33 repères (voir Figure 4.8a) liés à la pose d'un individu. Cependant, certains de ces repères ne rentrent pas dans les critères précédemment définis. En effet, l'information concernant le bas du corps (jambes, pieds, etc.) est considérée comme n'étant pas pertinente. Cela est également appuyé par le fait qu'une part significative des vidéos en langue des signes du dataset LSFB_CONT [6] présente un signeur assis dont le bas du corps ne rentre pas dans le cadre (voir Figure 4.8b).

En conséquence, le dataset sur la pose ne sélectionne qu'un sous-ensemble de 23 repères (voir Figure 4.8b) et en élimine donc 10 concernant le bas du corps. Chaque repère est considéré en 2-dimensions (x et y).



(a) Repères du squelette entier. Cela comprend les points de repères des jambes et des pieds.



(b) Repères du squelette supérieur. Les repères concernant le bas du corps sont ignorés.

FIGURE 4.8 – Repères concernant la pose (squelette) du signeur.

Finalement, l'ensemble de l'information relative à la pose du signeur sauvegardée pour une vidéo donnée de n frames est une séquence de longueur n où chacun de ses éléments consiste en 23 repères (x, y). Soit un tableau de n lignes et 46 colonnes. Le dataset complet consiste donc en tableaux au format CSV correspondants aux vidéos du dataset LSFB_CONT [6].

Afin d'améliorer la qualité des données et de remplir les valeurs manquantes, une étape de post-traitement similaire à celle des mains (voir section 4.2.1) est appliquée.

4.2.3 Troisième dataset : le visage

Le troisième et dernier dataset concerne le visage des signeurs. En effet, lorsqu'un signeur est actif et effectue une série de signes, il est souvent très expressif (voir Figure 4.9). Il se peut qu'il bouge les lèvres, qu'il hausse les épaules ou encore qu'il bouge la tête.

FIGURE 4.9 – Signeurs expressifs lorsqu'ils effectuent des signes.

Ces caractéristiques liées aux mouvements du visage peuvent apporter un supplément d'information pour la segmentation des signes. Cependant, l'utilisation de ce dataset a été laissée pour de futures expériences ou recherches. Celui-ci est tout de même brièvement abordé, car il est prêt et est mis à disposition [51].

Ce dataset contient un nombre important de repères. En effet, par défaut, MediaPipe [52] extrait 468 points de repères sur le visage d'un signeur (voir Figure 4.10). Cela constitue une information très détaillée. Dans le cadre de la langue des signes, il est envisageable de filtrer ces repères et de ne se concentrer que sur les plus pertinents, par exemple les lèvres du signeur et les sourcils du signeur (60 repères). Chaque repère est considéré en deux dimensions.

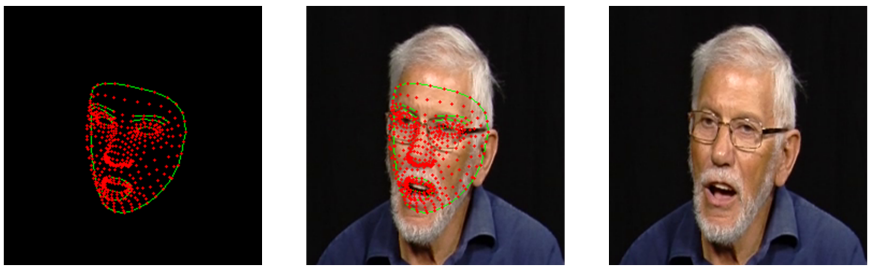


FIGURE 4.10 – Exemple de repères du visage extraits à partir d'une frame contenant un signeur.

L'ensemble de l'information relative au visage d'un signeur pour une vidéo donnée de n frames est une séquence de longueur n où chacun de ses éléments consiste en 468 repères (x, y). Soit un tableau de n lignes et 936 colonnes. Le dataset complet consiste donc en tableaux au format CSV correspondants aux vidéos du dataset LSFB_CONT.

Afin d'améliorer la qualité des données et de remplir les valeurs manquantes, une étape de post-traitement est appliquée. Celle-ci est similaire au datasets sur les mains et le squelette (voir sections 4.2.1 et 4.2.2) et est décrite dans la section 4.3.

4.3 Post-traitement des repères

Un soin particulier est apporté à la préparation des repères. Malgré tout, deux problèmes surviennent : certains repères sont manquants et des imprécisions engendrent un phénomène de vibration entre les frames. Les sections 4.3.1 et 4.3.2 atténuent ces problèmes de qualité des données.

4.3.1 Données manquantes

Les vidéos contenues dans le dataset LSFB_CONT contiennent un signeur dans une discussion en langue des signes. Dès lors, chaque frame des vidéos contient un individu unique à partir duquel les repères peuvent être extraits. Cependant, MediaPipe [52] ne détecte pas parfaitement cet individu. En effet, cet outil de détection considère que certaines frames ne contiennent pas d'individu. Cette erreur de détection entraîne des données manquantes pour certaines frames.

Une première étape de nettoyage des données consiste donc à définir une valeur pour ces repères dont la position n'a pas été définie par MediaPipe [52]. Au sein de ce mémoire, cela est réalisé grâce à l'interpolation linéaire [53]. Il s'agit de tracer une ligne depuis la dernière valeur connue jusqu'à la prochaine, comme illustré par un exemple sur la Figure 4.11.

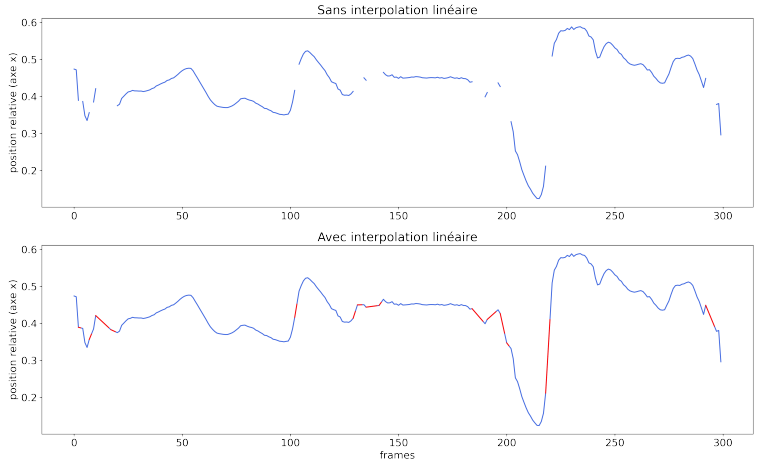


FIGURE 4.11 – Exemple d'interpolation linéaire appliquée sur l'évolution de la position d'un repère (Figure du haut). Après l'interpolation, les valeurs manquantes possèdent une valeur (Figure du bas).

4.3.2 Imprécisions et données aberrantes

Une seconde préoccupation dans la qualité des données au sein des repères est la présence d'imprécisions ou de valeurs aberrantes. Les imprécisions dans le calcul des repères provoquent de légères vibrations des repères tandis que les valeurs aberrantes, bien que rares, créent des mouvements soudains et imprévus.

La Figure 4.12 représente un exemple d'évolution de la position d'un repère au fil du temps (des frames). Les imprécisions et valeurs aberrantes y apparaissent comme des fluctuations rapides. Ces fluctuations peuvent être considérées comme du bruit dans un signal. Or, des techniques permettant de minimiser ce bruit existent. Celles-ci s'appellent des techniques de smoothing des données. Celle employée au sein du mémoire est le filtre Savitzky-Golay [54]. Il est utilisé avec une longueur de fenêtre de 7 et ordre polynomial de 2. La Figure 4.12 montre un exemple d'évolution de la position (axe x) d'un repère de la main avant et après l'application du filtre de Savitzky-Golay.

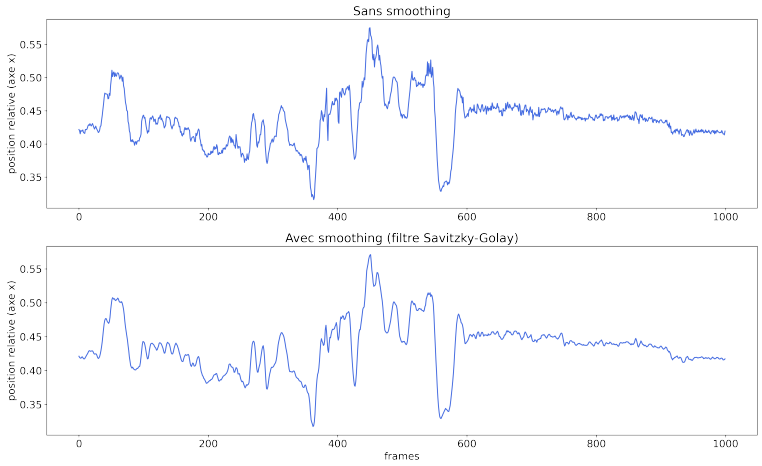


FIGURE 4.12 – Exemple de smoothing de la position sur l'axe x d'un repère (Figure du haut). Après l'application du smoothing, l'évolution de la position du repère est plus lisse et les variations soudaines sont atténuées (Figure du bas).

L'étape de smoothing améliore grandement la qualité des données. Les mouvements des signeurs sont plus fluides et les valeurs aberrantes sont mitigées.

4.4 Récapitulatif

Le mémoire présente et décrit en détail de nouveaux datasets réalisés dans la cadre de la recherche effectuée sur la segmentation automatique de la langue des signes. Ceux-ci sont réalisés à partir de repères extraits des vidéos du dataset LSFB_CONT [6]. Ils sont au nombre de trois :

- les repères des mains, section 4.2.1 ;
- les repères du squelette supérieur, section 4.2.2 ;
- les repères du visage, section 4.2.3.

Ces datasets utilisent tous comme cible les mêmes annotations mises sous forme de vecteurs représentant la segmentation (voir section 4.1). Finalement, la préparation ainsi que le nettoyage de ces données sont décrits section 4.3.

Décrire les datasets réalisés et mis à disposition est un des objectifs du mémoire. Ceux-ci sont également utilisés durant la suite du mémoire dans les expériences (voir chapitres 5, 6 et 7) afin d'entraîner les modèles, mais également afin d'évaluer la segmentation réalisée par ces modèles.

Chapitre 5

Améliorer la détection de la coarticulation

Ce chapitre décrit la constitution d'un premier système de segmentation automatique de la langue des signes (segmentation LS) à travers une première expérience. La segmentation LS s'intéresse à l'activité d'un signeur dans une discussion en langue des signes. Une approche simple est de se baser sur la présence ou l'absence d'un signe à un instant donné à partir des annotations des vidéos (voir section 4.1). Cela signifie entraîner un modèle capable de discriminer deux labels : *signing* lorsque le signeur effectue un signe, sinon *waiting*. Cependant, une seconde approche consiste en la séparation du label *waiting* en deux labels distincts : *coarticulation* pour les périodes de moins d'une seconde entre deux signes, et *waiting* pour les périodes d'inactivité plus longues. De cette façon, on encourage le modèle à apprendre de manière séparée le concept de coarticulation et le concept d'inactivité (voir section 2.2). La Figure 5.1 représente une ligne du temps avec les signes (*signing*) effectués par un signeur (les signes sont les zones grises) espacés par de longues périodes d'inactivités (*waiting*) ou par de courtes transitions entre les signes (*coarticulation*).

La principale difficulté de la segmentation des signes est la détection de la coarticulation. En effet, les zones de coarticulation interviennent lorsque les signes sont proches et sont difficilement différentiables des moments où le signeur effectue un signe. De plus, la coarticulation est très fréquente et dure peu de temps (500 millisecondes en moyenne) ce qui limite le nombre d'exemples de coarticulation disponibles pour le modèle.

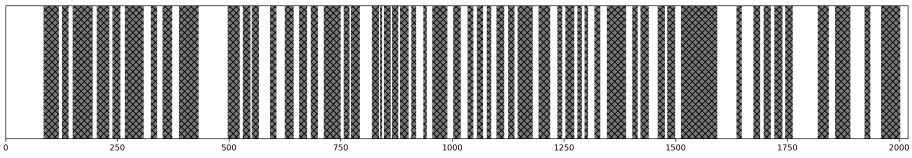


FIGURE 5.1 – Exemple de segmentation cible.

Cette constatation est la motivation principale d'une expérience menée dans

cette section. Celle-ci consiste à évaluer l'impact de cette séparation du label *waiting* en deux labels distinct : *coarticulation* et *waiting*. On obtient comme actions considérées du signeur : le signeur est en train de faire un signe, le signeur est dans une situation de coarticulation entre deux signes, et finalement le signeur est inactif. Cette expérience met en avant l'importance de la coarticulation dans les modèles utilisés pour la segmentation LS.

En parallèle, l'expérience évalue des pistes de solution au problème du non-balancement des données. C'est-à-dire que la proportion de frames, pour chaque label, est fortement différente (voir Table 5.1). Les résultats montrent le problème rencontré lorsqu'aucune stratégie n'est mise en place pour faire face à ce non-balancement : dans la prédiction, les signes proches fusionnent en des signes plus longs.

5.1 Cadre expérimental

Deux scénarios sont réalisés dans cette expérience avec, respectivement, deux et trois classes. Le premier scénario utilise comme configuration deux labels : *signing* lorsque le signeur est en train d'effectuer un signe, et *waiting* lorsque celui-ci n'est pas en train d'effectuer un signe. Le second scénario considère, lui, trois labels différents. Le premier est toujours *signing*, le second est *waiting*, tandis que le troisième est *coarticulation*. Dans cette expérience, une période de coarticulation est définie comme étant un segment de moins de 1s entre deux signes, où le signeur n'est pas en train d'effectuer un signe. De cette manière, les transitions entre signes sont différenciées des longues périodes d'inactivité du signeur. L'isolation de la coarticulation est détaillée section 4.1.

Le modèle utilisé est un LSTM (voir section 3.3.2) [12] possédant 128 états cachés suivis d'une couche linéaire combinant ces états en 2 ou 3 valeurs cibles suivant le scénario. Pour chacune des expériences, deux loss functions sont utilisées (*Cross Entropy Loss* [26] sans et avec poids). En effet, le modèle est entraîné une première fois en utilisant une loss function qui ne prend pas en compte le non-balancement des données, tandis qu'un second entraînement utilise, lui, une loss function avec une stratégie contre ce non-balancement. L'ajout de poids relatifs aux classes permet de contre-balancer les différences de proportions (non-balancement) des labels au sein des données. Le poids utilisé pour une classe est l'inverse de la proportion des instances de cette classe parmi l'ensemble du dataset (voir Table 5.1).

	classes	frames	fréquence	poids
sans label coarticulation	signing	1.836.565	31,72%	3,15
	waiting	3.953.383	68,28%	1,46
avec label coarticulation	signing	1.836.565	31,72%	3,15
	waiting	3.176.663	54,87%	1,82
	coarticulation	776.720	13,41%	7,45

TABLE 5.1 – Résumé des configurations des deux expériences avec et sans le label *coarticulation*.

Le dataset utilisé consiste en un ensemble de séquences de squelettes de la partie

supérieure du signeur (voir section 4.2.2). Il est construit à partir du dataset LSFB-CONT [6]. Le dataset LSFB-CONT est décrit plus précisément dans la section 2.3 tandis que la préparation des squelettes pour chaque frame est décrite dans la section 4.3. La Figure 5.2 présente un aperçu des données utilisées pour des frames sélectionnées aléatoirement. Le dataset d'entraînement et le dataset de test contiennent, respectivement, $3,5$ millions et $2,2$ millions de frames. Afin d'éviter les biais, les signeurs des deux datasets sont différents.

La longueur des vidéos varie de quelques minutes à plus d'une demi-heure. Comme les vidéos sont des séquences de longueurs variables, des sous-séquences (fenêtres) de taille fixe (1500 frames) sont utilisées. L'écart entre le début d'une fenêtre et la suivante est de 800 frames. Le modèle navigue 80 fois sur chaque instance du dataset. Ce nombre est appelé le nombre d'*epochs*.

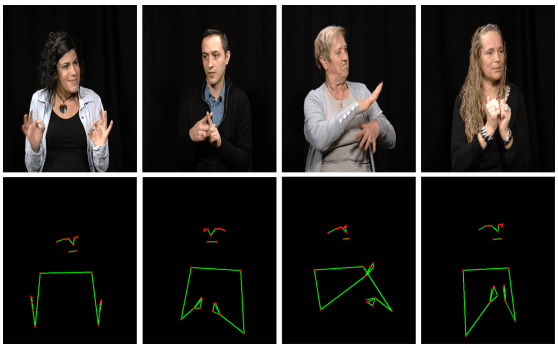


FIGURE 5.2 – Squelettes utilisés comme données d'entrée

Afin d'évaluer les performances du modèle, différentes métriques sont utilisées. Celles-ci permettent de quantifier sa performance et sont donc cruciales pour une évaluation rigoureuse. Les métriques (voir section 3.4) utilisées dans cette expérience sont l'accuracy, la balanced accuracy, et le recall pour chaque classe.

Dans cette expérience, l'accuracy seule est biaisée par la surreprésentation des classes *signing* et *waiting* par rapport à la classe *coarticulation* (voir section 3.4.3). Le non-balancement des données dans le dataset d'entraînement (voir Table 5.1) fait que la balanced accuracy est plus pertinente que l'accuracy seule. Le recall, lui, permet de savoir quelles classes sont plus complexes à modéliser par le modèle. La valeur maximum pour une métrique donnée est représentée en gras.

Une nouvelle métrique introduite est le taux de coarticulations détectées. Une zone de coarticulation, c'est-à-dire un temps d'inactivité de moins de 1s entre deux signes, est détectée lorsque les deux signes qui l'entourent sont bel et bien séparés par au moins une frame dont le label n'est pas *signing*. La Figure 5.3 montre un exemple de prédiction où la zone de coarticulation est effectivement détectée. Si ce taux est faible, cela signifie que les signes prédits ont tendance à fusionner entre eux.

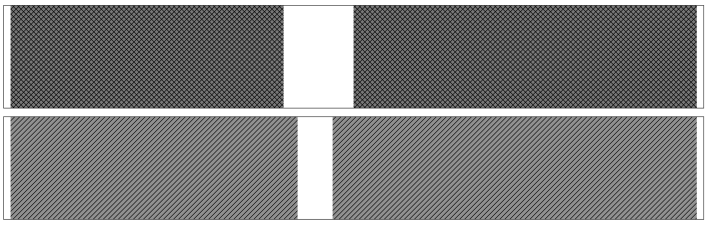


FIGURE 5.3 – La Figure du haut montre deux signes séparés par une zone de coarticulation, tandis que celle du dessous montre la prédiction associée. Dans les deux Figures, les deux signes sont séparés par au moins une frame qui n'est pas labellisée *signing*. Dès lors, on considère que la zone de coarticulation a bien été détectée même si elle ne correspond pas exactement aux annotations du corpus.

5.2 Résultats

Les Table 5.2 et Table 5.3 montrent les résultats des deux configurations avec, respectivement, 2 et 3 classes. Chacune des configurations montre les résultats avec et sans poids ajoutés à la loss function.

	acc.	bal. acc.	recall		detect. coart.
			signing	waiting	
avec poids	81,22%	**85,40**%	**96,62**%	74,19%	2,92%
sans poids	**81,60**%	85,32%	95,27%	**75,36**%	**3,67**%

TABLE 5.2 – Résultats de la segmentation LS réalisée avec seulement deux classes : *signing* et *waiting*.

	acc.	bal. acc.	recall			detect. coart.
			signing	waiting	coart.	
avec poids	73,82%	**67,93**%	46,20%	90,90%	**66,70**%	**75,23**%
sans poids	**81,56**%	62,45%	**93,58**%	**93,58**%	00,70%	5,35%

TABLE 5.3 – Résultats de la segmentation LS réalisée avec la coarticulation, soit les classes : *signing*, *waiting* et *coarticulation*.

5.3 Discussion et interprétation des résultats

Le premier scénario soulève un problème important : le très faible taux de coarticulations détectées. En effet, que ce soit avec des poids ou sans au sein de la loss function, ce taux reste très faible. Cela est fortement visible dans les résultats de la segmentation (voir Figure 5.4b) où la grande majorité ($\geq 96\%$) des segments blancs de moins de 1s entre les signes ne sont pas détectés par le modèle. Celui-ci fusionne les signes trop proches et forme des signes plus longs. **Un des plus grands défis de la segmentation des signes dans les vidéos en langue des signes est donc bel et bien la détection des périodes de coarticulation.** Dans cette configuration, l'ajout de poids à la fonction de

perte améliore sensiblement la balanced accuracy, mais ne règle pas le problème du faible taux de détection de la coarticulation.

Dans le second scénario, les résultats sont flagrants. Premièrement, l'ajout des poids à la fonction de perte améliore significativement les performances du modèle. La balanced accuracy augmente de plus de 5%. La classe la plus impactée par l'ajout des poids est la plus minoritaire, c'est-à-dire la classe coarticulation. Sans les poids, celle-ci est presque totalement ignorée tandis qu'avec les poids, son recall est de $66,70\%$.

L'isolation de la coarticulation nous apporte également d'autres informations. Le recall des classes *signing* et *waiting* diminue quand la coarticulation est considérée. Cela s'explique par le fait que, sans poids, la prédiction surreprésente ces classes en détriment de classe *coarticulation*.

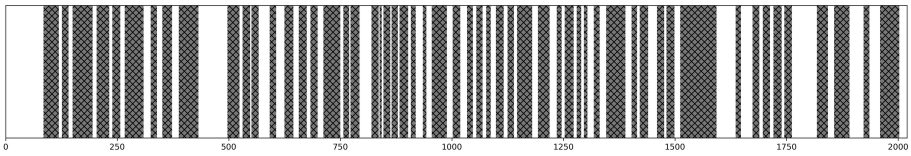
Un résultat significatif est l'augmentation drastique du taux de détection des périodes de coarticulation. Sans l'isolation de la coarticulation, ce taux est de $3,67\%$, alors qu'avec cette isolation et l'ajout de poids à la fonction de perte, celui-ci passe à $75,23\%$. La Figure 5.4c montre ce résultat où les signes ne sont plus fusionnés comme auparavant. Cependant, cette segmentation des signes est encore trop imprécise.

En conclusion, cette expérience montre l'importance de l'ajout d'une classe propre à la coarticulation. Elle montre également qu'il est crucial d'utiliser des mesures contre le non-balancement des données. Un exemple de telle mesure est l'ajout de poids à la loss function. L'expérience montre également que les résultats obtenus avec les squelettes sont insuffisants. Le chapitre 7 montre que de meilleurs repères peuvent être sélectionnés.

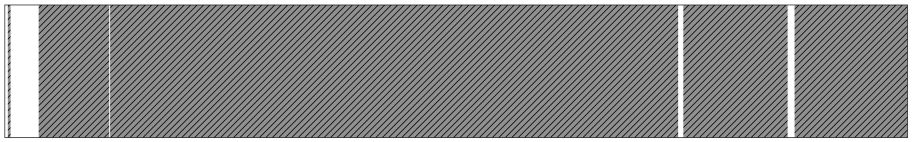
5.4 Récapitulatif

Un premier système de segmentation automatique de la langue des signes est réalisé à travers une expérience. Celle-ci montre le rôle crucial de la considération de la coarticulation dans la segmentation de la langue des signes. Cependant, cette isolation de la coarticulation entraîne un fort non-balancement des données. Celui-ci doit également être considéré.

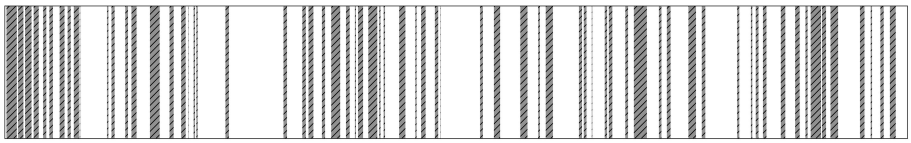
Les expériences réalisées dans la suite du mémoire (voir chapitres 6 et 7) considèrent ces éléments dans leur cadre expérimental : la coarticulation est isolée et des poids sont ajoutés à la loss function.



(a) Segmentation cible.



(b) Prédiction sans prise en compte de la coarticulation.



(c) Prédiction avec prise en compte de la coarticulation.

FIGURE 5.4 – Exemple de segmentation où la Figure 5.4a représente la cible, tandis que les Figures 5.4b et 5.4c montrent les prédictions réalisées, respectivement, avec et sans prise en compte de la coarticulation. Ces Figures représentent la ligne du temps d'un même segment tiré d'une vidéo du dataset LSFB_CONT (voir section 2.3). La classe *signing* est représentée en gris tandis que les classes *waiting* et *coarticulation* sont en blanc.

Chapitre 6

Une meilleure modélisation des transitions dans les réseaux récurrents

Comme expliqué dans l'état de l'art (voir section 3.3), les architectures récurrentes (RNN), en particulier les LSTMs (voir section 3.3.2), s'avèrent utiles pour traiter des données séquentielles. Cependant, ces modèles souffrent de limitations qui peuvent les empêcher de modéliser correctement les transitions d'état [55]. Plusieurs architectures ont été développées afin d'atténuer ces problèmes. Dans cette expérience, l'objectif est d'évaluer leur efficacité en se concentrant sur la segmentation des signes (segmentation LS).

En premier lieu, la section 6.1 décrit les limitations des LSTMs sur lesquelles l'expérience se concentre. Ensuite, deux extensions du LSTM sont décrites dans la section 6.2 et dans la section 6.3 : l'Explicit Duration Recurrent Network (EDRN) [55] et le Mogrifier LSTM (mLSTM) [56]. Après cela, le cadre expérimental de l'expérience est décrit dans la section 6.4, suivi des résultats en section 6.5 et de l'interprétation de ceux-ci en section 6.6.

6.1 Des limitations du LSTM

Les LSTMs sont parmi les RNNs les plus populaires. Cependant, ceux-ci souffrent de limitations. Celles-ci motivent la littérature à créer des extensions ou des versions améliorées de ceux-ci. Parmi ces extensions, deux s'intéressent à la modélisation des transitions : l'Explicit Duration Recurrent Network (EDRN) [55] et le Mogrifier LSTM (mLSTM) [56]. Cette section met en avant des limitations des LSTMs et en quoi l'EDRN et le mLSTM améliorent ceux-ci.

Des durées implicitement modélisées par des distributions géométriques

Dans son papier sur l'Explicit Duration Recurrent Network (EDRN) [55], Shun-Zheng Yu démontre qu'un réseau récurrent (RNN, voir section 3.3.2) peut être interprété à partir du fonctionnement d'un Hidden Markov Model (HMM, voir section 3.3.1). Un corollaire est que de la même façon qu'un HMM, les RNNs possèdent des états cachés et que leurs durées sont implicitement modélisées par des distributions géométriques. Afin de mettre en avant cette problématique, il est nécessaire de décrire les RNNs et les LSTMs avec les concepts des HMMs.

Dans un RNN, un hidden vector est mis à jour à chaque instant t d'une séquence. Dans la définition classique d'un simple RNN (simple RNN, voir section 3.3.2), elle peut être formalisée comme suit :

$$h_t = \tanh(\mathbf{W}x_t + \mathbf{U}h_{t-1} + b)$$

où $\mathbf{W}$ et $\mathbf{U}$ sont, respectivement, les poids entre les couches du réseau et les poids récurrents, tandis que b est le biais. Shun-Zheng Yu [55] formalise cette mise à jour à partir des concepts des HMMs à travers la formule des filtres de Kalman [57] :

$$\alpha_t = \tanh(\alpha_{t-1}\mathbf{A} + x_t\mathbf{B} + b)$$

où, par rapport à un HMM, $\mathbf{A}$ joue un rôle similaire à celui de la matrice des probabilités de transition d'état, tandis que $\mathbf{B}$ un rôle similaire que la matrice des probabilités d'émission et b la distribution marginale. Les durées d'état au sein d'un simple RNN sont donc modélisées de la même façon que dans un HMM.

En ce qui concerne les LSTMs, ceux-ci peuvent également être formalisés à partir des concepts des HMMs. Shun-Zheng Yu [55] démontre que les paramètres d'un LSTM ont également une signification qui permet de mettre en avant le problème de durées implicitement modélisées par des distributions géométriques :

$$\begin{aligned}
\mathbf{G}_{\text{fg}} &= \sigma(m_{t-1}\mathbf{A}_{\text{fg}} + x_t\mathbf{B}_{\text{fg}} + b_{\text{fg}}) \\
\mathbf{G}_{\text{in}} &= \sigma(m_{t-1}\mathbf{A}_{\text{in}} + x_t\mathbf{B}_{\text{in}} + b_{\text{in}}) \\
\mathbf{G}_{\text{th}} &= \tanh(m_{t-1}\mathbf{A}_{\text{th}} + x_t\mathbf{B}_{\text{th}} + b_{\text{th}}) \\
C_t &= C_{t-1} * \mathbf{G}_{\text{fg}} + \mathbf{G}_{\text{th}} * \mathbf{G}_{\text{in}} \\
m_t &= \tanh(C_t\mathbf{A}_{\text{ot}} + x_t\mathbf{B}_{\text{ot}} + b_{\text{ot}})
\end{aligned}$$

où $\mathbf{B}_{\text{in}}$, $\mathbf{B}_{\text{th}}$, $\mathbf{B}_{\text{fg}}$, et $\mathbf{B}_{\text{ot}}$ ont la même signification que la matrice de probabilité d'émission $\mathbf{B}$, tandis que $\mathbf{A}_{\text{in}}$, $\mathbf{A}_{\text{th}}$, $\mathbf{A}_{\text{fg}}$, et $\mathbf{A}_{\text{ot}}$ ont la même signification que la matrice des probabilités de transition d'état $\mathbf{A}$, et finalement, b_{in}, b_{th} et b_{fg} ont la même signification que la distribution marginale b. Les états cachés (hidden states), au sein d'un LSTM, sont représentés dans le cell state vector C_t. L'état courant est sa composante avec la valeur la plus importante.

En conclusion, il est possible de formaliser les réseaux récurrents à partir des concepts des Hidden Markov Models (HMM). Cette formalisation met en avant la présence d'**états cachés** au sein du LSTM. Finalement, de la même manière qu'un état caché dans un HMM, la durée de ceux-ci est **implicitement modélisée** par des distributions géométriques (voir section 3.3.1). L'EDRN améliore

le LSTM en s'inspirant des HSMM de la section 3.3.1 pour mieux modéliser la durée des hidden states (voir section 6.2).

Manque de dépendance au contexte

Dans un HMM, la fonction de transition d'un état vers un autre dépend uniquement de l'état précédent et ne dépend pas des observations. Les LSTMs assouplissent ces contraintes à travers l'utilisation de composants supplémentaires appelés feedback connections et gates.

Une gate joue un rôle similaire à une porte logique. Celle-ci conditionne un vecteur en activant ou désactivant certaines de ses composantes. De façon générale, cela est réalisé en multipliant chaque composante par un nombre entre 0 et 1. De cette façon, on peut partiellement modifier des paramètres et ne pas complètement effacer les anciens. La transition d'un état vers un autre oublie donc moins rapidement les états précédents éloignés. Cela est appelé une **contextualisation de la fonction de transition**, car celle-ci prend en compte plus d'éléments en rapport avec le contexte, c'est-à-dire la séquence d'entrée.

Ensuite, dans un HMM, le seul lien existant entre deux instants est la probabilité de transition entre les états. Cependant, dans un LSTM, des connexions supplémentaires multiplient les liens entre les itérations. Celles-ci sont appelées feedback connections. Par exemple, dans un LSTM, la prédiction notée h_t est envoyée à la cellule suivante qui l'utilise comme information supplémentaire dans le calcul de sa propre prédiction. La Figure 3.14a de la section 3.3.2 représente de nombreuses feedback connections représentées par les flèches.

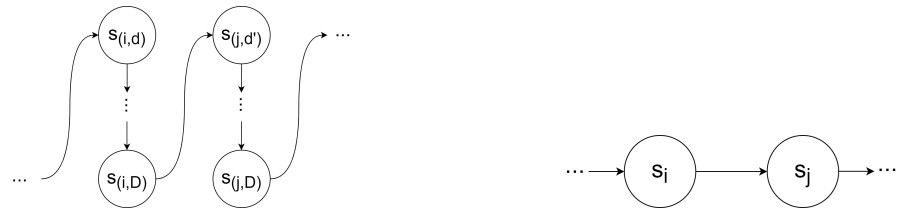
Malgré l'ajout des gates et des feedback connections, la fonction de transition d'un état vers un autre ne dépend pas assez des observations passées et courantes [56]. En effet, les gates et les feedback connections n'augmentent pas assez la dépendance aux observations. Cette limitation des LSTMs est appelée un **manque de dépendance au contexte**. Le mLSTM améliore la contextualisation de la fonction de transition grâce à une étape préliminaire qui fournit une entrée déjà contextualisée au LSTM (voir section 6.3).

6.2 Explicit Duration Recurrent Network

Le Explicit Duration Recurrent Network (EDRN) est une variante du LSTM inspirée des Hidden semi-Markov Models (HSMM) où la durée des états est explicitement modélisée. Le but est d'utiliser des distributions plus représentatives que les distributions géométriques (voir section 3.3.1) afin de mieux modéliser la durée des états et donc d'améliorer les performances.

Dans un EDRN, chaque état (hidden state) possède D sous-états. La transition entre états est donc différente que dans les LSTMs classiques. En effet, dans l'EDRN, il est nécessaire de transiter jusqu'au dernier sous-état avant de transiter vers l'état suivant. La Figure 6.1a représente une transition d'état au sein d'un EDRN. On suppose que l'on se trouve à l'état $S_{i,d}$, c'est-à-dire l'état i au sous-état d. On transite ensuite jusque l'état $S_{i,D}$ qui est le dernier sous-état du même état i. Une fois arrivé à ce dernier sous-état, on transite vers l'état suivant $S_{j,d'}$ où $j \neq i$ et $d' \in \{1, \ldots, D\}$. En comparaison, la Figure 6.1b montre un cas

similaire au sein d'un LSTM classique. Dans celui-ci, on passe simplement de l'état S_i à l'état S_j.



(a) Transition au sein d'un EDRN. (b) Transition au sein d'un LSTM.

FIGURE 6.1 – Comparaison d'une transition d'un état i vers un état j. La Figure de droite représente cette transition dans un LSTM classique, tandis que la Figure de gauche représente les transitions entre états dans l'EDRN. Dans celui-ci, les états sont précédés par des sous-états suivant la valeur d'une variable aléatoire de durée. L'état i effectue des transitions du sous-état d au sous-état final D, avant d'effectuer une transition vers l'état j au sous-état d'.

La durée des hidden states au sein des LSTM est implicitement géométrique (voir section 6.1). Dans l'EDRN, la transition entre deux états passe par une succession de transitions entre sous-états. Dès lors, cette transition est modélisée par une somme de distributions géométrique. Celle-ci forme une distribution plus complexe pouvant modéliser des durées d'états plus variées et pas nécessairement géométriques [58]. Un LSTM classique est un cas particulier de l'EDRN où au sein de chaque état, la probabilité de transition entre sous-état est la même et donc où la somme forme une distribution géométrique.

L'ajout de sous-états au sein de la cellule récurrent de l'EDRN nécessite d'adapter les équations de la cellule LSTM (voir section 3.3.2). Shun-Zheng Yu [55] formalise un EDRN avec M hidden states et D sous-états par hidden state à l'aide des équations suivantes :

$$
\begin{aligned}
\mathbf{G}_{\text{fg}} &= \sigma(m_{t-1}\mathbf{A}_{\text{fg}} + x_t\mathbf{B}_{\text{fg}} + b_{\text{fg}}) \\
\mathbf{G}_{\text{in}} &= \sigma(m_{t-1}\mathbf{A}_{\text{in}} + x_t\mathbf{B}_{\text{in}} + b_{\text{in}}) \\
\mathbf{G}_{\text{th}} &= \sigma(\alpha_{t-1}\mathbf{A}_{\text{pt}} + \alpha_{t-1}(:, D)\mathbf{A}_{\text{th}} + x_t\mathbf{B}_{\text{fg}} + b_{\text{fg}}) \\
\alpha_t &= \alpha_{t-1} \times \mathbf{G}_{\text{fg}} + \mathbf{G}_{\text{th}} \times \mathbf{G}_{\text{in}} \\
\mathbf{G}_{\text{ot}} &= \sigma(\alpha_t(:, D)\mathbf{A}_{\text{ot}} + x_t\mathbf{B}_{\text{ot}} + b_{\text{ot}}) \\
\alpha_t^{**} &= \tanh(\alpha_t \times \mathbf{G}_{\text{ot}}) \\
m_t(:) &= \sum_{d=1}^{D-1} \alpha_t^{**}(:, d) + \alpha_t^{**}(:, D)\mathbf{A}_{\text{st}}.
\end{aligned}
$$

où les paramètres $\mathbf{A}$, $\mathbf{B}$ et b sont, respectivement, les probabilités de transition, d'émission et les distributions marginales (voir section 6.1).

Dans un LSTM classique possédant M hidden states, l'état courant est la composante du cell state vector possédant la plus grande valeur. En ce qui concerne l'EDRN, le cell state vector α_t contient l'information sur l'état courant mais également sur le sous-état courant. α_t est donc un vecteur de longueur $M \times D$ où la

plus grande valeur représente le sous-état courant. Des détails supplémentaires quant aux équations de l'EDRN sont fournis dans l'article original [55].

6.3 Mogrifier LSTM

Le Mogrifier LSTM (mLSTM) [56] est une extension du LSTM se concentrant sur la problématique du manque de dépendance au contexte dans la fonction de transition des hidden states. La stratégie adoptée par le mLSTM est d'atténuer ce problème en fournissant une représentation plus contextualisée de l'entrée x et du hidden vector h à la cellule LSTM :

$$\text{mLSTM}(x, h) = \text{LSTM}(\tilde{x}, \tilde{h})$$

Décrire le fonctionnement du mLSTM revient donc à décrire le fonctionnement de cette contextualisation des entrées du LSTM. L'entrée x et le hidden vector h passent dans une succession d'opérations, appelées rounds, influençant mutuellement les valeurs de leurs composantes. La Figure 6.2 représente 5 de ces rounds avant d'envoyer x et h à une cellule LSTM.

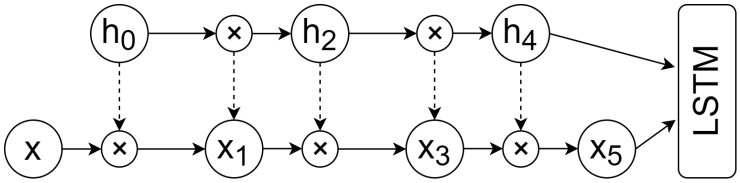


FIGURE 6.2 – Contextualisation des entrées du LSTM à travers 5 rounds du mLSTM. Chaque round mélange partiellement les composantes des vecteurs x et h avant de les passer à la cellule LSTM. Ce mélange est opéré grâce à des gates représentées par des $\times$.

L'opération pour chaque round i est formalisée par les équations suivantes :

$$x_i = 2\sigma(\mathbf{Q}_i h_{i-1}) \times x_{i-2} \text{ pour } i \in [1 \dots r] \text{ tel que } i \text{ est impair}$$
$$h_i = 2\sigma(\mathbf{R}_i x_{i-1}) \times h_{i-2} \text{ pour } i \in [1 \dots r] \text{ tel que } i \text{ est pair}$$

avec $x_{-1} = x$ et $h_0 = h$.

Tous les deux rounds, x et h possèdent de nouvelles valeurs obtenues après l'application mutuelle de gates. Une gate est une opération logique activant ou désactivant certaines composantes du vecteur. Cela est réalisé en multipliant chacune de ces composantes par un nombre entre 0 et 1. Un tel nombre est obtenu grâce à la fonction σ.

Par exemple, l'équation $x_i = 2\sigma(\mathbf{Q}_i h_{i-1}) \times x_{i-2}$ modifie x en appliquant une gate en fonction de h sur le vecteur x précédent. Les paramètres $\mathbf{Q}$ et $\mathbf{R}$ contrôlent le comportement des gates et doivent être entraînés par le modèle. La Figure 6.2 représente les gates par les symboles $\times$, les flèches continues représentent la multiplication tandis que les flèches en pointillés représentent l'utilisation des paramètres $\mathbf{Q}$ et $\mathbf{R}$.

En conclusion, le mLSTM mélange l'observation courante, c'est-à-dire l'entrée x, et le hidden vector h à travers une succession de rounds appliquant mutuellement des gates entre x et h. De cette façon, x et h dépendent l'un de l'autre et fournisse une entrée plus contextualisée, c'est-à-dire qui dépend davantage de l'observation, à la cellule LSTM.

6.4 Cadre expérimental

Le cadre expérimental de cette expérience est semblable à celui de l'expérience sur la détection de la coarticulation (chapitre 5). En effet, les données utilisées sont les squelettes supérieurs (voir section 4.2.2) avec des fenêtres de 1500 éléments avec un décalage de 800 éléments. Les modèles sont entraînés sur 30 epochs.

La loss function utilisée est la *cross entropy loss* et l'optimizer utilisé est la descente de gradient stochastique avec un learning rate de 0,01 et un facteur d'inertie de 0,9. Des poids sont ajoutés à la loss function, voir Table 5.1.

Trois modèles sont évalués, un LSTM classique (voir section 3.3.2) [12], un EDRN avec 4 sous-états par hidden state, et finalement un mLSTM avec un nombre de round défini à 5 d'après les recommandations des auteurs du modèle [56]. Chacun des modèles récurrents de cette expérience possèdent 128 hidden states. Un résumé des modèles est fourni dans la Table 6.1.

	input features	hidden features	sub-states	rounds
LSTM	46	128		
EDRN	46	128	4	
mLSTM	46	128		5

Table 6.1 – Configuration des modèles utilisés dans l'expérience.

6.5 Résultats

La Table 6.2 montre les résultats de l'expérience menées sur trois modèles différents : un LSTM, un EDRN et un mLSTM. Les métriques utilisées sont l'accuracy (acc.), la balanced accuracy (bal. acc.) et le recall pour chaque classe différente. Afin de mieux appréhender le comportement des modèles, la Figure 6.3 montre les distributions de durée pour les classes *signing* (première ligne) et *coarticulation* (seconde ligne) séparées en 3 colonnes pour, respectivement, le LSTM, l'EDRN et le mLSTM. Pour des raisons de lisibilité, les durées supérieures à 2s ne sont pas représentées, afin de se concentrer sur la partie la plus significative de la distribution.

	acc.	bal. acc.	recall		
			talking	waiting	coarticulation
LSTM	73.82	66.19	70.13	97.21	31.24
EDRN	74.41	67.11	**72.02**	**97.32**	31.99
mLSTM	**77.65**	**67.78**	71.54	95.91	**35.89**

Table 6.2 – Résultats de l'expérience avec le dataset LSFB_CONT

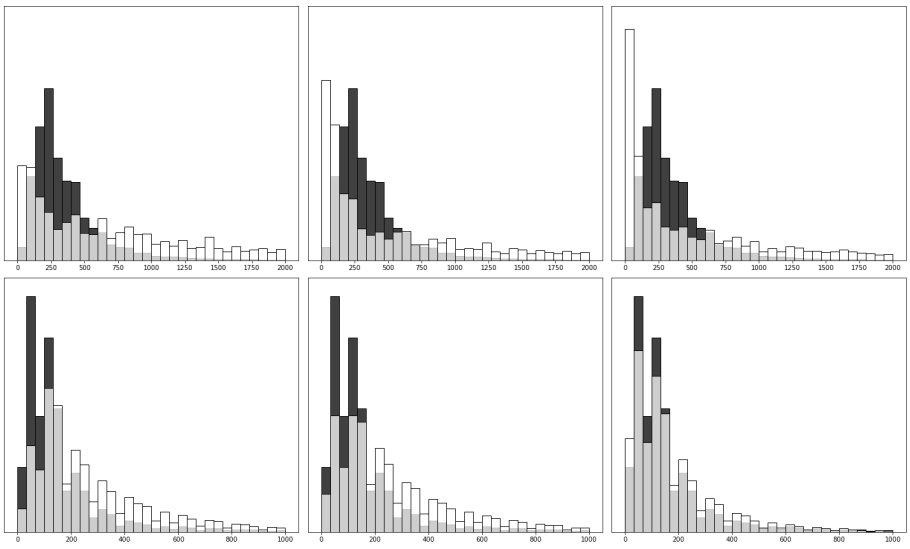


FIGURE 6.3 – Comparaison de la distribution des durées dans le dataset (noir) et pour la segmentation (blanc), pour les classes *signing* (première ligne) et *coarticulation* (seconde ligne). De gauche à droite : LSTM, EDRN, mLSTM.

6.6 Discussion et interprétation des résultats

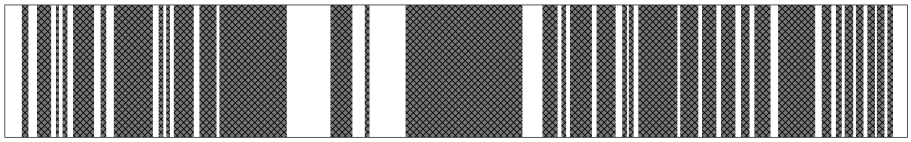
L'EDRN et le mLSTM ont de meilleures performances que le LSTM classique. En effet, leur balanced accuracy est plus élevée ainsi que leurs recalls. Cela s'observe également sur la Figure 6.3. La durée des signes est globalement trop longue en ce qui concerne le LSTM tandis que l'EDRN et le mLSTM prédisent des signes plus courts plus en accord avec les données réelles.

Un autre élément est le fait que le mLSTM modélise bien mieux les durées de coarticulations que l'EDRN. En effet, la Figure 6.3 (le graphique en bas à droite) montre que la distribution des durées des coarticulations prédites se superposent très bien avec les durées des coarticulations dans le dataset. Cela est cohérent avec le fait que le mLSTM ait un meilleur recall sur la classe coarticulation que l'EDRN. Cependant, le mLSTM prédit plus de signes courts que l'EDRN. En effet, la Figure 6.4c montre un exemple d'annotation d'un segment de vidéo avec le mLSTM. On peut observer que les signes prédits sont trop courts.

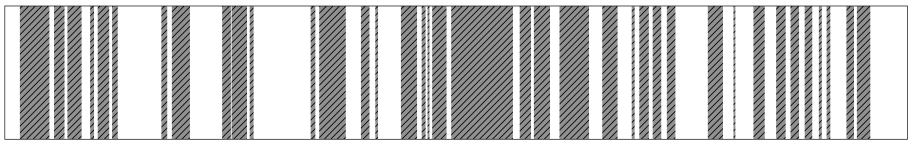
En ce qui concerne l'EDRN, celui-ci prédit également des signes trop courts (voir Figure 6.3). Cependant, le problème est moins prononcé qu'avec le mLSTM. La Figure 6.4b présente un exemple d'annotation d'un segment de vidéo avec l'EDRN. Malgré une meilleure distribution des durées que le mLSTM, l'EDRN a une balanced accuracy moindre que celui-ci et modélise moins bien la durée des coarticulations (voir Figure 6.3).

Cette expérience montre qu'une meilleure modélisation des transitions dans les RNNs améliore les performances du modèle. Cependant, ces expériences montrent que les solutions proposées dans ce mémoire ne sont pas suffisantes. Aucune approche ne se démarque, l'EDRN et le mLSTM ont chacun leurs qua-

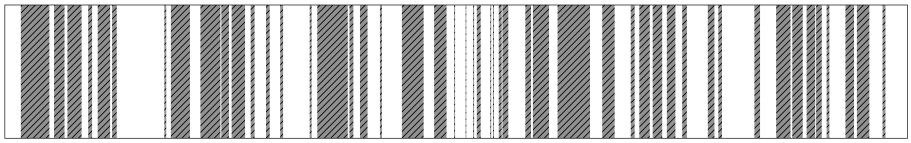
lités et leurs défauts. Cela démontre donc un intérêt à améliorer davantage la modélisation des transitions dans les RNNs.



(a) Segmentation dans le dataset.



(b) Prédiction de l'EDRN.



(c) Prédiction du Mogrifier LSTM.

FIGURE 6.4 – Exemple de prédictions. Les figures représentent la ligne du temps du même segment tiré d'une vidéo du dataset LSFB_CONT. La classe *signing* est représentée en gris tandis que les classes *waiting* et *coarticulation* sont en blanc. Le mLSTM prédit des signes plus courts, tandis que l'EDRN est moins précis.

6.7 Récapitulatif

Cette expérience s'intéresse aux limitations des RNNs, plus spécifiquement du LSTM, pour la modélisation des transitions. Afin d'évaluer l'impact de l'amélioration de cette modélisation dans le cas de la segmentation LS, deux extensions du LSTM sont évaluées en comparaison avec celui-ci : l'Explicit Duration Recurrent Network (EDRN) et le Mogrifier LSTM (mLSTM). Celles-ci arborent des stratégies différentes pour améliorer la modélisation des transitions.

L'expérience montre que l'amélioration de la modélisation des transitions augmente effectivement les performances dans le cadre de la segmentation LS. Cependant, cette modélisation peut encore être davantage améliorée. Le mémoire met donc en avant l'intérêt d'une meilleure modélisation des transitions au sein des RNNs.

Chapitre 7

Améliorations de la segmentation automatique

Ce mémoire présente un système de segmentation automatique de la langue des signes (segmentation LS) implémenté à partir d'un réseau récurrent (RNN, voir section 3.3.2) appelé le LSTM (voir section 3.3.2) [12]. Le chapitre 5 présente une première expérience menée afin de mettre en avant l'importance de l'isolation de la coarticulation à travers la séparation de la classe *waiting* en deux sous-classes distinctes : *waiting* et *coarticulation*. Ensuite, le chapitre 6 s'intéresse à la modélisation des transitions entre les hidden states au sein des RNNs. Deux extensions du LSTM sont mises en avant : l'Explicit Duration Recurrent Network (EDRN, voir section 6.2) et le Mogrifier LSTM (mLSTM, voir section 6.3).

Ces expériences ont permis d'améliorer significativement les performances de la segmentation LS. Ce chapitre investigue d'autres améliorations possibles afin d'améliorer davantage les performances d'un modèle de segmentation LS. Les aspects considérés sont :

- le choix du dataset ;
- le choix de la loss function ;
- le nombre de couches récurrentes dans le modèle ;
- l'utilisation de couches bidirectionnelles ou non.

Chaque section définit un de ces critères, les modèles finaux ainsi que leurs résultats respectifs sont présentés dans la section 7.5. Quatre modèles finaux sont présentés :

- un modèle détectant les périodes d'activité du signeur sans distinguer les signes proches ;
- un modèle segmentant les signes réalisés à la main droite ;
- un modèle segmentant les signes réalisés à la main gauche ;
- un modèle segmentant les signes réalisés quelle que soit la main utilisée par le signeur.

7.1 Cadre expérimental

Les benchmarks réalisés dans cette section se concentrent sur la segmentation LS des signes réalisés avec la main droite du signeur. Une approche similaire peut être réalisée pour la main gauche.

Afin d'évaluer les performances de cette segmentation, la configuration à 3 classes est utilisée. Celle-ci est décrite en détail dans l'expérience sur l'amélioration de la détection de la coarticulation chapitre 5. Les données utilisées sont des repères tels que décrits chapitre 4, soit de séquences de 1500 frames.

Le dataset utilisé dans les expériences est celui sélectionné par le benchmark ci-dessous dans la section 7.2. Pour des raisons de temps de calcul, le modèle évalué est un LSTM et non un EDRN ou un mLSTM. Cependant, les optimisations mises en avant dans cette expérience sont applicables sur ces modèles également.

Chaque résultat de benchmark est évalué suivant l'accuracy, la balanced accuracy et les recalls de chaque classe. L'entraînement des modèles est réalisé durant 50 epochs.

7.2 Choix du dataset

Une première optimisation est de sélectionner le dataset optimal pour la tâche de segmentation LS. En effet, dans ce mémoire, plusieurs datasets sont constitués (voir chapitre 4). Cette expérience se concentre sur les repères des mains et du squelette. Étant donné que l'on se concentre sur la segmentation des signes réalisés avec la main droite, les combinaisons de datasets évaluées sont :

- *right hand* : uniquement les repères de la main droite ;
- *both hands* : les repères des deux mains ;
- *skeleton* : uniquement les repères du squelette supérieur ;
- *skeleton + right hand* : les repères du squelette supérieur et de la main droite ;
- *skeleton + both hands* : les repères du squelette supérieur et des deux mains.

Résultats

dataset	acc.	bal. acc.	recall		
			signing	waiting	coarticulation
right hand	75,80	68,83	54,37	91,14	60,98
both hands	79,01	70,71	66,49	91,66	53,97
skeleton	71,87	66,42	40,41	90,21	68,65
skeleton + right hand	76,44	70,15	52,16	92,41	65,88
skeleton + both hands	78,10	**71,14**	60,60	91,75	61,07

TABLE 7.1 – Performance de la segmentation en fonction du dataset.

Conclusion

Dans cette expérience, l'utilisation des repères de la main donne de meilleurs résultats que l'utilisation des repères du squelette supérieur. La balanced accuracy (voir Table 7.1) montre que se contenter des repères du squelette supérieur n'est pas optimal.

Ensuite, l'utilisation des repères des deux mains du signeur donne de meilleurs résultats, alors même que la segmentation ne s'intéresse qu'aux signes réalisés avec la main droite. Le recall de la classe *coarticulation* est plus élevé pour le squelette seul qu'avec les mains. Cependant, cette augmentation est contrebalancée par une diminution significative du recall de la classe *signing*. Le squelette, avec les mains, a une meilleure balanced accuracy et est donc plus performant malgré son recall plus faible pour la coarticulation.

Finalement, la combinaison sélectionnée (en gras dans les résultats) est le squelette supérieur ainsi que les repères des deux mains. En effet, la meilleure balanced accuracy (voir Table 7.1) est obtenue à partir de cette configuration. C'est donc celle-ci qui est sélectionnée par la suite ainsi que dans les benchmarks suivants.

7.3 Choix de la loss function

loss function	paramètres	poids
Cross Entropy (CE)	/	Oui
Focal Loss (FL)	γ	Oui
Dice Loss (DL)	/	Non
Tversky Loss (TL)	α et β	Non

TABLE 7.2 – Résumé des loss functions

Un facteur déterminant dans les performance d'un modèle de deep learning est la loss function. Jusqu'à présent, dans le cadre de la segmentation LS, seule la cross entropy loss (CE) [26] était utilisée. Cette section compare les performances de différentes loss functions (voir Table 7.2) afin de sélectionner la plus pertinente.

Focal loss

Le chapitre 5 met en avant la problématique du non-balancement des données dans le cadre de la segmentation LS. Jusqu'à présent, afin de palier à ce problème, la solution présentée était l'ajout de poids α à la loss function. Cela peut être formalisé dans le cadre de la classification binaire comme suit :

$$\mathrm{CE}(p_t) = -\alpha \log(p_t)$$
$$p_t = \begin{cases} p & \text{si } y = 1 \\ 1 - p & \text{sinon.} \end{cases}$$

La focal loss (FL, voir Figure 7.1) modifie cette équation en ajoutant un facteur de modulation à la loss function. L'équation devient donc :

$$\mathrm{FL}(p_t) = -\alpha(1 - p_t)^{\gamma} \log(p_t)$$

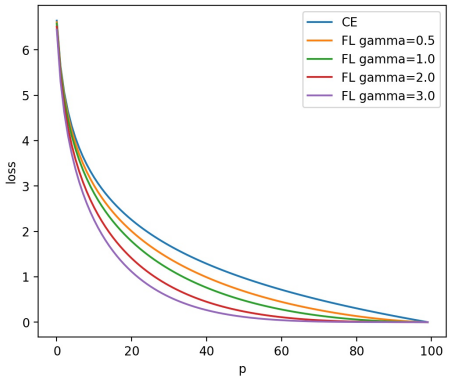


FIGURE 7.1 – Comparaison entre la cross entropy loss et la focal loss suivant le paramètre γ.

On obtient en plus des poids α un nouveau paramètre γ. Il réduit ou augmente l'importance du facteur de modulation. Celui-ci permet d'ajuster l'importance des classes plus difficiles à classifier. Dès lors, si $\gamma = 0$, la FL loss est équivalente à la CE loss. Tandis que si, par exemple, $\gamma = 2$, tel que recommandé dans la littérature [59], les classes plus difficiles sont privilégiées lors de l'entraînement. Dans le benchmark, la valeur de γ est considérée à 2.

Dice loss

La dice loss [60] est fort différente de la CE loss et de la FL loss. Celle-ci est généralement utilisée dans la segmentation d'images (segmentation sémantique) [61], par exemple des images médicales où l'on veut mettre en avant un zone particulière telle qu'une tumeur dans une image de cerveau. Dans le cadre de ce mémoire, le problème de la segmentation LS peut être considéré comme équivalent à la segmentation d'une image dont la hauteur est 1 pixel et la longueur N pixels. Chacun de ces pixels $(1, t)$ est remplacé par un squelette.

Pour une classe donnée, p_i représente les prédictions du modèle sous forme de probabilités pour cette classe, tandis que g_i représente les valeurs cibles pour cette classe. Le dice score est un score représentant à quel point les zones prédites (segmentation) correspondent aux zones cibles. Il est calculée comme suit :

$$D(p_i, g_i) = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 \sum_i^N g_i^2}$$

où le numérateur représente l'intersection entre les régions prédites et les régions cibles, tandis que le dénominateur normalise le score avec la somme des régions prédites et ciblées. Dans le cas de deux prédictions différentes, celle dont l'intersection est la plus grande obtient donc un score plus important.

Dans le cas de la segmentation LS, un score général D_m est obtenu en calculant la moyenne des dice scores de chaque classe. Ensuite, comme ce score doit être maximisé, on prend son opposé comme fonction de perte. La dice loss (DL) est calculée comme suit :

$$\text{DL} = 1 - D_m$$

Tversky loss

La Tversky loss (TL) [62] est similaire à la dice loss. Cependant, des paramètres supplémentaires sont ajoutés afin de mieux gérer le non-balancement dans les données. L'équation du Tversky score est :

$$T(p_i, g_i, \alpha, \beta) = \frac{\sum_i^N p_i g_i}{\sum_i^N p_i g_i + \alpha \sum_i^N (1 - p_i) g_i + \beta \sum_i^N p_i (1 - g_i)}$$

Les paramètres α et β sont des poids qui pénalisent, respectivement, l'oubli d'une région et la prédiction d'une fausse région. Ils permettent donc de moduler l'importance des faux positifs et faux négatifs. Si $\alpha = \beta = 1/2$, l'équation du Tversky score devient celle du dice score. Plusieurs configurations de la Tversky scores sont évaluées dans le benchmark. Celles-ci sont arbitrairement sélectionnées en faisant varier le rapport entre α et β :

- $\alpha = 1$ et $\beta = 1$;
- $\alpha = 0,3$ et $\beta = 0,7$;
- $\alpha = 0,1$ et $\beta = 0,9$.

Afin de transformer ce score en loss function, la moyenne T_m des Tversky scores de chaque classe est prise. Ensuite, la Tversky loss (TL) est calculée comme suit :

$$\text{TL} = 1 - T_m$$

Résultats

loss function	acc.	bal. acc.	recall		
			signing	waiting	coarticulation
CE	79,26	70,61	69,43	91,06	51,34
FL	79,82	**71,84**	69,16	91,39	54,98
DL	77,26	66,87	63,61	92,23	44,76
TL($\alpha = 1, \beta = 1$)	76,76	63,33	71,06	90,94	28,00
TL($\alpha = 0.3, \beta = 0.7$)	76,76	63,33	71,06	90,94	28,00
TL($\alpha = 0.1, \beta = 0.9$)	76,76	63,33	71,06	90,94	28,00

TABLE 7.3 – Performance de la segmentation en fonction de la loss function.

Conclusion

Les résultats (voir Table 7.3 montrent que les métriques issues de la segmentation d'images [61] engendrent des performances moindres que la cross entropy et la focal loss. Elles ne sont donc pas utilisées dans les modèles finaux. De façon similaire à l'expérience précédente, la focal loss, par rapport à la cross entropy, a un recall plus faible pour la classe *coarticulation* mais modélise bien mieux les classes *signing* et *waiting*. En conséquence, la cross entropy n'est pas la métrique la plus performante. En effet, la focal loss (en gras dans la Table 7.3) engendre une balanced accuracy plus élevée. Les modèles finaux utilisent donc la focal loss comme loss function.

7.4 Architecture de la recurrent layer

Les cellules récurrentes, dans ce cas une cellule LSTM (voir section 3.3.2), peuvent être combinées pour former des réseaux avec des architectures plus complexes. Deux modifications architecturales sont évaluées dans ce benchmark :

- multiplier les couches récurrentes ;
- utiliser une architecture bidirectionelle.

Une première modification possible est la multiplication des couches récurrentes. En effet, au même titre qu'un ANN (voir section 3.2.2) classique, un RNN peut avoir plusieurs couches intermédiaires qui se suivent [63]. Ensuite, une seconde modification courante dans les RNN est l'utilisation d'un réseau récurrent bidirectionnel. Dans un RNN bidirectionnel, une couche intermédiaire comprend deux couches récurrentes : la première traite la séquence normalement, de l'instant 0 à l'instant T. Tandis que la seconde prend comme entrée la séquence inversée, de l'instant T à l'instant 0. La couche intermédiaire suivante prend donc deux fois plus d'entrées que dans un RNN unidirectionnel [63]. Cela permet de mieux tirer parti de l'information séquentielle dans les deux directions.

Plusieurs modèles sont évalués dans cette section en fonction de leur nombre de couches récurrentes et de s'il s'agit d'un modèle bidirectionnel ou non. La Table 7.4 résume ces différentes configurations de modèles.

architecture	# couches récurrentes	bidirectionnel
1-layer LSTM	1	Non
1-layer BiLSTM	1	Oui
2-layer LSTM	2	Non
2-layer BiLSTM	2	Oui
4-layer LSTM	4	Non
4-layer BiLSTM	4	Oui

Table 7.4 – Architectures récurrentes évaluées dans le benchmark.

Résultats

architecture	acc.	bal. acc.	recall		
			signing	waiting	coarticulation
1-layer LSTM	79,19	71,23	67,55	91,14	55,02
1-layer BiLSTM	77,47	71,56	55,87	91,98	66,81
2-layer LSTM	76,59	69,93	60,75	89,23	59,81
2-layer BiLSTM	80,36	**72,67**	65,99	93,26	58,75
4-layer LSTM	71,33	65,45	34,79	92,24	69,32
4-layer BiLSTM	79,97	72,37	62,84	94,01	60,28

Table 7.5 – Performance de la segmentation en fonction de la loss function.

Conclusion

Utiliser une architecture bidirectionnelle améliore les performances du modèle, quel que soit le nombre de couches récurrentes. En effet, la balanced accuracy des modèles bidirectionnels (voir Table 7.5) est toujours plus élevée.

Finalement, l'architecture possédant deux couches récurrentes et étant bidirectionnelle (en gras dans la Table 7.5) est la plus performante. En effet, comme dans les expériences précédentes, son recall de la classe *coarticulation* est plus faible, mais sa balanced accuracy est la plus élevée. Cette expérience prouve que l'architecture du modèle peut être optimisée afin d'avoir de meilleures performances.

7.5 Résultats finaux

À partir des conclusions des expériences menées dans ce chapitre, ainsi que les conclusions des chapitres 5 et 6, les systèmes de segmentation LS finaux peuvent être constitués et évalués. La Table 7.6 montre la configuration sélectionnée pour le modèle final tandis que la Table 7.7 montre les résultats finaux pour plusieurs tâches de segmentation LS différentes :

- deux mains : la segmentation des signes réalisés quelque soit les mains utilisées
- main droite : la segmentation des signes réalisés avec la main droite
- main gauche : la segmentation des signes réalisés avec la main gauche

Une dernière tâche de segmentation est réalisée. De façon analogue à la segmentation de phrases dans un texte, celle-ci se concentre sur les longues périodes d'activité du signeur et non sur les signes individuels. Cette tâche est réalisée en fusionnant les classes *signing* et *coarticulation* dans une nouvele classe *signing*. La Table 7.8 montre ses résultats.

type de LSTM	Mogrifier LSTM
dataset	skeleton + both Hands
loss function	focal loss (FL)
architecture	2-layers BiLSTM
# epochs	200

Table 7.6 – Configuration des modèles finaux

tâche de segmentation	acc.	bal. acc.	recall		
			signing	waiting	coarticulation
main droite	80,98	75,18	63,49	93,61	68,45
main gauche	83,05	76,01	70,50	88,18	69,35
deux mains	80,51	74,90	64,84	92,89	66,96

Table 7.7 – Résultats finaux de la segmentation des signes selon les mains utilisées par le signeur.

tâche de segmentation	acc.	bal. acc.	recall	
			signing	waiting
périodes d'activité	95,42	95,45	95,92	94,99

TABLE 7.8 – Résultats finaux de la segmentation pour la détection de longue périodes d'activité du signeur.

L'optimisation du système de segmentation automatique de la langue des signes améliore significativement les performances de celui-ci. En effet, on obtient une balanced accuracy $7,12\%$ supérieure à celle de la segmentation des signes réalisés dans le chapitre 6. Cette augmentation est essentiellement due à une meilleure modélisation de la coarticulation. En effet, le recall de la *coarticulation* est de $66,96\%$ dans les résultats contre $35,89\%$ dans le chapitre 6. Soit une augmentation de $31,07\%$.

7.6 Récapitulatif

Différentes expériences ont mis en avant des améliorations permettant d'obtenir de meilleures performances dans la segmentation des signes réalisés à partir de la main droite. Les résultats de ces expériences montrent que :

- le dataset le plus performant est la combinaison des repères du squelette supérieur ainsi que les repères des deux mains ;
- la fonction de loss la plus adaptée est la focal loss ;
- l'architecture obtenant les meilleurs résultats est un RNN bidirectionnel possédant deux couches récurrentes.

Chapitre 8

Importance des différentes données

Les chapitres 5, 6 et 7 aboutissent à un système de segmentation LS. Ce chapitre, à partir de ce système, évalue l'importance de chaque donnée (feature) que reçoit le modèle. Dans ce cas-ci, cela revient à évaluer l'importance des repères dans les différents datasets (voir chapitre 4) utilisés dans le système de segmentation LS :

- les 23 repères du squelette supérieur, décrits section 4.2.2 ;
- les 21 repères de la main, décrits section 4.2.1.

L'évaluation des repères de la main se concentre uniquement sur la main droite. Dès lors, le modèle utilisé ne prédit que les signes réalisés de la main droite. En effet, cette analyse donne des résultats similaires pour la main gauche. La méthode utilisée pour l'évaluation est décrite dans la section 8.1 tandis que les résultats pour le squelette supérieur et la main droite sont présentés dans, respectivement, les sections 8.2 et 8.3.

L'objectif d'une telle évaluation est, d'une part augmenter la confiance que l'on a dans le modèle de segmentation LS en étant critique sur l'importance des données telles que le modèle les perçoit, et d'une autre part ouvrir des pistes de réflexion quant à l'importance des différentes articulations dans un discours en langue des signes.

8.1 Méthode d'évaluation

La méthode sélectionnée afin d'évaluer l'importance des repères est le calcul de l'importance par permutation de features [64]. Elle consiste en la permutation des différentes valeurs d'une feature spécifique au seins des données afin de la rendre complètement inutile. Dès lors, les résultats du modèle sont impactés par cette perte d'information. Il suffit donc de calculer la loss du modèle (voir section 3.1) et de la comparer à la loss de référence, c'est-à-dire celle obtenue lorsqu'aucune feature n'est permutée. Si l'on répète cette opération pour chaque

feature, on obtient à chaque fois une valeur, issue de la comparaison, qui fait office de score d'importance.

En résumé, voici le procédé à suivre afin d'évaluer l'importance des features selon cette méthode :

1. calculer la loss de référence à partir du dataset de test ;
2. pour chaque feature (colonne dans les données) :
   (a) permuter les valeurs de la feature (de la colonne) ;
   (b) mesurer la loss ;
   (c) calculer la différence entre cette loss et la loss de référence ;
   (d) collecter le résultat qui fait office de score d'importance pour cette feature.

Une façon de faire est de tout simplement soustraire la loss de référence à la loss obtenue. En effet, si cette différence, obtenue après permutation d'une feature, est positive, cela signifie que l'élimination de cette feature engendre une loss plus importante et que donc cette feature est importante pour que le modèle prenne une décision. De façon analogue, si la différence obtenue est négative, la loss est plus faible après l'élimination de cette feature et donc celle-ci n'est pas pertinente pour le modèle. Plus la valeur obtenue est grande, plus l'élimination de la feature entraîne une chute des performances, et donc plus la feature concernée est importante.

Ensuite, il est intéressant de calculer le score des repères. Dans ce mémoire, **un repère consiste en 2 features** (x et y, voir chapitre 4). Dans cette évaluation, le score d'importance d'un repère est la moyenne des scores de ses features associées.

8.2 Importance des repères du squelette supérieur

Les résultats de l'évaluation de l'importance des données du dataset contenant les repères du squelette supérieur (voir section 4.2.2) sont présentés dans les Figures 8.2 et 8.2. Celles-ci montrent, respectivement, l'importance de chaque feature (coordonnées) ainsi que l'importance de chaque repère. La Figure 8.1 fournit une autre représentation, plus schématique, afin de mieux interpréter l'importance des repères du squelette. En effet, plus un cercle et vert foncé, plus le repère positionné en son centre est important.

Les résultats mettent en lumière plusieurs éléments. Premièrement, les repères les plus importants pour la prédiction des signes réalisés à la main droite sont raisonnablement les repères sur la main droite. Cela est fortement visible sur la figure 8.1 où les cercles verts foncés sont sur la main droite du signeur. Ensuite, une seconde observation est que les coudes jouent un rôle important dans la détection des signes. Autant le coude droit que le coude gauche. Alors que, d'après cette analyse, les repères concernant le visage sont les moins importants.

Finalement, la figure 8.2 montre que les features concernant l'axe y sont les plus importantes. Cela peut s'expliquer par le fait que les signeurs bougent plus souvent leurs mains verticalement qu'horizontalement lorsqu'ils effectuent des signes.

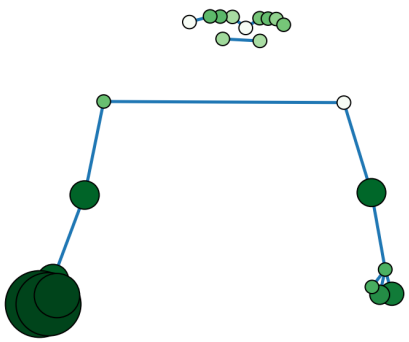


FIGURE 8.1 – Importance de chaque repère (x, y) sur le squelette supérieur du signeur.

8.3 Importance des repères de la main

Les résultats de l'évaluation de l'importance des données du dataset contenant les repères de la main droite (voir section 4.2.1) sont présentés dans les Figures 8.5 et 8.6 de façon similaire à l'évaluation des repères du squelette supérieur (voir section 8.2). En ce qui concerne la représentation plus schématique, Figure 8.4, celle-ci montre graphiquement l'importance des repères de la main. Les plus importants sont ceux concernant l'index, plus spécifiquement son extrémité. Ensuite, le petit doigt est le second plus important. Finalement, le pouce est le doigt le moins important.

D'après ces résultats, on peut émettre l'hypothèse selon laquelle l'index et le petit doigt combinés permettent d'avoir une bonne vue d'ensemble du mouvement de la main tandis que l'index est important, car celui-ci est souvent utilisé pour désigner une direction.

8.4 Récapitulatif

L'évaluation de l'importance des données permet de se rendre compte d'à quel point les données les plus significatives concernent les mains. En effet, dans le squelette, les repères concernant les mains et les coudes sont prépondérants, tandis que ceux concernant le visage le sont beaucoup moins.

Ensuite, en ce qui concerne les mains, les doigts les plus importants dans la segmentation des signes sont l'index et le petit doigt.

Il est également important de remarquer qu'aucune occurrence de résultat aberrant n'est apparue. Cela augmente la confiance que l'on peut avoir envers le système de segmentation LS. En effet, l'importance des repères tels que le modèle les perçoit semble compréhensible et peu d'hypothèses sont nécessaires afin de les expliquer. Cependant, la validation de ces hypothèses est laissée comme question ouverte.

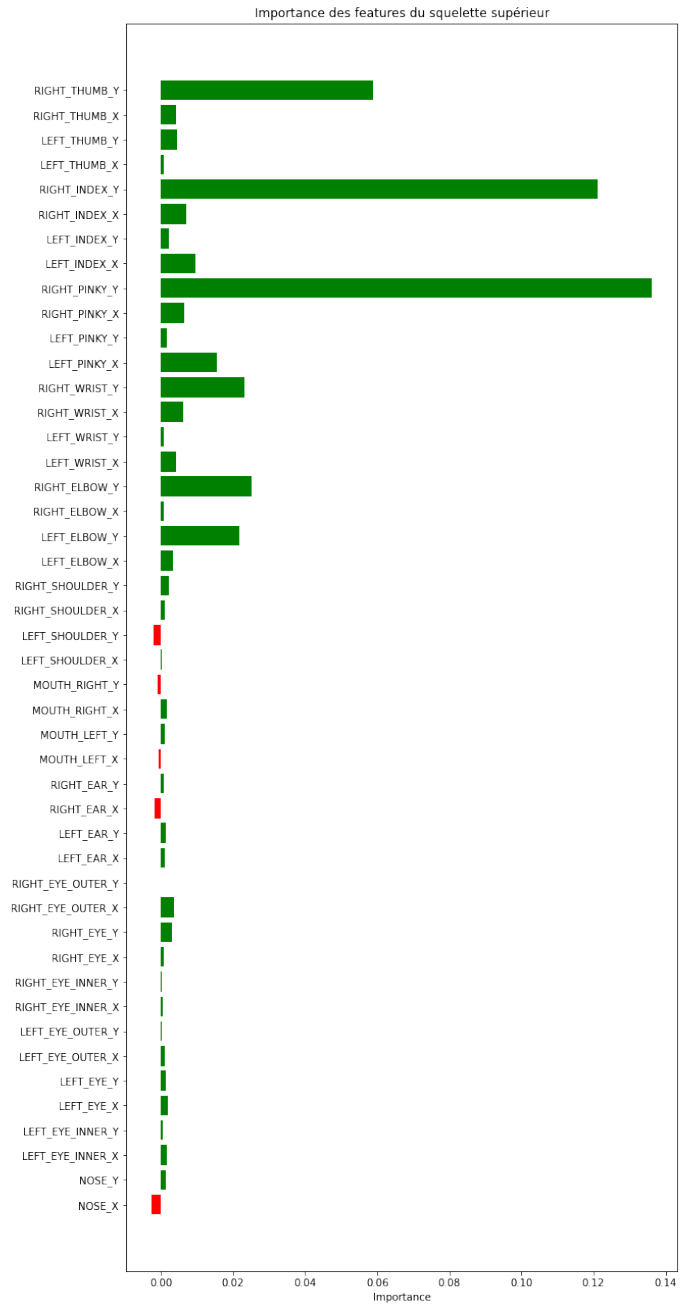


FIGURE 8.2 – Importance de chaque feature dans le dataset sur le squelette supérieur du signeur.

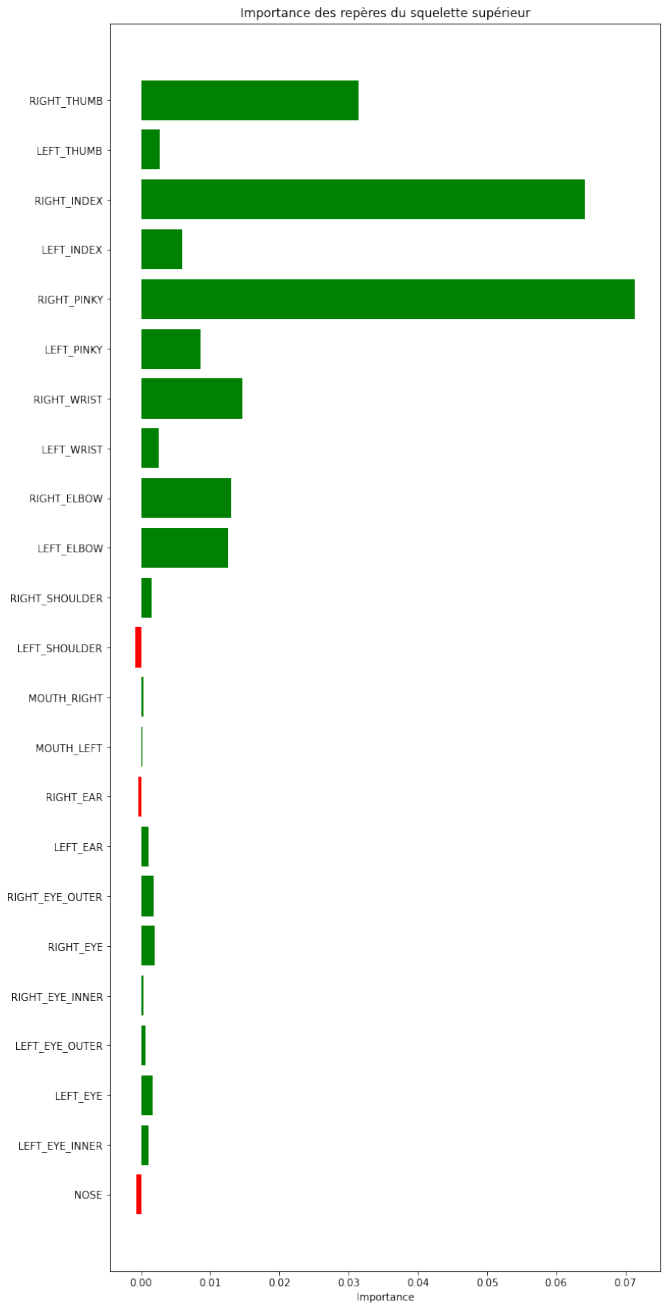


FIGURE 8.3 – Importance de chaque repère dans le dataset sur le squelette supérieur du signeur.

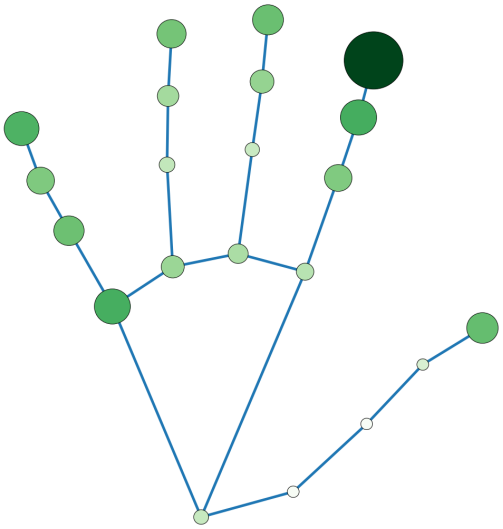


FIGURE 8.4 – Importance de chaque repère sur la main droite du signeur.

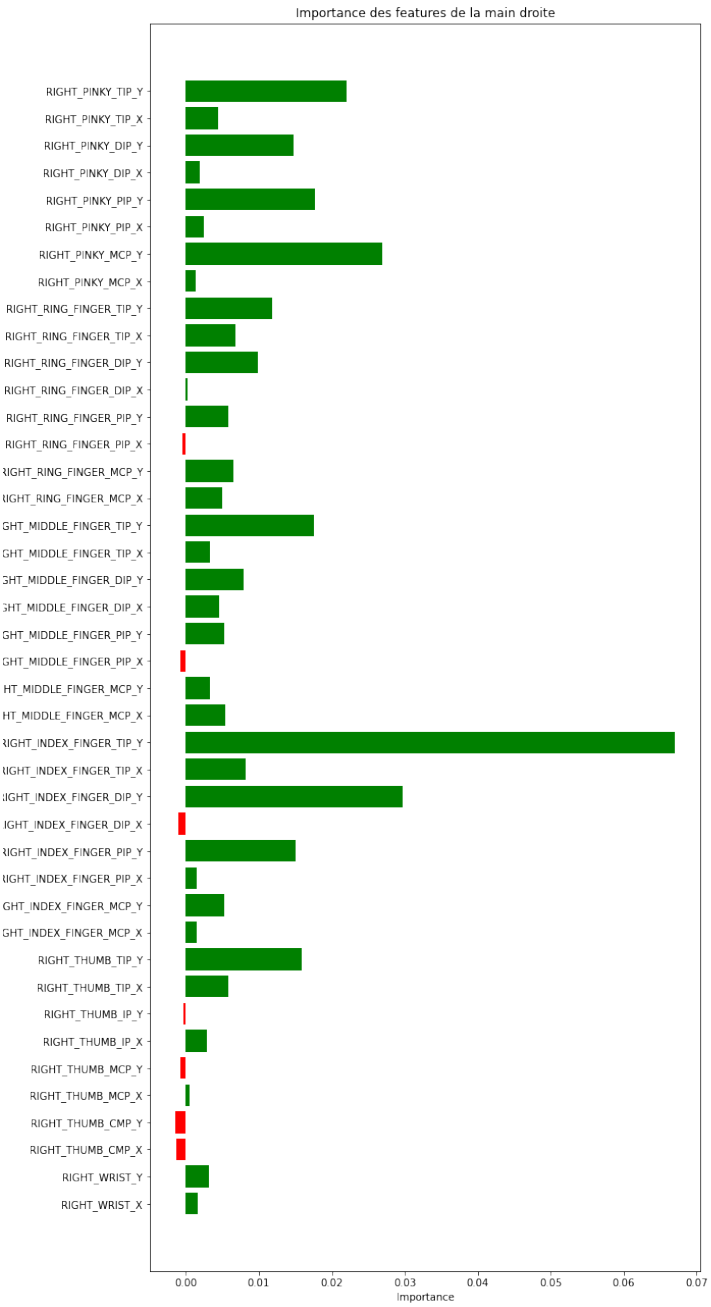


FIGURE 8.5 – Importance de chaque feature dans le dataset sur les mains du signeur. Seule la main droite est sélectionnée dans cet exemple.

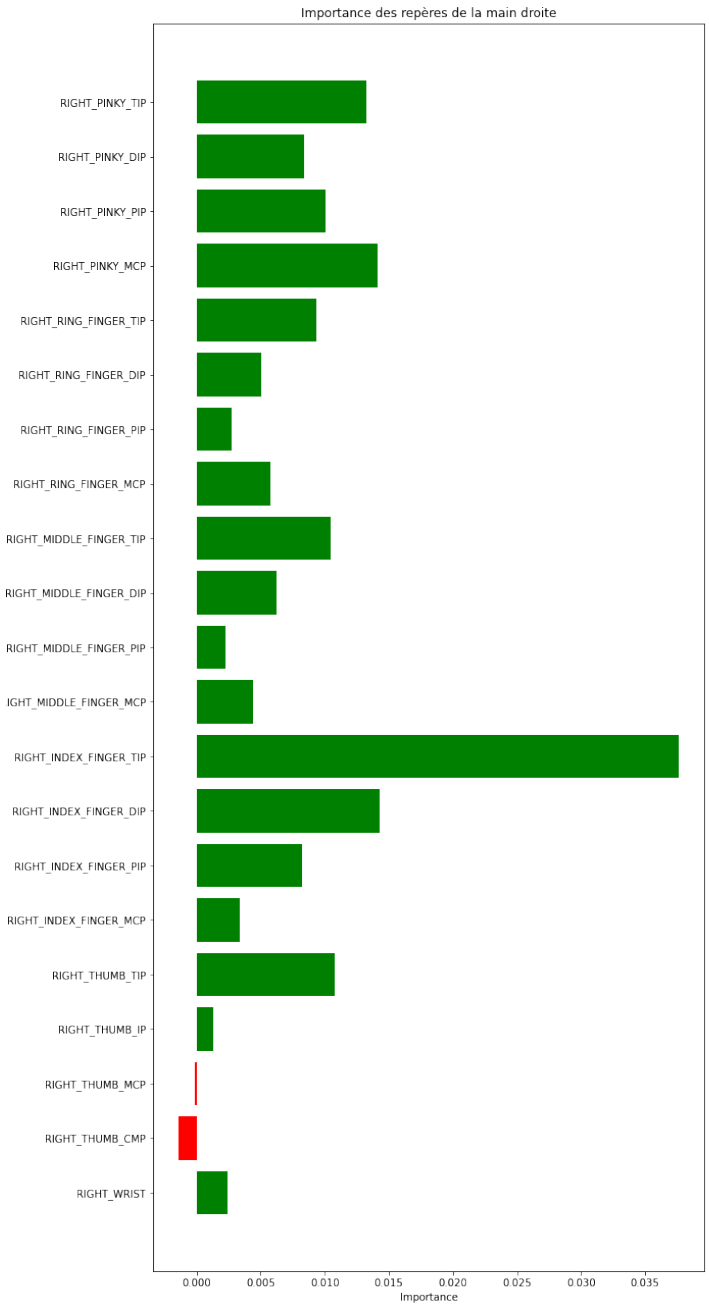


FIGURE 8.6 – Importance de chaque repère dans le dataset sur les mains du signeur. Seule la main droite est sélectionnée dans cet exemple.

Chapitre 9

Conclusion

Dans l'objectif de mettre en avant des pistes de solutions aux difficultés rencontrées dans la segmentation automatique de la langue des signes (segmentation LS), les chapitres 5 et 6 réalisent chacun une expérience montrant, respectivement, l'importance d'assigner un label aux zones de coarticulation et l'intérêt d'une meilleure modélisation des transitions dans les RNNs. Celles-ci sont également le sujet d'un papier scientifique réalisé dans le cadre de ce mémoire et soumis à la conférence internationale ESANN (voir annexes).

Dans la première expérience, les performances des modèles entraînés sont significativement meilleures lorsqu'une classe spécifique à la coarticulation est utilisée. Un problème survient lorsque la coarticulation est ignorée : le système de segmentation n'arrive pas à distinguer les signes proches et les considère comme un seul et long signe.

La seconde expérience, quant à elle, évalue deux extensions des LSTMs : l'EDRN et le mLSTM. Ceux-ci présentent des stratégies afin d'améliorer la modélisation des transitions entre états dans les LSTMs. Les performances de la segmentation sont plus élevées avec ces extensions. Cependant, la modélisation des transitions peut davantage être améliorée et aucune des extensions ne se démarque.

Pour atteindre un meilleur résultat, le système de segmentation automatique de la langue des signes est ajusté et optimisé dans le chapitre 7. Ses performances sont mesurées à l'aide de la balanced accuracy. Au travers de ces optimisations, sa balanced accuracy passe de 67.78% à $74,90\%$. La classe la plus impactée est la *coarticulation* dont le recall augmente en passant de $35,89\%$ à $66,96\%$.

En parallèle, les datasets créés sont décrits chapitre 4, alors qu'une évaluation de l'importance des données est menée chapitre 8. Celle-ci met en avant l'importance des mains et des coudes au niveau du squelette, tandis qu'au niveau des mains l'index et le petit doigt sont les plus importants. Cette évaluation permet de renforcer la confiance envers les résultats de la segmentation.

En conclusion, ce mémoire présente un système de segmentation automatique de vidéos continues en langue des signes. En outre, celui-ci est évalué en abordant des concepts tels que la coarticulation et la modélisation des transitions dans les RNNs. Des exemples de segmentation sont fournis dans la section 9.1.

Finalement, bien que fonctionnel, ce système peut encore être amélioré. C'est pourquoi la section 9.2 aborde des perspectives et des améliorations possibles.

9.1 Exemples de segmentation

Cette section présente différents exemples de segmentation d'une seule vidéo de discussion continue en langue des signes. Chaque exemple représente la segmentation cible présente dans le dataset (voir section 4.1) ainsi que la segmentation prédite.

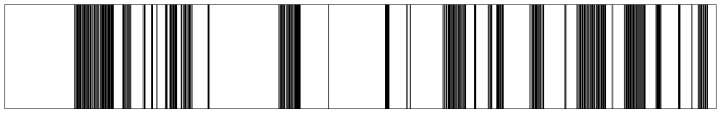
Le premier exemple concerne la segmentation des signes réalisés avec les deux mains (voir Figure 9.1), tandis que les suivants concernent les segmentation des signes réalisés avec, respectivement, la main droite (voir Figure 9.2) et la main gauche (voir Figure 9.3). On peut observer que les prédictions sont assez proches des segmentations cibles quelles que soient les mains concernées.

Un dernier exemple de segmentation (voir Figure 9.4) s'intéresse, lui, aux périodes d'activités du signeur. Celles-ci contiennent les labels *signing* et *coarticulation*. Ce modèle permet donc d'identifier l'activité du signeur sans pour autant distinguer les signes entre eux. Dans ce cas, la prédiction est très proches de la segmentation cible. Cela met en avant le fait que la difficulté de la segmentation LS réside dans la détection de la coarticulation (voir chapitre 5).

(a) Segmentation cible.

(b) Segmentation prédite.

FIGURE 9.1 – Exemple de segmentation des signes effectués à l'aide des deux mains. Les signes sont représentés en noir.



(a) Segmentation cible.

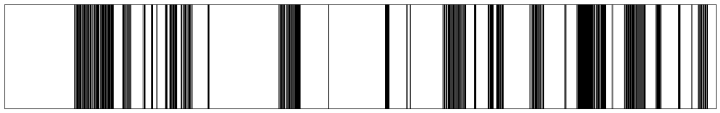
(b) Segmentation prédite.

FIGURE 9.2 – Exemple de segmentation des signes effectués à l'aide de la main droite. Les signes sont représentés en noir.

(a) Segmentation cible.

(b) Segmentation prédite.

FIGURE 9.3 – Exemple de segmentation des signes effectués à l'aide de la main gauche. Les signes sont représentés en noir.



(a) Annotations dans le dataset.

(b) Zones d'activité prédites.

FIGURE 9.4 – Exemple de segmentation des périodes d'activité du signeur. Elle est représentée en noir.

9.2 Perspectives et améliorations

Durant la réalisation de ce mémoire, toutes les pistes d'amélioration de la segmentation LS n'ont pas été évaluées à cause, essentiellement, de contraintes de temps et d'expertise. Cette section regroupe donc un ensemble d'améliorations et de perspectives ayant été identifiées et pouvant être développées dans de futures recherches :

- une première perspective est de tester la segmentation automatique dans une situation réelle afin d'évaluer son intérêt pour des annotateurs ;
- les réseaux récurrents (RNN, voir section 3.3.2) sont en concurrences avec d'autres techniques de deep learning, notamment les transformer networks [13]. Il serait pertinent d'évaluer les performances avec ceux-ci pour possiblement améliorer les résultats ;
- le chapitre 5 soulève l'importance de la prise en compte du non-balancement des données. Ce mémoire utilise des poids dans la loss function afin de palier à ce problème. Cependant, d'autres techniques existent et peuvent potentiellement améliorer les performances de la segmentation ;
- le chapitre 6 met en avant l'intérêt d'une meilleure modélisation des transitions dans les RNNs. Cet intérêt peut mener à des recherches supplémentaires. Un projet a d'ailleurs été lancé sur cette voie ;
- le chapitre 7, dans la section 7.3, aborde le problème de la segmentation à travers des concepts de la segmentation d'images (segmentation sémantique). Une perspective consiste en l'étude de la problématique de la segmentation LS à travers les techniques de segmentation sémantique ;
- dans le mémoire, les positions des points de repères (x et y) sont directement passées en entrée du modèle. Cependant, il existe des techniques de convolution telles que les Spatio-Temporal Graph Convolution Networks (STGCN) [65]. Celles-ci sont susceptibles d'améliorer les résultats ;
- les expériences au sein de ce mémoire utilisent exclusivement des points de repères calculés à partir des frames des vidéos. Cependant, l'utilisation des réseaux neuronaux à convolution (CNN) [30] directement sur les frames est également susceptible d'apporter de meilleurs résultats ;
- finalement, dans le système de segmentation automatique de la langue des signes tel que décrit et constitué dans ce mémoire, l'information sémantique des signes est ignorée. Une perspective serait de la prendre en compte afin d'améliorer les résultats.

Chapitre 10

Annexes

Towards Better Transition Modeling in Recurrent Neural Networks: the Case of Sign Language Tokenization

Pierre Poitier, Jérôme Fink and Benoît Frénay

University of Namur - NaDI - Faculty of Computer Science - PReCISE
rue Grandgagnage 21, B-5000 Namur - Belgium

Abstract. Recurrent neural networks can be used to segment sequences such as videos, where transitions can be challenging to detect. This paper benchmarks strategies to better model the transition between states. The specific task of SL video tokenization is chosen for the evaluation, as it remains challenging. Tokenizers are the cornerstone of natural language processing pipelines. There exist powerful tokenizers for text data, but sign language (SL) video tokenizers are still under development. Benchmarked strategies prove to be useful to improve SL videos tokenization, but there is still room for improvement to better model state transitions.

1 Introduction

Recurrent architectures have proven to be effective at processing sequential data. However, such models suffer from limitations that may prevent them to correctly model state transitions. Several architectures have been developed to mitigate those issues. This paper aims to assess their effectiveness, with a specific focus on sign language tokenization. This task was chosen as it remains challenging and correct modelling of state transitions would greatly improve the performance for real-world videos. Section 2 introduces sequential models and strategies to better model state transitions, Section 3 discusses the complexity of sign language tokenization and presents the real-world dataset used in the experiments. Section 4 presents and discusses the experiments performed on the retained models. Finally, Section 5 concludes with future perspectives.

2 Sequential Models and Transition Modeling

Hidden Markov models (HMMs) [1] were among the first successful models for sequential data. However, they suffer from some limitations. First, they implicitly use a geometric distribution to approximate the duration of each state. This led to the creation of hidden semi-Markov models (HSMMs) [1, 2], where each state duration is explicitly modelled with a probability density function (PDF). A second related limitation is an assumption that state transitions only depend on the current state. Furthermore, observations are assumed to be conditionally independent, given the current state. These assumptions hinder the performance of HMMs on complex tasks. This led to the creation of recurrent neural networks (RNNs) [3] such as the long short-term memory (LSTM) [4].

Despite their achievements, LSTMs may fail to properly model state transitions. In this paper, two architectures that address this limitation are considered:

(i) the explicit duration recurrent network (EDRN) [5] that aims to better model state durations by relying on sub-states transitions and (ii) the Mogrifier LSTM (mLSTM) [6] that tackles the assumption that state transitions only depend on the current state. They were chosen as they represent the state of the art.

In an EDRN [5], each hidden state has D sub-states. Before moving to the next hidden state, sub-state transitions must be performed, i.e., transition to the next hidden state can be performed when the last sub-state is reached. Similar to an LSTM [4], each sub-state has a duration approximated by a geometric distribution [5]. Therefore, the duration of a state, decomposed into sub-states, is approximated by a mixture of geometric distributions and can model a complex duration distribution [5]. The standard LSTM is a specific case of EDRN.

The Mogrifier LSTM (mLSTM) [6] improves the LSTM expressiveness by using a context-dependent transition function. This is done by applying a gate between the input and the hidden state before the LSTM cell for several rounds. Consequently, the transition function depends on the current hidden state and the input. This process results in a contextualised representation of the input.

A benchmark of EDRN and mLSTM models against the standard LSTM architecture is conducted to evaluate **the improvement in transition modelling**. This benchmark is done in the specific case of sign language tokenization.

3 Sign Language Segmentation

In recent years, significant progress has been made in natural language processing (NLP). Tokenizers are important parts of NLP pipelines: this mandatory preprocessing step divides a document into a list of sequential tokens. While it is easy to find efficient tokenizers for written languages, it is not the case in sign language (SL). Unlike textual data where splits are based on white spaces or punctuation symbols, transitions between tokens are harder to detect in SL video recordings. For example, the end of a sign often overlaps with the beginning of the next one. The creation of a good SL tokenizer would be a great step toward SL translation. It could also speed up the currently manual and tedious annotations of SL video recordings, leading to even larger datasets.

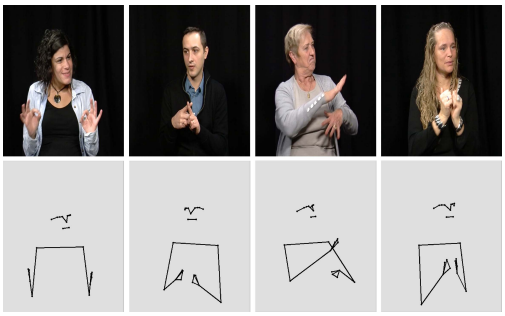


Fig. 1: Examples of frames with corresponding skeletons in LSFB-CONT.

Models are trained here with the large, real-world LSFB-CONT [7] dataset. It contains 50 FPS videos of individual people in real-life discussions without speed or vocabulary constraints. They are extracted from the LSFB Corpus [8], a large effort since 2012 by researchers at the University of Namur to collect and annotate LSFB (French Belgian Sign Language) conversations, with the aim to better understand this sign language. Consequently, it offers a large vocabulary (more than 6,000 words) with fast signs and short periods between them. One of the challenges is the imprecise nature of signs. Indeed, the boundaries of signs are highly dependent on the surrounding signs. The transition between two signs is called a *coarticulation* [9]. Conversations are annotated with the start and the end of all signs along with their labels. In order to avoid issues due to the large number of different signs and serious imbalance (10 most frequent signs account for 22% of annotations), we consider two simplified settings where annotations are replaced by a much smaller set of labels. In the two-class setting, labels are *talking* or *waiting*. In the three-class setting, labels are *talking*, *coarticulation* for the segments of less than 1 second between two signs and *waiting* for the others.

In this paper, tokenization is performed on the basis of pre-processed upper-body skeletons. Each skeleton consists of 23 landmarks extracted with MediaPipe [10] (see Figure 1). A model input is thus a sequence with n frames and 23 coordinates (x and y). Landmarks have been linearly interpolated to avoid discontinuity and coordinates are smoothed using a Savitzky-Golay filter [11] with a window length of 7 and a polynomial order of 2.

4 Benchmarking Transition Modelling in RNNs

This section evaluates the interest of models that aim to better model transitions, either through duration modelling (EDRNs) or contextual transitions (mLSTMs). The LSFB-CONT dataset [7] is used as described in Section 3 to create two segmentation tasks. Open research directions are highlighted.

4.1 Experimental Setting

Two experiments are carried out with two and three classes, respectively. LSTMs, EDRNs and mLSTMs are trained with a weighted cross-entropy loss, where frame weights are the inverse of the frequency of each class (see Table 1 for all details). The dataset consists of a subset of the LSFB-CONT dataset [7]: the training set and the validation set contain 3.5M and 2.2M frames, respectively. Signers in both sets are different. As sequences have varying dimensions, they are windowed for batches with a window size of 1500 and a stride of 800.

All the models have 128 features in their hidden states and end with a linear layer. The optimizer is a stochastic gradient descent with a learning rate of 0.01 and a momentum with a factor of 0.9. The EDRN is trained with 4 sub-states per hidden state and the Mogrifier LSTM uses 5 rounds [6] (see Table 2).

4.2 Experimental Results

Table 3 and 4 show the results of the two-class and three-class experiments, respectively. The metrics include the accuracy, the balanced accuracy and the

	classes	frames	frequency	weight
two-class setting	talking	1,836,565	31.72%	3.15
	waiting	3,953,383	68.28%	1.46
three-class setting	talking	1,836,565	31.72%	3.15
	waiting	3,176,663	54.87%	1.82
	coarticulation	776,720	13.41%	7.45

Table 1: Summary of the two experiments with two and three classes.

	input features	hidden features	sub-states	rounds
LSTM	46	128		
EDRN	46	128	4	
mLSTM	46	128		5

Table 2: Configuration of the models used in the two experiments.

recall for each class. In order to better apprehend the behaviour of the models, Figure 2 shows duration distributions for the *signing* and *coarticulation* classes in the three-class experiment. For reasons of readability, durations greater than 2s are not shown as we focus on the most significant part of the distribution. Figure 3 shows two examples of mistakes that occur for SL segmentation.

	acc.	bal. acc.	recall	
			talking	waiting
LSTM	81.22	80.53	63.10	**97.96**
EDRN	**82.12**	**81.06**	**64.41**	97.71
mLSTM	82.03	81.02	64.26	97.80

Table 3: Results of the two-class experiment with the LSFB-CONT dataset.

	acc.	bal. acc.	recall		
			talking	waiting	coarticulation
LSTM	73.82	66.19	70.13	97.21	31.24
EDRN	74.41	67.11	**72.02**	**97.32**	31.99
mLSTM	**77.65**	**67.78**	71.54	95.91	**35.89**

Table 4: Results of the three-class experiment with the LSFB-CONT dataset.

4.3 Discussion

In the two-class setting, EDRN outperforms both LSTM and mLSTM in terms of accuracy and balanced accuracy. However, all three models tend to ignore coarticulation and thus merge signs together as shown by Figure 3a. The three-class settings aims to mitigate this issue with the additional *coarticulation* class that reflect how signers move from one sign to the next. In that case, mLSTM performs better in terms of accuracy and has a better recall for *transition*. EDRN

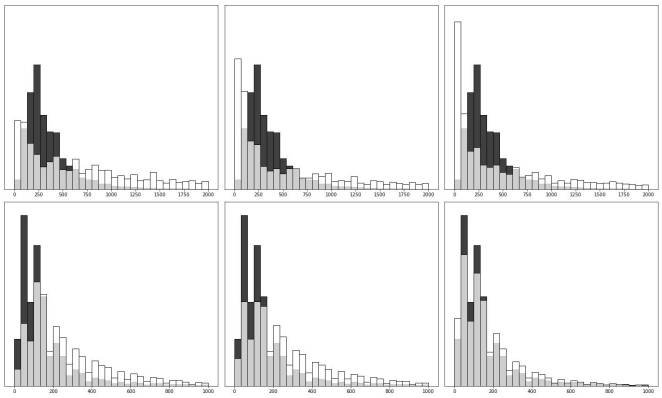
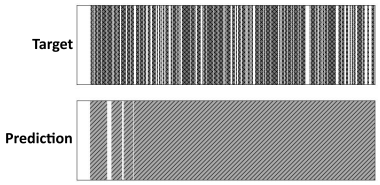


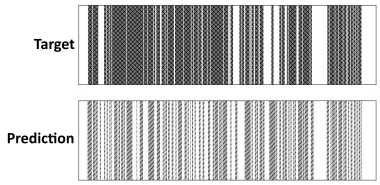
Fig. 2: Comparison of duration distributions in the dataset (black) and in the segmentation (white), for *signing* (first row) and *coarticulation* (second row) in the three-class experiment. From left to right: LSTM, EDRN and mLSTM.

has a better recall for other classes. In both settings, EDRNs and mLSTMs outperform LSTMs, showing the effect of better transitions modelling .

Figure 2 shows that the duration distributions are better modeled by the EDRN and the mLSTM. In particular, mLSTM is significantly more accurate for the *coarticulation* class, with most of the probability mass put in the left part of the distribution and a smaller distribution tail. Yet, there is still room for improvement. The mLSTM model tends to produce very short signs as shown by the segmentation example in Figure 3b. The EDRN also favours short signs but the problem is less important. However, the *coarticulation* duration distribution has a rather heavy tail in Figure 2, i.e., predicted transitions are too long. Notice that those observations are consistent with the higher transition recall and higher balanced accuracy measured for the mLSTM.



(a) Too long predicted signs with an LSTM in the two-class experiment.



(b) Too short predicted signs with an mLSTM in the three-class experiment.

Fig. 3: Examples of target (top) and predicted (bottom) segmentation. Black segments correspond to *talking*, white space indicates *waiting* or *coarticulation*.

5 Conclusion

In this paper, two experiments are carried out on the real-world task of sign language tokenization. Three RNN models are compared, which differ in the modeling of state durations and state transitions: LSTM, EDRN and mLSTM.

The first experiment highlights the difficulty of the prediction of coarticulation between signs. Both the EDRN and the mLSTM outperform the LSTM in the second experiment. mLSTM is more accurate, but it predicts too short signs. The EDRN is more accurate than the LSTM and predicts longer signs compared to the mLSTM. However, predicted coarticulations are too long.

This paper demonstrates the interest of developing better models of state transitions and state durations for RNNs through the case of SL tokenization. However, it also demonstrates that there is room for improvement. Future work includes proposing new powerful models in that direction. To the best of our knowledge, there exist only few works [12, 13], that tackle SL tokenization, but they do not focus on the problem of state transitions and durations.

Acknowledgments

This work is supported by the Funds InBev-Baillet Latour and the F.R.S.-FNRS EOS VeriLearn project n. 30992574. The authors thank Valentin Delchevalerie, Mohammed El Adoui and Géraldin Nanfack for their comments.

References

[1] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[2] S.-Z. Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2):215–243, 2010.

[3] D. E. Rumelhart, G. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California University San Diego, 1985.

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

[5] S.-Z. Yu. Explicit duration recurrent networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2021.

[6] G. Melis, T. Kočiskỳ, and P. Blunsom. Mogrifier LSTM. *arXiv:1909.01792*, 2019.

[7] J. Fink, B. Frénay, L. Meurant, and A. Cleve. LSFB-CONT and LSFB-ISOL: Two new datasets for vision-based sign language recognition. In *Proc. IJCNN*, pages 1–8, 2021.

[8] L. Meurant. Corpus LSFB. Corpus informatisé en libre acces de vidéo et d'annotations de langue des signes de Belgique francophone. Namur: Laboratoire de langue des signes de Belgique francophone (LSFB Lab), FRS-FNRS, Université de Namur, 2015.

[9] L. Naert, C. Larboulette, and S. Gibet. Coarticulation analysis for sign language synthesis. In *Proc. UAHCI*, pages 55–75, 2017.

[10] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann. Mediapipe: A framework for building perception pipelines. *arXiv:1906.08172*, 2019.

[11] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8):1627–1639, 1964.

[12] J. Zheng, Z. Zhao, M. Chen, J. Chen, C. Wu, Y. Chen, X. Shi, and Y. Tong. An improved sign language translation model with explainable adaptations for processing long sign sentences. *Computational Intelligence and Neuroscience*, 2020, 2020.

[13] A. Orbay and L. Akarun. Neural sign language translation by learning tokenization. In *Proc. IEEE FG*, pages 222–228, 2020.

Bibliographie

[1] Oscar Koller. Quantitative survey of the state of the art in sign language recognition. *arXiv preprint arXiv :2008.09918*, 2020.

[2] Ahmad Sami Al-Shamayleh, Rodina Ahmad, Mohammad AM Abushariah, Khubaib Amjad Alam, and Nazean Jomhari. A systematic literature review on vision based gesture recognition techniques. *Multimedia Tools and Applications*, 77(21) :28121–28184, 2018.

[3] Laurence Meurant and Magaly Ghesquière. *École et surdité : Une expérience d'enseignement bilingue et inclusif*, volume 1. Presses universitaires de Namur, 2018.

[4] Laurence Meurant, Maxime Gobert, and Anthony Cleve. Modelling a parallel corpus of French and French Belgian Sign Language. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4236–4240, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

[5] Laurence Meurant. Corpus lsfb. corpus informatisé en libre accès de vidéo et d'annotations de langue des signes de belgique francophone, 2015.

[6] J. Fink, B. Frénay, L. Meurant, and A. Cleve. LSFB-CONT and LSFB-ISOL : Two new datasets for vision-based sign language recognition. In *Proc. IJCNN*, pages 1–8, 2021.

[7] Ying Wu and Thomas S Huang. Vision-based gesture recognition : A review. In *International gesture workshop*, pages 103–115. Springer, 1999.

[8] Alan F Newell and Peter Gregor. Extra-ordinary human–machine interaction : what can be learned from people with disabilities ? *Cognition, Technology & Work*, 1(2) :78–85, 1999.

[9] Zahoor Zafrulla, Helene Brashear, Thad Starner, Harley Hamilton, and Peter Presti. American sign language recognition with the kinect. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 279–286, 2011.

[10] Simon Lang, Marco Block, and Raúl Rojas. Sign language recognition using kinect. In *International Conference on Artificial Intelligence and Soft Computing*, pages 394–402. Springer, 2012.

[11] Wen Gao, Jiyong Ma, Jiangqin Wu, and Chunli Wang. Sign language recognition based on hmm/ann/dp. *International journal of pattern recognition and artificial intelligence*, 14(05) :587–602, 2000.

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

[13] Mathieu De Coster, Mieke Van Herreweghe, and Joni Dambre. Sign language recognition with transformer networks. In *12th International Conference on Language Resources and Evaluation*, pages 6018–6024. European Language Resources Association (ELRA), 2020.

[14] Erik Cambria and Bebo White. Jumping nlp curves : A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2) :48–57, 2014.

[15] KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[16] S Vijayarani, R Janani, et al. Text mining : open source tokenization tools-an analysis. *Advanced Computational Intelligence : An International Journal (ACII)*, 3(1) :37–47, 2016.

[17] L. Naert, C. Larboulette, and S. Gibet. Coarticulation analysis for sign language synthesis. In *Proc. UAHCI*, pages 55–75. Springer, 2017.

[18] R Elakkiya and K Selvamani. Enhanced dynamic programming approach for subunit modelling to handle segmentation and recognition ambiguities in sign language. *Journal of Parallel and Distributed Computing*, 117 :246–255, 2018.

[19] Ruiduo Yang, Sudeep Sarkar, and Barbara Loeding. Handling movement epenthesis and hand segmentation ambiguities in continuous sign language recognition using nested dynamic programming. *IEEE transactions on pattern analysis and machine intelligence*, 32(3) :462–477, 2009.

[20] Hannah Bull, Michele Gouiffes, and Annelies Braffort. Automatic segmentation of sign language into subtitle-units. In *European Conference on Computer Vision*, pages 186–198. Springer, 2020.

[21] Jens Forster, Christoph Schmidt, Oscar Koller, Martin Bellgardt, and Hermann Ney. Extensions of the sign language recognition and translation corpus rwth-phoenix-weather. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1911–1916, 2014.

[22] Nikolas Adaloglou, Theocharis Chatzis, Ilias Papastratis, Andreas Stergioulas, Georgios Th Papadopoulos, Vassia Zacharopoulou, George J Xydopoulos, Klimnis Atzakas, Dimitris Papazachariou, and Petros Daras. A comprehensive study on sign language recognition methods. *arXiv preprint arXiv :2007.12530*, 2(2), 2020.

[23] Pierre-Emmanuel Aguera, Karim Jerbi, Anne Caclin, and Olivier Bertrand. Elan : a software package for analysis and visualization of meg, eeg, and lfp signals. *Computational intelligence and neuroscience*, 2011, 2011.

[24] Shu-Hsien Liao. Expert system methodologies and applications—a decade review from 1995 to 2004. *Expert systems with applications*, 28(1) :93–103, 2005.

[25] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315. Ieee, 2016.

[26] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[27] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training. *arXiv preprint arXiv :1903.03614*, 2019.

[28] Hans-Dieter Block. The perceptron : A model for brain functioning. i. *Reviews of Modern Physics*, 34(1) :123, 1962.

[29] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15) :2627–2636, 1998.

[30] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification : A comprehensive review. *Neural computation*, 29(9) :2352–2449, 2017.

[31] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv :1506.00019*, 2015.

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

[33] In Lee. Big data : Dimensions, evolution, impacts, and challenges. *Business horizons*, 60(3) :293–303, 2017.

[34] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet : A comprehensive survey on deep learning approaches. *arXiv preprint arXiv :1803.01164*, 2018.

[35] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[36] Thomas G Dietterich. Machine learning for sequential data : A review. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 15–30. Springer, 2002.

[37] Daphne Koller and Raya Fratkina. Using learning for approximation in stochastic processes. In *ICML*, pages 287–295, 1998.

[38] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–286, 1989.

[39] Michael A Benjamin, Robert A Rigby, and D Mikis Stasinopoulos. Generalized autoregressive moving average models. *Journal of the American Statistical association*, 98(461) :214–223, 2003.

[40] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields : Probabilistic models for segmenting and labeling sequence data. *Penn Libraries*, 2001.

[41] S.-Z. Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2) :215–243, 2010.

[42] Biing Hwang Juang and Laurence R Rabiner. Hidden markov models for speech recognition. *Technometrics*, 33(3) :251–272, 1991.

[43] John Henderson, Steven Salzberg, and Kenneth H Fasman. Finding genes in dna with a hidden markov model. *Journal of Computational Biology*, 4(2) :127–141, 1997.

[44] Bhavya Mor, Sunita Garhwal, and Ajay Kumar. A systematic review of hidden markov models and their applications. *Archives of computational methods in engineering*, 28(3) :1429–1448, 2021.

[45] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3) :268–278, 1973.

[46] Paul M Baggenstoss. A modified baum-welch algorithm for hidden markov models with multiple observation spaces. *IEEE Transactions on speech and audio processing*, 9(4) :411–416, 2001.

[47] Axel Cleeremans, David Servan-Schreiber, and James L McClelland. Finite state automata and simple recurrent networks. *Neural computation*, 1(3) :372–381, 1989.

[48] Duc Truong Pham and Xing Liu. Training of elman networks and dynamic system modelling. *International Journal of Systems Science*, 27(2) :221–226, 1996.

[49] Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[50] Mohammad Hossin and Md Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2) :1, 2015.

[51] Jerome Fink, Benoit Frenay, Laurence Meurant, and Anthony Cleve. LSFB-CONT and LSFB-ISOL : Two new datasets for vision-based sign language recognition. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2021.

[52] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann. Mediapipe : A framework for building perception pipelines. *arXiv :1906.08172*, 2019.

[53] Peter Kuffel, Kelvin Kent, and Garth Irwin. The implementation and effectiveness of linear interpolation within digital simulation. *International Journal of Electrical Power & Energy Systems*, 19(4) :221–227, 1997.

[54] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical chemistry*, 36(8) :1627–1639, 1964.

[55] S.-Z. Yu. Explicit duration recurrent networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2021.

[56] G. Melis, T. Kočiskỳ, and P. Blunsom. Mogrifier LSTM. *arXiv :1909.01792*, 2019.

[57] Gary Bishop, Greg Welch, et al. An introduction to the kalman filter. *Proc of SIGGRAPH, Course*, 8(27599-23175) :41, 2001.

[58] Leon Jay Gleser. The gamma distribution as a mixture of exponential distributions. *The American Statistician*, 43(2) :115–117, 1989.

[59] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[60] Jiachi Zhang, Xiaolei Shen, Tianqi Zhuo, and Hong Zhou. Brain tumor segmentation based on refined fully convolutional neural networks with a hierarchical dice loss, 2017.

[61] Yanming Guo, Yu Liu, Theodoros Georgiou, and Michael S Lew. A review of semantic segmentation using deep neural networks. *International journal of multimedia information retrieval*, 7(2) :87–93, 2018.

[62] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks. In *International workshop on machine learning in medical imaging*, pages 379–387. Springer, 2017.

[63] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks : Lstm cells and network architectures. *Neural computation*, 31(7) :1235–1270, 2019.

[64] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance : a corrected feature importance measure. *Bioinformatics*, 26(10) :1340–1347, 2010.

[65] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-second AAAI conference on artificial intelligence*, 2018.