Cohorte de réseaux de neurones récurrents pour la reconnaissance de l'écriture

Bruno Stuner

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat
Informatique

Préparée au sein de l'université de Rouen

Cohorte de Réseaux de Neurones Récurrents pour la Reconnaissance de l'Écriture

Présentée et soutenue par
Bruno STUNER

Manuscrit confidentiel jusqu'à cinq ans après la soutenance. Thèse soutenue publiquement le 11juin 2018 devant le jury composé de		
Laurence LIKFORMAN	PCH / HDR / Telecom ParisTech	Rapporteur
Christian WOLF	PCH / HDR / INSA de Lyon	Rapporteur
Florence D'ALCHÉ-BUC	PR / Telecom ParisTech	Examinateur
Éric ANQUETIL	PR / INSA de Rennes	Examinateur
Nicole VINCENT	PR / Université Paris Descartes	Examinateur
Thierry PAQUET	PR / Université de Rouen	Directeur de thèse
Clément CHATELAIN	MCF / INSA de Rouen	Examinateur, encadrant de thèse
Stéphane POIRIER	Dr / SOLYSTIC	Examinateur, encadrant de thèse

Thèse dirigée par Thierry PAQUET, **laboratoire LITIS- EA 4108, Université de ROUEN**

SOLYSTIC a NORTHROP GRUMMAN company

UNIVERSITÉ DE ROUEN NORMANDIE

litis

Remerciements

Un doctorat est une expérience unique en soi et très exigeante demandant curiosité, ingéniosité, travail et investissement. La rédaction de la thèse en est un concentré intense, aussi intense que le plus noir des cafés, café que je dus d'ailleurs me résoudre à boire pour surmonter cette épreuve. Pour moi, c'est probablement l'exercice le plus difficile. En effet, il exige une grande rigueur, un style rédactionnel de bonne tenue et une grande concentration. Seul ce paragraphe est assez facile à rédiger, où aucun style, ni rigueur y est exigé.

Je pense avoir vécu comme tout le monde les difficiles étapes de la thèse. Les rêves du début, prêt à révolutionner le monde, les premiers succès, les premiers accomplissements, puis les échecs, les remises en question et les désillusions. Mais j'espère, je crois, je pense ressortir grandi de toutes ces épreuves. Ces quelques mots qui vont suivre sont pour vous, vous tous qui m'ont accompagné, aidé, poussé dans des réflexions plus intenses et qui m'ont supporté au quotidien.

Je souhaite tout d'abord remercier mon directeur de thèse, Thierry PAQUET, pour m'avoir laissé librement travailler, sans jamais me contraindre, et m'avoir conseillé, écouté, guidé tout au long de cette thèse.

Je remercie ensuite mes deux encadrants, Clément CHATELAIN dans mon université et Stéphane POIRIER dans mon entreprise. Clément, merci de me suivre depuis bientôt 5 ans, merci pour ton soutien, tes conseils et toutes les réflexions que tu m'as apportées. Travailler avec toi a toujours été un plaisir. Stéphane, merci de m'avoir soutenu et merci pour nos nombreuses discussions de bureau qui m'ont toujours poussé à me dépasser. Je suis très heureux de continuer à tes côtés.

Un grand merci à Hicham EL BERNOUSSI, mon responsable hiérarchique, pour son implication dans ma thèse et son soutien dans ma mission. Un grand merci à Solystic de m'avoir accueilli dans l'entreprise et permis de faire cette thèse CIFRE dans d'excellentes conditions et avec un grand degré de liberté. Merci également à l'ANRT d'avoir contribué au financement de cette thèse.

J'aimerais aussi remercier celles et ceux qui lisent ces quelques lignes dans le but de m'évaluer. Un grand merci à Laurence LIKFORMAN-SULEM et Christian WOLF d'avoir accepté de rapporter ma thèse. Un grand merci à Isabelle BLOCH, Nicole VINCENT et Eric ANQUETIL d'avoir accepté d'examiner ma thèse.

Je remercie tous les collègues que j'ai pu côtoyer à Solystic, notamment les personnes de l'équipe OVS pour leur bonne humeur et la bonne ambiance. Merci également à tous les collègues du laboratoire LITIS pour nos discussions et tous les moments partagés en-

semble. J'aimerais donner des noms mais il y en a trop, alors merci à vous tous qui avez croisé mon chemin dans les couloirs de l'université ou d'Aristide à Bagneux.

Un grand merci à ma famille pour laquelle j'ai été peu présent durant ma thèse et qui m'a soutenu. Maman, papa, cent fois merci.

Les derniers mots vont à tous mes amis, parfois camarades de galère et traversant aussi cette épreuve qu'est la thèse. Vous avez toujours été présents et nous avons partagé de très bons moments, de nombreux voyages parfois à l'autre bout du monde. Merci Anne-Laure, Antoine, Aurore, Aymeric, Dimitri, Eddie, Erik, Laurent, Madeline, Maxime[2], Sophie. J'espère ne pas en oublier.

Encore un grand merci à tous et place à la science.

Résumé

Les méthodes à l'état de l'art de la reconnaissance de l'écriture sont fondées sur des réseaux de neurones récurrents (RNN) à cellules LSTM ayant des performances remarquables. Dans cette thèse, nous proposons deux nouveaux principes la **vérification lexicale** et la génération de **cohorte** afin d'attaquer les problèmes de la reconnaissance de l'écriture : i) le problème des grands lexiques et des décodages dirigés par le lexique ii) la problématique de combinaison de modèles optiques pour une meilleure reconnaissance iii) la nécessité de constituer de très grands ensembles de données étiquetées dans un contexte d'apprentissage profond. La vérification lexicale est une alternative aux décodages dirigés par le lexique peu étudiée à cause des faibles performances des modèles optiques historiques (HMM). Nous montrons dans cette thèse qu'elle constitue une alternative intéressante aux approches dirigées par le lexique lorsqu'elles s'appuient sur des modèles optiques très performants comme les RNN LSTM. La génération de cohorte permet de générer facilement et rapidement un grand nombre de réseaux récurrents complémentaires en un seul apprentissage. De ces deux techniques nous construisons et proposons un nouveau schéma de cascade pour la reconnaissance de mots isolés, une nouvelle combinaison au niveau ligne LV-ROVER et une nouvelle stratégie d'auto-apprentissage de RNN LSTM pour la reconnaissance de mots isolés. La cascade proposée permet de combiner avec la vérification lexicale des milliers de réseaux et atteint des résultats à l'état de l'art pour les bases Rimes et IAM. LV-ROVER a une complexité réduite par rapport à l'algorithme original ROVER et permet de combiner des centaines de réseaux sans modèle de langage tout en dépassant l'état de l'art pour la reconnaissance de lignes sur le jeu de données Rimes. Notre stratégie d'auto-apprentissage permet d'apprendre à partir d'un seul réseau BLSTM et sans paramètres grâce à la cohorte et la vérification lexicale, elle montre d'excellents résultats sur les bases Rimes et IAM.

Mots clés : Réseaux de neurones récurrents • LSTM • Reconnaissance de l'écriture • Vérification Lexicale • Cohorte • Combinaison en Cascade • LV-ROVER • Auto-apprentissage

Abstract

Title : Cohort of Recurrent Neural Networks for Handwriting Recognition

State-of-the-art methods for handwriting recognition are based on LSTM recurrent neural networks (RNN) which achieve high performance recognition. In this thesis, we propose the **lexicon verification** and the **cohort** generation as two new building blocs to tackle the problem of handwriting recognition which are : i) the large vocabulary problem and the use of lexicon driven methods ii) the combination of multiple optical models iii) the need for large labeled dataset for training RNN. The lexicon verification is an alternative to the lexicon driven decoding process and can deal with lexicons of 3 millions words. The cohort generation is a method to get easily and quickly a large number of complementary recurrent neural networks extracted from a single training. From these two new technichs we build and propose a new cascade scheme for isolated word recognition, a new line level combination LV-ROVER and a new self-training strategy to train LSTM RNN for isolated handwritten words recognition. The proposed cascade combines thousands of LSTM RNN with lexicon verification and achieves state-of-the art word recognition performance on the Rimes and IAM datasets. The Lexicon Verified ROVER : LV-ROVER, has a reduce complexity compare to the original ROVER algorithm and combine hundreds of recognizers without language models while achieving state of the art for handwritten line text on the RIMES dataset. Our self-training strategy use both labeled and unlabeled data with the unlabeled data being self-labeled by its own lexicon verified predictions. The strategy enables self-training with a single BLSTM and show excellent results on the Rimes and Iam datasets.

Keywords : Recurrent Neural Networks • LSTM • Handwriting Recognition • Lexicon Verification • Cohort • Cascade Combination • LV-ROVER • Self-Training

Table des matières

Liste des tableaux

Table des figures

Table des abréviations

Les acronymes utilisés dans ce document sont issus de la langue anglaise, le tableau suivant reprend ces acronymes.

Acronyme	Signification - *Traduction*
ANN	Artificial Neural Network - *Réseau de neurones artificiel*
BLSTM	Bidirectional Long Short Term Memory - *Réseau de neurones récurrent à cellules LSTM bidirectionnel*
BPTT	Backpropagation Through Time - *Rétropropagation à travers le temps*
BRNN	Bidirectional Recurrent Neural Network - *Réseau de neurones récurrent bidirectionnel*
CER	Character Error Rate - *Taux d'erreur caractère*
CNN	Convolutional Neural Network - *Réseau de neurones convolutif*
CTC	Connectionist Temporal Classification - *Classification temporelle connexionniste*
DNN	Deep Neural Network - *Réseau de neurones profond*
dpi	dots per inch - *points par pouce*
HMM	Hidden Markov Model - *Modèle de Markov caché*
ICDAR	International Conference on Document Analysis and Recognition
ICFHR	International Conference on Frontiers in Handwriting Recognition
LSTM	Long Short Term Memory - *Neurone à mémoire à long et court terme*
LV-ROVER	Lexicon Verified Recognizer Output Voting Error Reduction - *Réduction de l'erreur de sorties de reconnaisseurs par vote et vérification lexicale*
MDLSTM	Multi-Dimensional Long Short Term Memory - *Réseau de neurones récurrent à cellule LSTM multi-dimensionnel*
MLP	Multi Layer Perceptron - *Perceptron multi-couches*
OCR	Optical Character Recognition - *Reconnaissance optique de caractère*
OOV	Out Of Vocabulary - *Hors Vocabulaire*
RNN	Recurrent Neural Network - *Réseau de neurones récurrent*
ROVER	Recognizer Output Voting Error Reduction - *Réduction de l'erreur de sorties de reconnaisseurs par vote*
SGD	Stochastic Gradient Descent - *Descente de gradient stochastique*
WER	Word Error Rate - *Taux d'erreur mot*

Table 1 – Acronymes

Introduction

Les réseaux de neurones sont une famille de modèles d'apprentissage statistique ancienne dont l'origine remonte au neurone formel [McCulloch and Pitts, 1943] et à la création du perceptron de Rosenblatt [Rosenblatt, 1958]. L'apprentissage de réseaux de neurones s'avère cependant complexe en raison du besoin de nombreuses données, étiquetées ou non, et du temps nécessaire pour les apprendre. Pendant très longtemps, l'apprentissage des réseaux de neurones profonds étaient problématiques à cause du problème de la disparition du gradient [Bengio et al., 1994, Hochreiter, 1998], du temps d'apprentissage trop long et du manque de données. Un premier pas a été franchi en 1997 [Hochreiter and Schmidhuber, 1997] avec l'introduction des cellules de mémoires à long et court terme (LSTM) et en 2006 [Hinton et al., 2006] avec les réseaux de croyance profonds (DBN). Grâce à ces avancées mais aussi à l'amélioration des méthodes de calcul, l'apprentissage de réseaux profonds a été rendu possible et est devenu une technologie incontournable [LeCun et al., 2015]. C'est une technologie en plein essor qui a été appliquée avec succès pour résoudre des problèmes complexes dans de nombreux domaines de recherche comme la vision par ordinateur, la reconnaissance de formes, les interactions avec l'homme ou encore la biologie pour ne citer que ceux là.

Concernant les réseaux récurrents à cellules LSTM, ils ont été remis en avant en 2005 [Graves and Schmidhuber, 2005] avec une nouvelle méthode d'apprentissage permettant un bien moindre effort d'étiquetage des données. Les réseaux de mémoires récurrents à cellules LSTM obtiennent aujourd'hui des résultats à l'état de l'art pour la reconnaissance de séquences, comme en reconnaissance de la parole [Graves and Schmidhuber, 2005], en prédiction de protéines [Thireou and Reczko, 2007], en traduction automatique [Sutskever et al., 2014], et en reconnaissance de l'écriture [Grosicki and El Abed, 2009, Menasri et al., 2012].

Les réseaux de neurones convolutifs (CNN) ont été proposés en 1990 [LeCun et al., 1990] mais leur intérêt pratique n'est apparu que récemment en 2009 avec l'arrivée d'imagenet [Deng et al., 2009], une base de données étiquetées particulièrement grande. Les réseaux convolutifs profonds ont été popularisés par les résultats très prometteurs du réseau AlexNet [Krizhevsky et al., 2012] pour la classification d'images naturelles. D'autres applications notables des CNN sont la segmentation d'images bio-médicales [Ronneberger et al., 2015], la mise en couleur d'images en noir et blanc [Zhang et al., 2016] ou les jeux vidéos [Mnih et al., 2013]. Une équipe de Google Deepmind a très récemment permis de battre les meilleurs joueurs mondiaux au jeu de GO avec *AlphaGo Zero* [Silver et al., 2017] une architecture profonde basée sur des réseaux à convolutions.

La reconnaissance de l'écriture se décompose traditionnellement en deux étapes [Plamondon and Srihari, 2000] : la reconnaissance optique de caractères et les traitements linguistiques. Les méthodes à l'état de l'art de la reconnaissance optique de caractères sont basées sur les réseaux de neurones récurrents (RNN) à cellules de mémoire à long et

court terme (LSTM) [Graves and Schmidhuber, 2009, Voigtlaender et al., 2016]. Ils produisent des performances de reconnaissance de caractères élevées. Néanmoins, le grand nombre de paramètres de ces réseaux nécessite des jeux de données d'apprentissage annotés conséquent. Ces derniers sont difficiles à obtenir car l'annotation est très coûteuse en moyens humains. L'apprentissage non supervisé de réseaux de neurones à l'aide de données non étiquetées est l'une des prochaines frontières à dépasser et ce même pour une tâche comme la reconnaissance de caractères manuscrits. L'apprentissage semi-supervisé [Chapelle et al., 2009] est un compromis entre l'apprentissage supervisé (uniquement des données étiquetées) et l'apprentissage non supervisé (uniquement des données non étiquetées). Plusieurs auteurs [Ball and Srihari, 2009, Frinken et al., 2011, Frinken and Bunke, 2009] se sont attaqués au problème de l'apprentissage semi-supervisé de réseaux LSTM pour la reconnaissance de l'écriture. Ces méthodes se basent sur l'apprentissage de plusieurs classifieurs sélectionnés pour étiqueter de nouvelles données en utilisant des règles de validation basées sur des scores de confiance. Ces approches ont l'inconvénient de requérir à la fois l'entraînement de multiples classifieurs et de dépendre d'hyperparamètres.

En reconnaissance d'écriture, l'apprentissage de reconnaisseurs de caractères n'est pas la seule difficulté. En effet, la reconnaissance de l'écriture est la plupart du temps couplée à l'utilisation de ressources linguistiques. Il y a deux types de connaissances linguistiques : les lexiques et les modèles de langages. L'utilisation de ressources lexicales consiste généralement à réaliser un décodage dirigé par un lexique et dédié à l'application. Malheureusement, ces lexiques sont limités à quelques centaines de milliers de mots pour les meilleurs systèmes [Hamdani et al., 2014, Bluche et al., 2014a], ce qui empêche d'avoir une couverture lexicale totale, limitant ainsi les performances globales de reconnaissance. Concernant l'utilisation des modèles de langage, il s'agit d'une modélisation probabiliste d'une langue permettant de calculer la vraisemblance d'une hypothèse de reconnaissance au niveau phrase. Pour apprendre un modèle de langage il faut cependant disposer d'un corpus d'apprentissage dont le choix peut s'avérer délicat. L'objectif des méthodes dirigées par le lexique est de reconnaître les mots grâce à l'utilisation d'un lexique et d'un décodage de type recherche de faisceaux de Viterbi [Fissore et al., 1988]. Le mot du lexique correspondant le mieux à la concaténation des hypothèses de caractères est recherché par ces méthodes. Les méthodes dirigées par le lexique sont les plus utilisées et il n'existe pas à notre connaissance d'alternative efficace à cette stratégie. La question du lexique à utiliser est difficile à appréhender et affecte directement les performances de reconnaissance. Un lexique trop petit ne permet pas de couvrir suffisamment les données à reconnaître, manquant ainsi des solutions. Cependant l'utilisation d'un lexique trop grand (1000 mots ou plus) conduit à des temps de calcul plus élevés et à une baisse de la précision [Koerich et al., 2003].

En plus de l'utilisation d'un modèle linguistique pour fiabiliser la reconnaissance, bien souvent plusieurs systèmes complets de reconnaissance de l'écriture sont combinés afin d'accroître les performances. Les performances à l'état de l'art sont majoritairement obtenues en combinant de multiples modèles optiques ou de langage lors des compétitions [Sánchez et al., 2016]. Des méthodes de combinaison comme ROVER (Recognizer Output

Voting Error Reduction) [Fiscus, 1997] sont souvent utilisées. Cet algorithme permet de combiner des systèmes de reconnaissance complets dès lors qu'ils sont complémentaires. Il est composé de deux modules : un module d'alignement et un module de vote. L'entraînement de nombreux systèmes complémentaires (modèles optiques ou linguistiques) est difficile. C'est pourquoi, assez peu de reconnaisseurs (une vingtaine) sont combinés dans la littérature [Bluche et al., 2014b, Sánchez et al., 2016, Bertolami and Bunke, 2005].

Dans cette thèse nous nous concentrons sur la reconnaissance de l'écriture manuscrite avec des réseaux de neurones récurrents LSTM. Nous nous attachons plus particulièrement aux questions suivantes :

- Comment utiliser un réseau LSTM sans décodage dirigé par le lexique afin de contourner les problèmes liés aux lexiques ?
- Comment obtenir facilement et rapidement des reconnaisseurs (réseaux LSTM) ayant une bonne complémentarité afin de les combiner ?
- Comment combiner efficacement un grand nombre (des centaines voir des milliers) de réseaux LSTM afin d'améliorer les performances ?
- Comment apprendre des réseaux LSTM pour la reconnaissance de l'écriture avec un nombre restreint de données étiquetées en entrée ?

En réponse à ces questions nos contributions portent principalement sur l'introduction du principe de **cohorte** de classifieurs. Nous montrons que les réseaux récurrents peuvent être entraînés d'une manière spécifique pour obtenir un ensemble de classifieurs aux propriétés complémentaires. Ces classifieurs peuvent alors être combinés efficacement dans un schéma de cascade grâce à l'introduction d'un opérateur de **vérification lexicale**. La **cascade** proposée permet de combiner des milliers de réseaux récurrents issus de plusieurs cohortes et atteint des résultats à l'état de l'art pour les bases RIMES et IAM pour une tâche de reconnaissance de mots isolés. Ce principe est ensuite étendu à la reconnaissance de phrase. Nous modifions pour y parvenir l'approche ROVER en introduisant l'opérateur de vérification lexicale, et en développant une stratégie spécifique d'exploration des solutions. Cette approche, baptisée **LV-ROVER**, sur-passe l'état de l'art pour la reconnaissance de lignes sur la base RIMES. Enfin, nous examinons l'apport des principes de cohorte et de vérification lexicale, sur une tâche de reconnaissance semi-supervisée où nous réalisons un **auto-apprentissage** d'un réseau LSTM pour la reconnaissance de mots isolés. L'obtention d'une cohorte de reconnaisseurs par l'entraînement d'un seul réseau LSTM est particulièrement efficace pour imiter un schéma itératif d'apprentissage et d'étiquetage de données. Cette stratégie contrôlée par un opérateur de vérification lexicale permet d'obtenir d'excellents résultats vis à vis de l'état de l'art.

Nous présentons dans cette thèse des expérimentations pour valider nos propositions. Ces expérimentations ont été essentiellement conduites sur deux bases de données publiques : RIMES (français) et IAM (anglais). Nous obtenons sur ces deux bases de mul-

tiples résultats à l'état de l'art tant pour la reconnaissance de mots isolés, de lignes de texte que pour l'apprentissage semi supervisé de réseaux LSTM. De plus, nous présentons des résultats avec des lexiques de tailles gigantesques de quelques millions de mots, ce qui, à notre connaissance, n'a jamais été réalisé.

Cette thèse est divisée en 2 parties. La première partie est dédiée à l'état de l'art sur les réseaux de neurones et la reconnaissance de l'écriture . Dans le chapitre 1, nous présentons les différents types de réseaux de neurones, leur fonctionnement, ainsi que les algorithmes pour les apprendre. Nous portons un intérêt tout particulier à l'explication des réseaux récurrents et de la cellule de mémoire LSTM. Dans le chapitre 2, un bref historique des méthodes de reconnaissance de l'écriture est rappelé. Nous présentons également les méthodes actuellement à l'état de l'art et les bases de données utilisées pour évaluer ces systèmes.

Dans la seconde partie de cette thèse nous présentons nos contributions. Dans le chapitre 3, nous traitons de la reconnaissance de mots isolés. Nous commençons par exposer le problème des grands lexiques qui a initialement motivé nos travaux. Nous proposons un nouveau paradigme de reconnaissance fondé sur la vérification lexicale résolvant ce problème. Ce paradigme est couplé à une combinaison en cascade avec des milliers de réseaux LSTM complémentaires (cohorte) pour lesquels nous présentons une stratégie novatrice d'apprentissage. Dans le chapitre 4, nous réutilisons le principe de cohorte pour l'étendre à la reconnaissance de lignes. Pour ce faire, nous proposons une nouvelle stratégie de combinaison : LV-ROVER pour "Lexion Verified Recognizer Output Voting Error Reduction", qui est d'une complexité inférieure à celle de la combinaison ROVER dont elle s'inspire. Enfin, dans le chapitre 5 nous proposons une nouvelle méthode d'apprentissage semi-supervisé des réseaux LSTM pour la reconnaissance de mots isolés à partir de données étiquetées et non étiquetées. Notre stratégie se base également sur la vérification lexicale et le principe de cohorte.

Nous concluons cette thèse par un résumé de nos travaux et des résultats obtenus. Nous proposons également des perspectives et des recherches pouvant s'inscrire dans le prolongement de cette thèse.

Première partie

État de l'art

Chapitre 1

Réseaux de neurones artificiels

Contents

1.1 Introduction

Les réseaux de neurones artificiels sont une classe de méthodes d'apprentissage automatique, originellement inspirés des neurones biologiques. L'histoire des réseaux de neurones débute en 1958 avec le perceptron de Rosenblatt [Rosenblatt, 1958] et se poursuit jusqu'à aujourd'hui à des architectures profondes [LeCun et al., 2015] présentant un empilement d'un grand nombre de couches de neurones. Avec l'apprentissage profond les réseaux de neurones apprennent des mécanismes perceptifs propres et ne dépendant pas de l'implémentation donnée par le développeur. On les considère, à ce titre, comme faisant partie des méthodes d'intelligence artificielle. Il existe de nombreux types de réseaux de neurones et de méthodes d'apprentissage qui varient selon l'utilisation souhaitée. Dans ce chapitre nous présentons les différentes architectures de réseaux de neurones artificiels, leurs fonctionnements et spécificités, leurs principales applications, les méthodes permettant de les entraîner et d'optimiser leur performance sur une tâche spécifique.

Dans un premier temps, nous présentons les bases de l'apprentissage automatique qui nous permettent ainsi d'introduire les notions nécessaires pour comprendre les réseaux de neurones.

Dans un second temps, nous définissons les concepts clés des réseaux de neurones, leur fonctionnement et leur apprentissage à travers le perceptron multi-couches.

Nous présentons ensuite le principe d'un réseau de neurones profond, le problème de disparition du gradient et comment les algorithmes d'apprentissage permettent de le contourner.

Nous nous intéressons par ailleurs aux réseaux de neurones plus précisément étudiés dans cette thèse, les réseaux de neurones récurrents. Nous décrivons leur apprentissage et les cellules de mémoires à long et court terme (LSTM).

Enfin, nous présentons les autres types d'architectures profondes parmi lesquels les réseaux convolutifs avant de conclure ce chapitre.

1.2 Apprentissage automatique

L'apprentissage automatique (machine learning en anglais) est un domaine d'étude de l'intelligence artificielle. Il consiste à concevoir des méthodes capables d'évoluer et d'apprendre un comportement permettant de résoudre des tâches complexes sans jamais avoir codé explicitement comment les réaliser. Le but de ces méthodes est d'apprendre de manière empirique, à l'aide d'exemples d'une base de données ou de données de capteurs, à résoudre une tâche comme prédire une catégorie (classification) ou une valeur numérique (régression).

1.2.1 Les méthodes d'apprentissage automatiques

De nombreuses méthodes ou algorithmes d'apprentissage automatiques ont été utilisés dans la littérature. On peut citer notamment :

– La régression logistique [Berkson, 1944] ;
– Les réseaux de neurones [Rosenblatt, 1958] ;
– Les k plus proches voisins [Cover and Hart, 1967] ;
– Les arbres de décisions [Quinlan, 1986] et les forêts aléatoires [Breiman, 2001] ;
– Les algorithmes génétiques [Davis, 1991] ;
– Les machines à vecteur de support (SVM) [Vapnik, 1995].

Chaque méthode d'apprentissage possède ses spécificités permettant d'accomplir des tâches complexes, comme par exemple de classification ou de régression. Après que les réseaux de neurones aient connu un fort intérêt au cours des années 80, les machines à vecteur de support sont apparues comme des solutions algorithmiques plus abouties dans les années 90 [Cortes and Vapnik, 1995]. Les SVM mettent en avant des capacités théoriques à généraliser efficacement en grandes dimensions et de bonnes performances applicatives. Depuis les années 2010, les réseaux de neurones sont utilisés plus largement que les SVM grâce aux nouvelles techniques d'apprentissage profond [LeCun et al., 2015], à l'utilisation de calculs sur carte graphique et aux excellents résultats obtenus pour des tâches devenues plus complexes.

Les problèmes que cherchent à résoudre les méthodes d'apprentissage automatique sont nombreux. On peut les regrouper dans des grandes classes qui sont les problèmes :

– De classification : déterminer la catégorie d'une donnée, on parle aussi d'étiquette (label) pour distinguer la catégorie (e.g. donner le chiffre représenté par une image) ;
– De régression : déterminer une valeur à partir d'une donnée dans un espace continu (e.g. trouver des coordonnées dans une image) ;
– De clustering : séparer des données non étiquetées en groupes (cluster) (e.g. réunir des mots ayant la même sémantique) ;
– D'estimation de densité : trouver la densité de probabilité des données (e.g. estimer l'audience audiovisuelle à partir d'un échantillon sondé).

Nous allons étudier quelles sont les types d'apprentissage de ces méthodes leur permettant d'optimiser leur prédiction à partir d'exemples.

1.2.2 Les types d'apprentissage statistique

Quelle que soit la méthode d'apprentissage, la réalisation de son apprentissage revient à optimiser ses paramètres afin qu'à partir d'un signal d'entrée X la méthode puisse déduire la sortie Y attendue. Pour apprendre, l'objectif est de modifier les paramètres de l'algorithme en minimisant un critère (une fonction de coût), à l'aide de techniques d'optimisation comme la descente de gradient. La fonction de coût ou objectif dans un problème d'optimisation est la fonction qui relie un événement à un nombre réel représentant le coût. L'apprentissage repose sur des données d'entrée pouvant être étiquetées ou non pour lesquelles on évalue le critère.

Les trois types majeurs d'apprentissage sont : l'apprentissage supervisé, l'apprentissage par renforcement [Sutton and Barto, 1998] et l'apprentissage non supervisé [Sanger, 1989]. Il existe également d'autres types ou sous types tels que l'apprentissage semi-supervisé [Chapelle et al., 2009], à mi chemin entre le supervisé et le non supervisé, et l'apprentissage

par transfert [Pan and Yang, 2010].

L'apprentissage supervisé est l'apprentissage le plus utilisé. Il est difficile de dater sa première utilisation. Il consiste en un apprentissage à partir d'exemples X d'étiquette Z où l'on cherche à optimiser la sortie Y de telle sorte qu'elle soit la plus proche de Z. Cet apprentissage nécessite qu'un expert généralement humain réalise un étiquetage des données au préalable. L'apprentissage semi-supervisé [Chapelle et al., 2009] peut être considéré comme un sous type de l'apprentissage supervisé car ce sont des données étiquetées par un expert humain et des données non-étiquetées au départ et étiquetées par le processus qui sont utilisées. L'apprentissage par transfert [Pan and Yang, 2010] est un sous-type d'apprentissage supervisé où l'objectif est de transférer des connaissances acquises à partir de données d'un domaine pour apprendre des données d'un autre domaine qui ne pouvaient pas faire l'objet d'un apprentissage seules car trop peu nombreuses.

L'apprentissage non supervisé est un apprentissage où aucune étiquette n'est donnée et où l'algorithme apprend à partir des seuls signaux d'entrées. C'est un apprentissage où l'on cherche à regrouper les données selon leur ressemblance dans l'espace d'entrée (aussi appelé clustering). Il se prête particulièrement bien à des problèmes de clustering où on ne sait pas à quels clusters appartiennent les données.

L'apprentissage par renforcement présente un paradigme d'apprentissage où la méthode d'apprentissage interagit avec un environnement en fonction de son but et reçoit de l'environnement un retour positif ou négatif lui permettant d'évaluer et d'apprendre de ses actions. Un exemple célèbre est AlphaGo Zero [Silver et al., 2017] qui apprît à jouer au Go en jouant contre lui même avec pour seul retour le vainqueur de la partie.

Lors d'un apprentissage statistique d'une méthode, la définition et le contrôle du critère d'apprentissage sont des éléments très importants.

1.2.3 Critère d'apprentissage

Dans un problème d'optimisation d'un apprentissage automatique, l'objectif est de trouver les meilleurs paramètres de l'algorithme pour résoudre le problème à l'aide des données disponibles. Le critère d'apprentissage permet à la fois d'optimiser les paramètres en donnant un retour sur la mesure de l'erreur obtenue et d'évaluer la capacité de l'algorithme à résoudre la tâche. Une technique courante en optimisation consiste à faire évoluer les paramètres dans le sens opposé au gradient de l'erreur en utilisant une technique comme la descente de gradient. Les données disponibles permettent d'inférer les meilleurs paramètres en fonction du critère pour résoudre le problème sur ces données. Cependant, lors d'un apprentissage on cherche à généraliser et pouvoir résoudre le problèmes sur des données inconnues. Le fait d'apprendre les données disponibles par cœur sans généraliser s'appelle le sur-apprentissage.

Il est donc crucial d'éviter le sur-apprentissage. Pour détecter le sur-apprentissage d'une méthode d'apprentissage automatique, une base de validation est fréquemment utilisée afin d'évaluer le critère sur des exemples non vus. On mesure le critère sur l'ensemble des données de validation et lorsque celui-ci augmente il faut arrêter l'apprentissage pour ne pas sur-apprendre comme montré sur la figure 1.1.

Il existe des méthodes permettant de diminuer l'effet de sur-apprentissage des données,

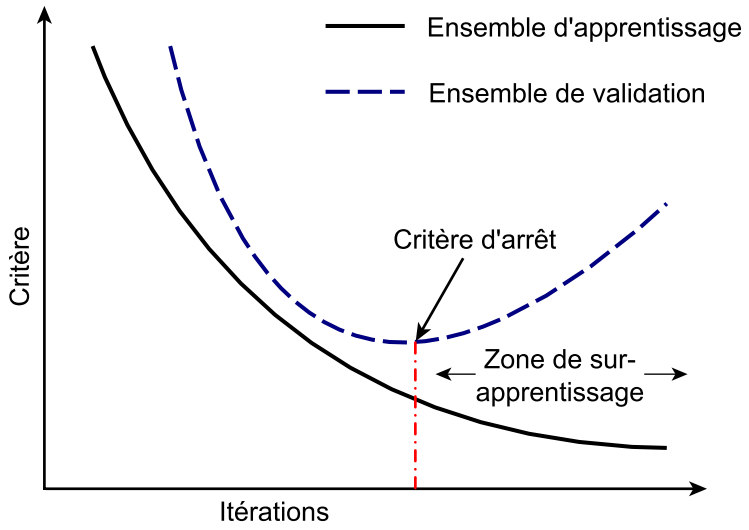


FIGURE 1.1 – Courbes d'apprentissage et de validation d'une méthode d'apprentissage. Des pointillés rouges marquent le point d'arrêt de l'apprentissage pour ne pas sur-apprendre.

parmi lesquelles l'application d'une régularisation qui est un processus d'introduction d'informations supplémentaires. Un terme de régularisation $R(f)$ contrôlé par un paramètre λ est alors ajouté à la fonction de coût V où f représente la méthode ayant les paramètres θ et x_i des exemples :

$$F = min_f \sum_{i=1}^{n} V(f(x_i), y_i) + \lambda R(f) \tag{1.1}$$

Ces informations ont pour effet de lisser les paramètres et d'éviter un apprentissage par cœur. Les régularisations les plus connues sont les régularisations $L1$ et $L2$ [Ng, 2004] qui consistent à pénaliser les valeurs extrêmes (forme de lissage) des paramètres à optimiser par l'application d'une norme sur ceux-ci. Pour L1 on a alors :

$$R(f) = |\theta| \tag{1.2}$$

et pour L2 :

$$R(f) = \theta^2 \tag{1.3}$$

Nous avons vu les bases de l'apprentissage automatique à travers les méthodes, les types et l'importance du critère d'apprentissage. Nous nous intéressons maintenant aux méthodes d'apprentissage supervisé de réseaux de neurones.

1.3 Perceptron multi-couches(MLP)

Le premier réseau de neurones artificiel proposé dans la littérature fut le perceptron de Rosenblatt [Rosenblatt, 1958] qui ne comporte qu'une seule couche. Son principe a été repris et étendu à plusieurs couches pour constituer le perceptron multi-couches (en anglais "Multi Layer Perceptron" : MLP) [Rumelhart et al., 1985]. Le MLP a été pendant longtemps l'architecture privilégiée en raison de sa capacité lui permettant d'approximer n'importe quelle fonction continue d'après le théorème d'approximation universelle [Cybenko, 1989]. Les domaines d'application du MLP sont variés et recouvrent les problèmes de reconnaissance de formes au sens large. Ils ont notamment permis des gains de performances importants en reconnaissance de la parole, de l'écriture ou encore en traitement automatique du langage naturel. Ils sont aujourd'hui les modèles statistiques les plus répandus en apprentissage automatique.

1.3.1 Principe

Les réseaux de neurones sont construits en couches. Chaque couche est composée de neurones et chaque neurone possède des poids d'entrée. Un MLP est représenté figure 1.2, les neurones sont représentés par des nœuds et les poids par des arcs. Un réseau de neurones est toujours composé au minimum d'une couche d'entrée puis d'une ou plusieurs couche(s) de neurones appelée(s) couche(s) cachée(s) et enfin d'une couche de sortie. Il n'y a pas d'architecture type d'un réseau de neurones en termes de nombres de couches et de neurones car celle-ci dépend de la tâche à résoudre, des données disponibles et des contraintes d'utilisation (espace mémoire, temps). L'architecture d'un réseau de neurones est un méta-paramètre.

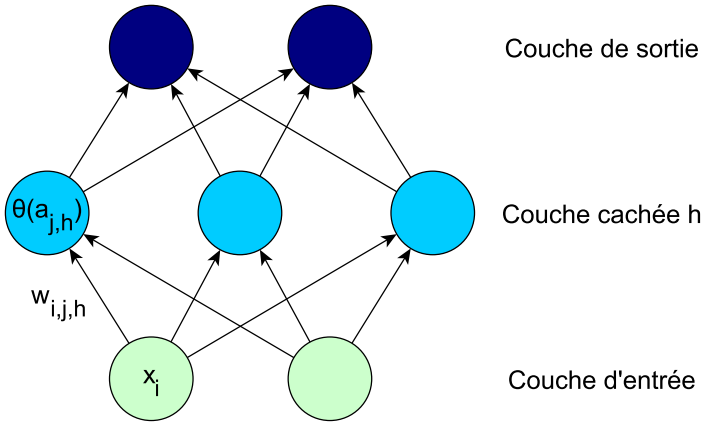


FIGURE 1.2 – Perceptron multi-couches

La couche d'entrée n'est pas une couche apprise et ne possède donc pas de neurones à proprement parler. Les valeurs de la couche d'entrée sont directement celles du vecteur de données d'entrée X et c'est pourquoi leur nombre est toujours égal au nombre d'éléments du vecteur d'entrée. Il est donc nécessaire pour utiliser un MLP d'avoir des vecteurs

d'entrée de taille fixe. La couche d'entrée est toujours la première couche ou la couche la plus basse d'un réseau de neurones ($h = 0$) et elle est suivie de couches cachées (h).

Les couches cachées sont les couches contenant des neurones ayant des poids. Les poids des neurones des couches cachées sont les paramètres d'un réseau de neurones qu'il faut optimiser par l'apprentissage. Les valeurs de sortie ou valeurs d'activation des neurones sont calculées par produit scalaire des vecteurs de poids de chaque neurone et du vecteur d'entrée qui est le vecteur des valeurs de sortie des neurones de la couche cachée précédente ($h - 1$). Un neurone possède un biais qui est un poids dont l'entrée vaut toujours 1 et qui permet de modifier le seuil d'activation du neurone. Avant de définir l'opération élémentaire d'un neurone d'une couche cachée, nous définissons les termes qui vont suivre. $w_{i,j,h}$ est le poids entre le neurone j de la couche cachée h et le neurone i de la couche précédente $h-1$. I et J sont respectivement le nombre de neurones de la couche $h-1$ et h. x_i est la valeur du neurone i de la couche précédente. $\beta_{j,h}$ est le biais du neurone j de la couche cachée h.

Le calcul de la valeur d'activation $a_{j,h}$ du j$^{\text{ème}}$ neurone de la couche cachée est présenté dans l'équation 1.4.

$$a_{j,h} = \sum_{i=1}^{I} (w_{i,j,h} x_i) + \beta_{j,h} \tag{1.4}$$

L'étape suivante consiste à calculer (1.5) la fonction d'activation des neurones.

$$b_{j,h} = \theta_h(a_{j,h}) \tag{1.5}$$

La fonction θ_h est généralement une tangente hyperbolique :

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{1.6}$$

ou une sigmoïde logistique :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.7}$$

La non-linéarité de ces deux fonctions permet d'apprendre des réseaux capables d'établir des frontières non linéaires de classification. Ces fonctions sont également dérivables et permettent le calcul d'un gradient.

La dernière couche d'un réseau de neurones est la couche de sortie. Ces neurones représentent les valeurs attendues afin de résoudre la tâche comme par exemple des probabilités de classes pour une tâche de classification ou des valeurs réelles dans le cadre d'une régression. Les neurones de la couche de sortie se calculent de la même manière que ceux d'une couche cachée sauf qu'on n'y applique généralement pas la même fonction d'activation. Celle-ci dépend de la tâche que l'on cherche à réaliser : classification ou régression.

Pour une tâche de classification, une fonction d'activation softmax est généralement utilisée. Le nombre de neurones de sorties est égal aux nombre de classes. L'équation de la fonction softmax est définie par l'équation 1.8 où C_k est la k$^{\text{ième}}$ classe, y_k la probabilité

de sortie de la k$^{\text{ième}}$ classe et a_k l'activation du k$^{\text{ième}}$ neurone.

$$p(C_k|X) = y_k = \frac{e^{a_k}}{\sum_{k=1}^{K} e^{a'_k}} \tag{1.8}$$

Dans le cadre d'une tâche de régression, la sortie correspond aux valeurs à prédire, comme par exemple des coordonnées dans une image. Il n'est alors pas nécessaire d'utiliser une fonction d'activation. Pour la régression, le nombre de sorties est égal au nombre de valeurs à estimer.

1.3.2 Apprentissage supervisé d'un réseau de neurones

Pour l'apprentissage supervisé d'un réseau de neurones, la difficulté est de trouver les bons poids permettant de résoudre la tâche. Il faut donc apprendre les poids du réseau de telle sorte que la sortie Y d'un réseau auquel on a présenté la donnée X donne la sortie Z attendue ou vérité terrain. C'est un problème d'optimisation où l'on cherche à faire correspondre au mieux la sortie Y à la sortie attendue Z. Par exemple, pour un problème de classification d'images de chiffres, on utiliserait un réseau à 10 sorties représentant les 10 chiffres. Pour apprendre ce réseau il faut des images de chiffres (X) et leur étiquette associée (Z) qui est un chiffre de 0 à 9. Z est alors un vecteur de taille 10 dont toutes les valeurs sont à 0 sauf celle du chiffre attendu qui est à 1.

La fonction de coût à optimiser est choisie pour rendre compte de l'aptitude du réseau à prendre les bonnes décisions.

1.3.2.1 Critère d'apprentissage d'un réseau de neurones

L'apprentissage supervisé d'un réseau de neurones se fait par l'optimisation des poids du réseaux en fonction d'un critère sur les données d'apprentissage X. Le critère d'apprentissage est évalué à l'aide d'une fonction de coût qui calcule la différence entre la sortie du réseau Y et la sortie attendue Z. La fonction de coût dépend donc de la tâche à résoudre. Il existe différentes fonctions de coût, généralement représentées par les notations $E(Y, Z)$ ou $\mathcal{L}(X, Z)$. Les plus connues sont l'erreur quadratique et l'entropie croisée. Comme pour les fonctions d'activation, la fonction de coût doit être dérivable car on utilise ses dérivées partielles pour mettre à jour les poids. Ce sont ces dérivées de l'erreur qui sont rétropropagées à travers les couches.

La fonction de coût quadratique (équation 1.9) est généralement privilégié pour des problèmes de régression ou de classification :

$$E(Y, Z) = \frac{1}{2}\|Y - Z\|^2 \tag{1.9}$$

L'entropie croisée est généralement utilisée pour la classification et s'exprime pour un problèmes à K classes par l'équation 1.10 où z_k est le k$^{\text{ième}}$ élément de Z et y_k le k$^{\text{ième}}$ de Y .

$$\mathcal{L}(X, Z) = -\sum_{k=1}^{K} z_k \ln y_k \tag{1.10}$$

Nous allons maintenant voir comment rétropropager l'erreur en calculant le gradient à partir des dérivées partielles.

1.3.2.2 Rétropropagation du gradient de l'erreur

La rétropropagation du gradient, trouvée indépendamment par [Rumelhart et al., 1985] et [LeCun, 1985], est une méthode fondée sur la descente de gradient. La descente de gradient est une méthode d'optimisation itérative permettant de trouver un minimum local d'une fonction. Cette méthode calcule le gradient (direction de la pente) en un point de la fonction à optimiser et modifie les paramètres θ de la fonction de façon à diminuer l'erreur. La modification des poids doit s'opposer à l'augmentation de l'erreur, elle est donc de sens opposé au gradient. Puis, on recommence à partir du nouveau point jusqu'à atteindre le minimum local. Pour une fonction E paramétrée par θ, la descente de gradient ayant un pas de descente λ est donnée par l'équation 1.11 :

$$\theta = \theta - \lambda \frac{\partial E}{\partial \theta} \tag{1.11}$$

Dans le cadre de l'apprentissage supervisé d'un réseau de neurones, seule l'évaluation du critère de la couche de sortie est possible. Pour un réseau de neurones, le problème est d'évaluer le critère d'optimisation pour les couches cachées. La descente de gradient à travers les couches d'un réseau de neurones est possible grâce à la règle de dérivation en chaîne. La règle de dérivation en chaîne permet de transmettre le coût à travers les couches cachées. La fonction de coût $E(y, z)$ étant dérivable, on en déduit la dérivée partielle de l'erreur E par rapport à la sortie y :

$$\frac{\partial E}{\partial y} \tag{1.12}$$

Grâce à la règle de dérivation en chaîne, le calcul de la dérivée de l'erreur par rapport aux poids de la couche de sortie n s'exprime alors par l'équation 1.13 :

$$\frac{\partial E}{\partial w_{i,n}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_{i,n}} \tag{1.13}$$

Il n'est pas possible de calculer directement $\frac{\partial E}{\partial w_{i,n-1}}$, la couche $h = n - 1$ n'étant pas directement reliée à la sortie. Il faut alors appliquer de manière récursive la règle de dérivation en chaîne afin de rétropropager l'erreur pouvant alors être rétropropagée jusqu'à la couche d'entrée :

$$\frac{\partial E}{\partial w_{i,0}} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_{i,n}} \frac{\partial w_{i,n}}{\partial w_{i,n-1}} \cdots \frac{\partial w_{i,1}}{\partial w_{i,0}} \tag{1.14}$$

Grâce à la règle de dérivation en chaîne, il est donc possible de transmettre le gradient de l'erreur aux poids de la couche de sortie, puis de le transférer aux couches cachées. C'est ce processus que l'on appelle la rétropropagation du gradient. Le gradient de l'erreur est donc diffusé à partir de la couche de sortie à travers toutes les couches cachées grâce à la règle de dérivation en chaîne. Cette règle permet de calculer le gradient à soustraire

à un poids afin de le mettre à jour. Le calcul du gradient des poids $w_{i,h}$ du neurone i de la couche h s'exprime alors comme suit :

$$\delta_{i,h} = \frac{\partial E}{\partial w_{i,h}} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial w_{i,h}} \tag{1.15}$$

On itère alors de la couche de sortie $h = n$ jusqu'à la couche d'entrée $h = 0$ en suivant la règle de dérivation en chaîne afin de pouvoir rétropropager l'erreur sur l'ensemble du réseau. La rétropropagation du gradient ne peut donc se faire que séquentiellement en partant de la couche de sortie jusqu'à la couche d'entrée.

La mise à jour du gradient dans la rétropropagation est contrôlée par un taux d'apprentissage λ qui régule la descente de gradient. Le taux d'apprentissage est un pas de descente qui pourrait être calculé exactement par un algorithme d'ordre 2 (méthode de Newton). Cependant, c'est rarement le cas en pratique car le coût est en $O(n^4)$ contre $O(n^2)$ pour une rétropropagation sans calcul du pas de descente à chaque exemple, où n est le nombre de paramètres du réseau. La mise à jour d'un poids s'exprime alors en fonction du taux λ par :

$$w_{i,h} = w_{i,h} - \lambda\frac{\partial E}{\partial w_{i,h}} = w_{i,h} - \lambda\delta_{i,h} \tag{1.16}$$

Plus ce taux est grand, plus la descente est rapide ; plus il est petit, plus la descente est lente. Un taux d'apprentissage trop élevé ne permet généralement pas de converger vers l'optimum global. La direction du gradient induira un changement régulier de pentes de descente permettant de sortir des minimums locaux mais ne permettant pas d'y descendre. Un taux d'apprentissage trop faible conduit en général à une convergence lente et qui peut être bloquée dans un minimum local.

On peut stopper la descente de gradient de plusieurs manières. Une condition d'arrêt facilement évaluable est de déterminer un nombre d'itérations, c'est-à-dire le nombre de fois où toutes les données d'apprentissage ont été propagées en avant et rétropropagées. Comme vu dans la section 1.2, l'utilisation d'un critère peut conduire à un sur-apprentissage des données. Pour contrôler ce sur-apprentissage une des techniques est d'utiliser une base de validation qui permet d'évaluer la valeur du critère pour des données inconnues. Une condition d'arrêt fréquemment utilisée consiste à arrêter lorsque la valeur du critère sur la validation augmente ou n'évolue plus.

Afin d'apprendre un réseau de neurones, il faut appliquer la descente de gradient pour tous les exemples X et renouveler l'opération jusqu'à atteindre la condition d'arrêt. La descente de gradient peut nécessiter un grand nombre de calculs qui croit exponentiellement avec le nombre de neurones lorsqu'on calcule le pas de descente de gradient (complexité en $O(n^4)$). Il faut donc utiliser une méthode dont les temps de calculs permettent un apprentissage dans un temps réaliste. Pour y parvenir, l'algorithme de descente de gradient le plus efficace et rapide est la descente de gradient stochastique [Bottou, 1998, Bottou, 2012]. Nous détaillons cet algorithme ci-après dans l'algorithme 1. La descente de gradient utilise un pas fixe non optimal permettant des calculs de gradients plus rapides en $O(n^2)$ et mélange les données ce qui permet une bonne convergence du réseau vers l'optimum.

Algorithme 1: Algorithme de descente de gradient stochastique.

Input : network, X, Z, λ
Output: network
// Initialisation du réseau
network.initializeWeights() ;
// Boucle d'apprentissage
while *network has not converged* **do**
  $X_{shuffled}$,$Z_{shuffled}$ = shuffle(X,Z);
  foreach x, z ***in*** $X_{shuffled}$,$Z_{shuffled}$ **do**
    $y \leftarrow$ network.compute(x) ;
    $e \leftarrow E(y, z)$;
    foreach $w_{i,j}$ ***in*** *network* **do**
      $w_{i,j} = w_{i,j} - \lambda \frac{\partial e}{\partial w_{i,j}}$;
  testingNetworkConvergence();
return network

L'un des inconvénients les plus importants de la SGD est le taux d'apprentissage fixe. En effet, s'il est trop petit l'apprentissage est piégé dans un minimum local et s'il est trop grand l'apprentissage ne convergera pas. Généralement, il est intéressant de commencer par un taux d'apprentissage plus grand pour finir par un taux plus petit. Il existe des améliorations ou variantes à la descente du gradient stochastique.

Une technique consiste à introduire un moment (momentum) [Qian, 1999] dans le calcul du gradient lors de la descente. Ce moment garde l'inertie en ajoutant les informations des changements précédents lors de la descente de gradient, c'est-à-dire qu'à l'exemple t, on prend en compte le gradient de l'exemple précédent $t-1$. Cette méthode permet de sortir des minimums locaux et d'accélérer l'optimisation du réseau. La mise à jour du gradient avec un moment η se traduit par l'équation 1.17.

$$w_{i,h}^{t} = w_{i,h}^{t-1} - \lambda\left(\frac{\partial E}{\partial w_{i,h}^{t}} + \eta \frac{\partial E}{\partial w_{i,h}^{t-1}}\right) \tag{1.17}$$

Une autre méthode AdaGrad [Duchi et al., 2011] utilise un algorithme de gradient adaptatif. Cette méthode permet de diminuer le taux d'apprentissage des poids qui ont souvent été mis à jour et d'augmenter celui de ceux qui ne l'ont pas été. Pour ce faire, le carré des gradients G est conservé et on obtient l'équation de G :

$$G_{i,h}^{t} = \sum_{\tau=1}^{t} (\delta_{i,h}^{\tau})^2 \tag{1.18}$$

Et la misr à jour des poids se fait alors comme suit :

$$w_{i,h}^{t+1} = w_{i,h}^{t} - \frac{\lambda}{\sqrt{G_{i,h}^{t}}} \frac{\partial E}{\partial w_{i,h}^{t}} \tag{1.19}$$

AdaDelta [Zeiler, 2012] et RMSProp [Tieleman and Hinton, 2012] sont également des méthodes permettant d'adapter le gradient en fonction de chaque paramètre. RMSProp est identique à AdaDelta mais avec $\gamma = 0.9$. L'idée est de diviser le taux d'apprentissage par une moyenne E des magnitudes des valeurs de gradients récentes d'équation :

$$E_{i,h}^t = \gamma E_{i,h}^{t-1} + (1-\gamma)(\delta_{i,h}^t)^2 \tag{1.20}$$

où γ est un facteur d'oubli. La mis à jour des poids se fait alors suivant l'équation :

$$w_{i,h}^{t+1} = w_{i,h}^t - \frac{\lambda}{\sqrt{E_{i,h}^t}} \frac{\partial E}{\partial w_{i,h}^t} \tag{1.21}$$

Il existe d'autres méthodes comme Adam [Kingma and Ba, 2014] qui fonctionnent comme AdaDelta mais conservent en plus un moment composé de l'historique du carré des gradients. Aucune de ces méthodes n'a montré de supériorité absolue ; le choix de la méthode dépend du problème et des données.

En utilisant ces techniques de descente de gradient, comme pour toutes les méthodes d'apprentissage automatique, il faut prêter attention aux problèmes de sur-apprentissage.

1.3.2.3 Sur-apprentissage d'un réseau de neurones

Comme vu précédemment, l'objectif est de trouver une solution générale capable de résoudre le problème pour des données que le système n'a jamais vu. Dans le cas d'un réseau de neurones la taille du réseau a de l'importance car un réseau contenant trop de neurones au regard du nombre d'exemples aura tendance à sur-apprendre. Pour contrôler le sur-apprentissage nous avons vu qu'une des bonnes pratiques est d'utiliser une base de données de validation différente de la base d'apprentissage. Pour un réseau de neurones, on vérifie alors l'erreur de validation à chaque itération. Nous avons vu dans la sous-section 1.2.3 que l'application d'une régularisation $L1$ ou $L2$ [Ng, 2004] sur la fonction de coût permettait de diminuer le sur-apprentissage.

Il existe d'autres méthodes comme l'augmentation du nombre de données d'entrée avec du bruit ou des déformations qui permet d'insérer un grand nombre d'exemples supplémentaires qu'il est plus difficile de sur-apprendre. La normalisation des données d'entrée (en anglais "batch normalization") permet aussi de diminuer le sur-apprentissage en réduisant la variation de la valeur des neurones ce qui a un petit effet de régularisation. La normalisation par batch permet également d'améliorer la vitesse de convergence. Le dropout [Srivastava et al., 2014, Zaremba et al., 2014] est une méthode plus récente et spécifique aux réseaux de neurones qui permet également de diminuer le sur-apprentissage. Elle consiste à déterminer au hasard un sous-ensemble de neurones ou poids ou entrées qui seront ignorés. Le dropout permet d'éviter d'apprendre une inter-dépendance entre les neurones qui pourrait être efficace pour les données d'apprentissage mais pas pour de nouvelles données. Ceci devient possible avec le dropout puisqu'un neurone n'est pas systématiquement présent dans l'apprentissage de tous les exemples, donnant ainsi d'excellents résultats au prix d'un temps d'apprentissage plus long. De manière générale, l'utilisation

d'une normalisation par batch, de l'augmentation des données et du dropout fonctionne très bien ensemble et permettent d'apprendre plus rapidement sans sur-apprendre.

L'algorithme de rétropropagation du gradient pose un autre problème. Une de ses limitations tient au fait qu'il ne permet pas d'apprendre une architecture avec un trop grand nombre de couches. Or, de telles architectures peuvent en principe résoudre des tâches plus complexes. Ce problème est lié au problème de la disparition du gradient [Hochreiter, 1998] : plus le réseau a de couches, plus le gradient est faible sur les couches proches de la couche d'entrée. Nous voyons plus en détails dans la section suivante les architectures profondes, le problème de la disparition du gradient et les manières d'apprendre de telles architectures.

1.4 Architectures profondes

Les architectures profondes sont des architectures de réseaux de neurones où un grand nombre de couches sont empilées pour réaliser des fonctions de prédiction plus complexes. On définit les couches basses comme les couches les plus proches de la couche d'entrée, et les couches hautes comme les plus proches de la couche de sortie.

1.4.1 Réseaux de neurones profonds

Les réseaux de neurones profonds sont composés des mêmes éléments qu'un MLP, c'est-à-dire des couches de neurones. Ils présentent cependant un plus grand nombre de couches cachées. L'objectif d'un réseau profond est de remplacer l'extraction de caractéristiques par sa propre représentation des données d'entrée. Pour y parvenir, il est nécessaire d'avoir des niveaux d'abstractions supplémentaires et donc d'ajouter des couches. En effet, chaque couche ajoutée réalise une transformation non-linéaire des données de sortie de la couche précédente. Un réseau profond apprend alors dans ses couches basses une représentation peu abstraite et directe des données (peu d'applications de non linéarités). Plus on évolue vers les couches hautes, plus cette représentation est abstraite [Zeiler and Fergus, 2014], permettant ainsi de prendre une décision plus sûre. Le schéma d'un réseau profond est donné figure 1.3.

L'apprentissage de tels réseaux avec un grand nombre de couches s'avère difficile en raison du problème de la disparition du gradient, que nous évoquons dans le paragraphe suivant.

1.4.1.1 Disparition du gradient

La disparition du gradient [Bengio et al., 1994, Hochreiter, 1998] est un phénomène observé lors de l'apprentissage basé sur la rétropropagation du gradient. Il apparaît particulièrement pour les réseaux profonds et les réseaux récurrents. Le problème de la disparition du gradient provient de deux phénomènes : la règle de dérivation en chaîne d'une part et l'utilisation de fonctions d'activation telles que la tangente hyperbolique et la sigmoïde d'autre part. La dérivée de la tangente hyperbolique est comprise entre -1 et 1, et celle de

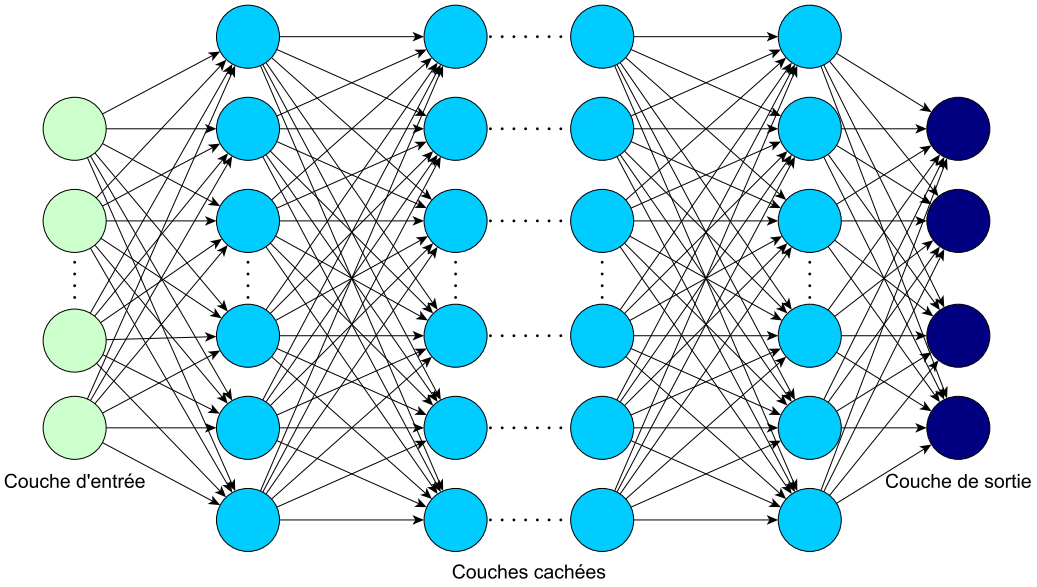


FIGURE 1.3 – Réseau profond à n couches cachées.

la fonction sigmoïde entre 0 et 0.25. En appliquant la règle de dérivation en chaîne donnée par l'équation 1.14 à un réseau contenant n couches, on multiplie la dérivée ou gradient n fois. Les dérivées étant inférieures à 1, le gradient décroît exponentiellement avec n vers 0 notamment sur les couches du réseau. Inversement, pour des fonctions d'activation dont la dérivée peut être supérieure à 1, on peut rencontrer un phénomène d'explosion du gradient. L'explosion du gradient est le phénomène inverse de la disparition du gradient et nuit autant à l'apprentissage. Cependant, il est peu présent car les fonctions d'activation principalement utilisées ont une dérivée inférieure à 1. Afin de contourner le problème de disparition du gradient et d'apprendre un réseau profond, il existe différentes solutions.

1.4.1.2 Apprentissage profond

Une première solution consiste à utiliser la normalisation de gradient [Bengio et al., 1994] (gradient clipping en anglais). Cette méthode permet de modifier la valeur du gradient lorsque sa norme est inférieure (disparition) ou supérieure (explosion du gradient) à un seuil donné. Elle permet ainsi d'éviter qu'un gradient soit trop petit ou trop grand. L'inconvénient de cette méthode est l'introduction d'un nouveau paramètre d'apprentissage qui est le seuil de normalisation.

Une des méthodes les plus célèbres consiste à pré-apprendre un réseau couche par couche de manière non supervisée. Cette méthode a été popularisé par G. Hinton [Hinton et al., 2006] afin d'apprendre rapidement des DBN ("Deep Beliefs Networks") à l'aide d'une machine de Boltzman restreinte (RBM). Un DBN est composé de multiples RBMs et chaque RBM peut être considérée comme un auto-encodeur. Chaque RBM apprend, couche par couche, à reconstruire ses entrées de manière non supervisée. Ainsi, il est appris à chaque couche du réseau une représentation des entrées permettant de détecter des caractéristiques. Lorsque l'ensemble des couches sont pré-apprises, le réseau est lui

même appris globalement par rétropropagation supervisée afin d'effectuer un réglage fin.

Une méthode récente permet de résoudre le problème de disparition du gradient avec des réseaux de neurones résiduels [He et al., 2016]. Le principe des réseaux résiduels est de découper un réseau très profond en blocs et de passer les sorties de couches plus basses en même temps que les entrées. L'intérêt de ce processus est de permettre de remonter l'information d'une couche basse qui aurait été diluée à travers les couches. Toutefois, cette méthode est davantage un contournement du problème dans la mesure où on pourrait le voir comme un ensemble de sous réseaux imbriqués les uns aux autres.

Le problème de disparition du gradient provient des fonctions d'activation tangente hyperbolique et sigmoïde. Il faudrait donc une fonction d'activation dont la dérivée ne conduirait pas à une décroissance exponentielle du gradient. Une nouvelle fonction d'activation la "Rectified Linear Unit" ReLU [Nair and Hinton, 2010] a été proposée pour résoudre ce problème. L'équation de la fonction ReLU est : $ReLU(x) = max(0, x)$. Dans ce cas, la dérivée est soit nulle lorsque x est négatif, soit égale à 1 lorsque x est positif. Cette fonction permet de limiter fortement la disparition du gradient. Les activations ReLU sont la solution privilégiée pour l'apprentissage de réseaux profonds.

Le problème de disparition du gradient est aussi présent dans l'apprentissage des réseaux de neurones récurrents. Nous nous intéressons maintenant à ces réseaux qui ont une mémoire et permettent de réaliser des tâches de prédiction séquentielle.

1.5 Réseaux de neurones récurrents (RNN)

Les réseaux profonds et MLP ne sont pas adaptés au traitement des problèmes séquentiels. En effet, ils n'ont pas de mémoire et ne prennent pas en compte de contexte, qu'il soit temporel ou spatial. De plus, ils travaillent sur des données de taille fixe ce qui est difficilement compatible avec un fonctionnement où les données séquentielles n'ont pas de longueur fixe. Les problèmes séquentiels sont nombreux et afin de les appréhender il faut utiliser des modèles dynamiques qui prennent en compte les états précédents d'une séquence comme le font les modèles de Markov cachés [Rabiner, 1989] ou les champs aléatoires conditionnels [Lafferty et al., 2001].

Les réseaux de neurones récurrents (RNN pour "Recurrent Neural Networks") ont été créés pour résoudre des problèmes séquentiels. Le principe de RNN a été introduit pour la première fois dans le réseau de Hopfield [Hopfield, 1982]. Ce principe a été ensuite repris dans la machine de Boltzmann [Ackley et al., 1985] et le réseau d'Elman [Elman, 1990]. Par la suite, les RNN se sont formalisés et améliorés avec le RNN bidirectionnel (BRNN) [Schuster and Paliwal, 1997]. D'autres réseaux similaires aux RNN sont apparus également : Echo state network [Jaeger, 2001]. Au final, ce sont les cellules de mémoire à long et court terme (LSTM)[Hochreiter and Schmidhuber, 1997], en lieu et place des cellules classiques, qui ont permis l'essor des RNN avec en 2005 les réseaux bidirectionnels à cellules LSTM (BLSTM) [Graves and Schmidhuber, 2005]. Ces réseaux ont pu être appris grâce à une nouvelle méthode d'apprentissage (Classification temporelle connexionniste (CTC)).

La force des réseaux récurrents réside dans leur capacité de mémoire pour le traitement de séquences. Ils sont donc naturellement utilisés pour des problèmes tels que

la reconnaissance de la parole [Graves and Schmidhuber, 2005] ou la reconnaissance de textes manuscrits [Graves and Schmidhuber, 2009]. Autour de la problématique de la reconnaissance de l'écrit, il existe des applications pour la détection de mots (word spotting) [Frinken et al., 2012] mais également pour la génération de texte manuscrit [Graves, 2013a]. Il existe également d'autres applications hors du domaine du traitement de la langue naturelle comme la reconnaissance d'émotions [Wöllmer et al., 2010], la segmentation ou classification des images [Graves, 2012], la prédiction de séquence de protéines [Thireou and Reczko, 2007]. Récemment, les RNN ont été utilisés dans le cadre de la génération de langage pour des applications de génération de la légende d'image (image captioning) [Xu et al., 2015, Wu et al., 2017, Anderson et al., 2017] par exemple.

1.5.1 Principe

Les réseaux de neurones récurrents ont pour particularité de ne pas prendre un vecteur en entrée mais une séquence de vecteurs. Cette séquence de vecteurs est généralement extraite à l'aide d'une fenêtre glissante parcourant le signal à analyser. On appelle les vecteurs ou éléments de cette séquence : trames. Comme les réseaux profonds et les MLP, un RNN est composé de couches qui contiennent des neurones appelées aussi cellules. Sur la figure 1.4, est mise en évidence la différence entre un RNN et un MLP : les neurones récurrents possèdent des poids supplémentaires (en rouge) reliant leurs sorties à leurs entrées et permettant de modéliser la mémoire pour le traitement de séquences. La sortie d'une couche récurrente à un instant donné dépend donc de sa sortie précédente. Elle a pour entrée la sortie de la couche précédente à l'instant courant t et sa propre sortie à l'instant précédent $t-1$.

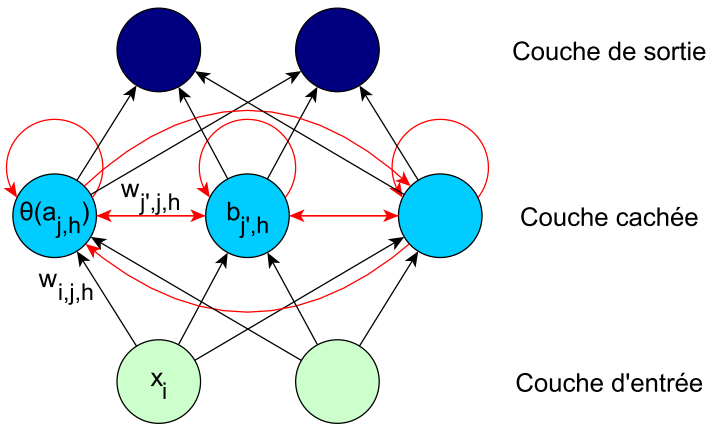


FIGURE 1.4 – Réseau de neurones récurrents

L'équation 1.22 d'un neurone récurrent est identique à celles d'un MLP pour les poids reliant deux couches. Les poids de récurrence $w_{j',j,h}$ ajoutent un terme (en rouge) à l'équation par rapport à celle d'un MLP. La formulation des entrés $X = x_0...x_i...x_I$ et des sorties d'un neurone présente en plus un exposant t précisant la position (temporelle) dans la séquence $X^0...X^t...X^T$. Dans l'équation 1.22, $b_{j',h}^{t-1}$ fait référence à la valeur du neurone

j' calculée précédemment pour les entrées x_i^{t-1} de l'élément de séquence précédent. Pour réaliser la passe en avant du premier élément d'une séquence X^0, il n'y pas de valeur en mémoire. Il faut donc initialiser les valeurs de sortie des neurones récurrents à 0.

$$a_{j,h}^t = \sum_{i=1}^{I} w_{i,j,h} x_i^t + \sum_{j'=1}^{J} w_{j',j,h} b_{j',h}^{t-1} \tag{1.22}$$

Comme pour un MLP, on applique également une fonction d'activation sigmoïde ou tangente hyperbolique (cf équation 1.5).

1.5.2 Apprentissage

La rétropropagation du gradient ne peut pas s'appliquer aux poids récurrents. C'est pourquoi, d'autres algorithmes d'apprentissage ont été proposés [Jaeger, 2002], tels que le Real-time recurrent learning [Williams and Zipser, 1989] (RTRL) et l'Extended Kalman-filtering [Puskorius and Feldkamp, 1994] (EKF).

RTRL a l'avantage de calculer le gradient exact de l'erreur, comme les méthodes du second ordre. EKF se base sur l'utilisation de filtres de Kalman pour prédire les poids du réseau, mais cela nécessite également des calculs du second ordre. La **Backpropagation Through Time (BPTT)** [Werbos, 1990] est principalement utilisée pour sa facilité d'implémentation et son coût de calcul plus faible $O(n^2)$ contre $O(n^4)$ pour les autres méthodes (où n est le nombre de paramètres).

La BPTT est l'adaptation de la rétropropagation au problème séquentiel. La première étape consiste à déplier le réseau. Le dépliement (figure 1.5) consiste à répliquer le réseau à chaque instant. C'est-à-dire qu'une couche récurrente prend alors en entrée la couche précédente et une copie d'elle même qui a pour entrée les données au temps précédent. Il en résulte que les réseaux récurrents dépliés possèdent un grand nombre de couches cachées et s'apparentent à des réseaux profonds. Le réseau est déplié autant de fois que souhaité. Plus un RNN est déplié et plus sa mémoire est apprise sur le long terme, l'inconvénient est alors le nombre plus élevé de couches. Comme pour un réseau profond, le nombre de couches influe sur le problème de disparition du gradient [Hochreiter, 1998] évoqué précédemment. Une fois le réseau déplié, l'apprentissage est similaire au MLP.

Afin d'apprendre un RNN, il est préférable d'effectuer une initialisation des poids spécifique pour aider la convergence, comme pour les réseaux de neurones profonds. Un bon procédé a été proposé par [Glorot and Bengio, 2010]. Il consiste à initialiser tous les biais à 0 et à tirer les poids au hasard dans l'intervalle donné dans l'équation 1.23 centré en 0 où n_k est le nombre de neurones de la couche k. Cette initialisation est aussi utilisée pour les autres types de réseaux profonds.

$$W_k \in U[-\frac{\sqrt{6}}{\sqrt{n_{k-1}+n_k}}, \frac{\sqrt{6}}{\sqrt{n_{k-1}+n_k}}] \tag{1.23}$$

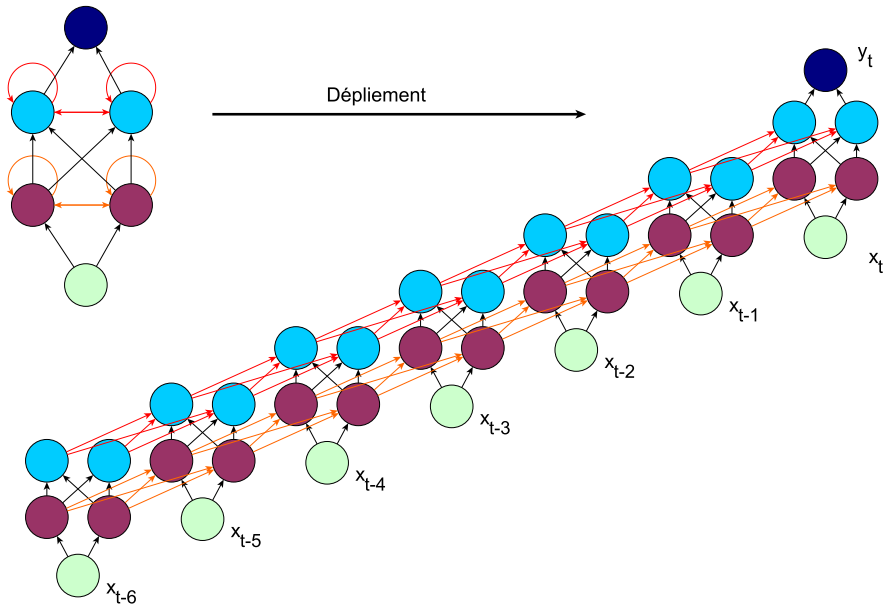


FIGURE 1.5 – Réseau de neurones récurrent déplié.

1.5.3 Réseau de neurones récurrents à mémoire à long et court terme (LSTM)

Les RNN à mémoire à long et court terme (LSTM) ont été introduits avec un double objectif : d'une part, pour résoudre le problème de disparition du gradient lors de l'apprentissage d'un réseau récurrent, d'autre part, pour permettre de garder la mémoire sur le long terme. Les blocs de mémoire LSTM ont d'abord été introduits par Hochreiter [Hochreiter and Schmidhuber, 1997]. Ils possédaient alors 3 neurones internes (un cœur de bloc et deux portes : une d'entrée et une de sortie). La cellule LSTM possède une mémoire interne qui conserve l'état de la cellule avec une fonction d'activation identité. La dérivée est donc égale à 1, ce qui n'affecte pas la rétropropagation de l'erreur. Cette première formulation, bien qu'aidant à résoudre le problème, était perfectible, c'est pourquoi elle fût complétée dans un premier temps avec l'ajout d'une porte d'oubli [Gers et al., 2000], et dans un second temps par l'ajout de connexions internes appelées "peephole" [Gers et al., 2003].

1.5.3.1 Fonctionnement du bloc de mémoire LSTM

Le fonctionnement du bloc LSTM est particulièrement bien décrit par A.Graves dans [Graves, 2012]. Ce document en reprend les formules et s'inspire des schémas. Le bloc de mémoire LSTM schématisé sur la figure 1.6 présente plusieurs éléments essentiels :

- Le cœur du bloc ou cellule de mémoire (Cell en anglais) ;
- Trois portes : entrée, sortie, oubli ;
- Trois cellules multiplicatrices (noires avec une croix) ;
- Des peephole (pointillé orange).

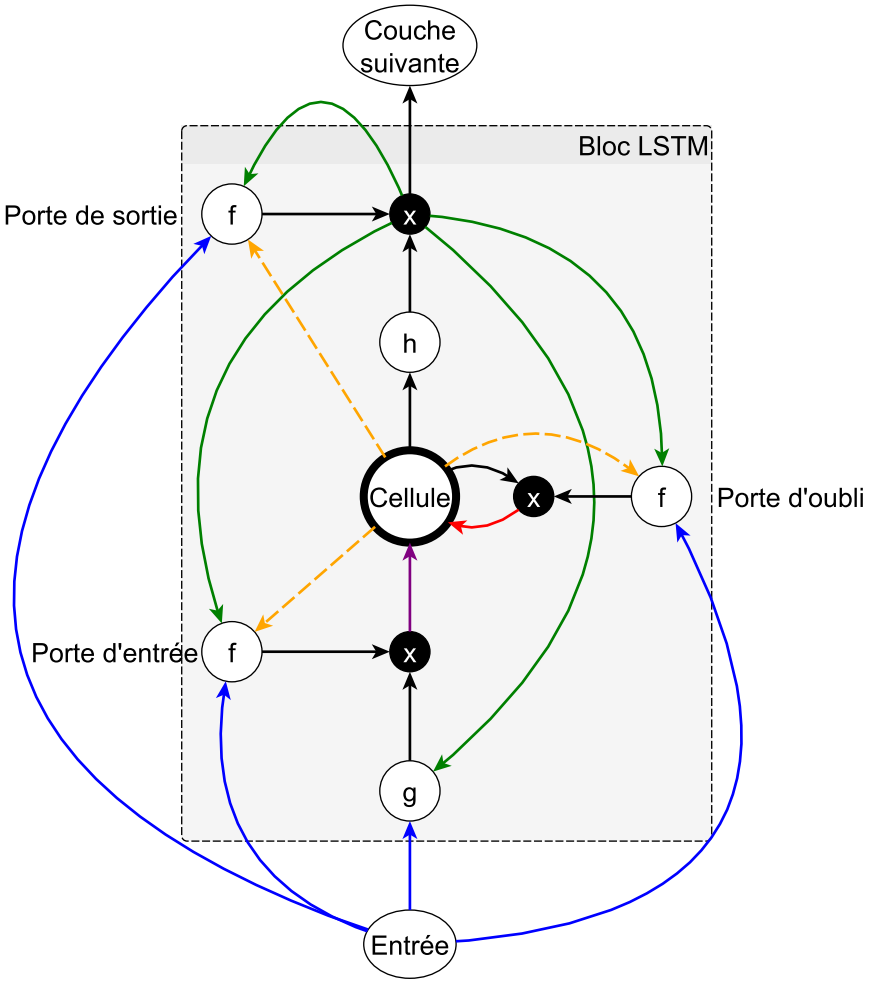


FIGURE 1.6 – Bloc de mémoire LSTM

Il y a I neurones dans la couche précédente, H cellules dans la couche LSTM et C cœur de bloc ($C = 1$ en général). La mise à jour de la mémoire interne est contrôlée par des portes (porte d'entrée et d'oubli). Ainsi, chaque cellule peut maintenir un état sur une durée potentiellement très longue grâce à ce mécanisme qui évite l'effet de disparition au cours du temps. Lorsqu'une nouvelle donnée arrive en entrée du bloc, celle-ci est multipliée par la porte d'entrée prenant des valeurs de la fonction sigmoïde. Des valeurs proches de 1 laissent passer la nouvelle donnée, des valeurs proches de 0 ne la laissent pas passer. La valeur du cœur, quant à elle, peut être réinitialisée par la porte d'oubli grâce au même mécanisme de multiplication/sigmoïde. Enfin, c'est ce même mécanisme qui permet à la porte de sortie de contrôler la sortie du bloc LSTM.

La passe avant pour un bloc LSTM se fait de manière séquentielle. D'abord l'entrée du bloc mémoire (1.24) est calculée. Elle se décompose en 2 parties. On retrouve en bleu la somme des entrées pondérées et en vert la somme des récurrences pondérées. Les b_h^{t-1} sont les valeurs de la couche à l'instant précédent et les x_i^t sont les valeurs d'entrée à l'instant

t.

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{1.24}$$

La valeur de la porte d'entrée est ensuite calculée (1.25),(1.26). Par rapport au calcul de l'entrée, la valeur de la cellule à la position précédente s_c^{t-1} est prise en compte grâce au peephole (en orange).

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{1.25}$$

$$b_\iota^t = f(a_\iota^t) \tag{1.26}$$

La porte d'oubli est calculée de la même manière, avec ses poids(1.27),(1.28).

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{1.27}$$

$$b_\phi^t = f(a_\phi^t) \tag{1.28}$$

L'étape suivante consiste à calculer le cœur du bloc de mémoire (1.29) qui est la somme de deux cellules multiplicatrices. La première (en rouge) est le produit de la valeur de la cellule à l'instant précédent par la porte d'oubli. La seconde (en violet) est la multiplication de l'entrée du bloc LSTM par la porte d'entrée.

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{1.29}$$

C'est seulement après le calcul du cœur du bloc que la porte de sortie est calculée (1.30),(1.31) selon la même méthode que les autres portes. La seule différence porte sur le peephole qui prend en compte l'état courant de la cellule et non celui à l'instant précédent.

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{1.30}$$

$$b_\omega^t = f(a_\omega^t) \tag{1.31}$$

Enfin, la sortie du bloc de mémoire est calculée (1.32) par la multiplication de la valeur de la cellule par la porte de sortie.

$$b_c^t = b_\omega^t h(s_c^t) \tag{1.32}$$

L'apprentissage d'un RNN à cellules LSTM est identique à celui d'un RNN.

Plus récemment, une nouvelle cellule avec des portes a été proposée [Cho et al., 2014] : la cellule GRU pour "gated recurrent unit". Cette cellule possède moins de poids et est donc plus rapide à calculer pour des performances comparables à celles de la cellule LSTM [Chung et al., 2014] lorsque le nombre de paramètres est équivalent.

1.5.3.2 LSTM bidirectionnel (BLSTM)

Le traitement séquentiel des signaux est un problème difficile à aborder, dans la mesure où il est possible de parcourir une séquence de plusieurs manières. En effet, en prenant l'exemple d'une séquence de texte, le parcours peut se faire de la gauche vers la droite ou inversement. La mémoire ne modélise pas les mêmes éléments selon le sens de parcours retenu. Il est donc intéressant de pouvoir modéliser l'information selon différents parcours et ainsi de mettre en œuvre plusieurs réseaux à mémoires LSTM. C'est pour cette raison que les RNN bidirectionnels [Schuster and Paliwal, 1997] ont été introduits et que l'idée a été reprise pour les RNN LSTM [Graves and Schmidhuber, 2005].

La figure 1.7 représente une architecture de BLSTM avec différents types de couches. L'idée est de parcourir la séquence dans les deux sens. Le réseau a alors deux couches pour un même étage au lieu d'une seule. On définit la notion d'étage par un ensemble de couches. Par exemple, il y a 2 couches pour une étage LSTM dans un BLSTM, ayant les mêmes entrées mais étant indépendantes l'une de l'autre. Dans le cadre d'un MLP, une couche est équivalent à un étage. Pour la reconnaissance de texte, ces deux couches correspondent aux deux sens de lecture gauche/droite ou droite/gauche. Pour les réseaux LSTM, on a donc n directions $= n$ couches par niveau. Les deux couches sont calculées indépendamment l'une de l'autre et parcourent la séquence dans le sens gauche/droite pour l'une et droite/gauche pour l'autre. La passe en avant consiste à calculer et stocker les activations pour tous les éléments de la séquence pour chaque étage avant de passer aux calculs de l'étage suivant.

La couche suivant un étage de couches LSTM est généralement une couche de rassemblement (Gather layer en anglais) permettant de réunir les sorties des deux couches d'un élément de la séquence X^t. La couche de rassemblement peut soit concaténer les vecteurs correspondant à un élément de la séquence, soit utiliser une couche cachée pour rassembler les vecteurs des deux directions. Sur la figure 1.7, on peut voir que les sorties du premier étage LSTM sont rassemblés par concaténation dans un vecteur unique représenté par un carré. Le rassemblement par concaténation permet de reconstituer la séquence de vecteurs. Lors de l'utilisation d'une couche cachée pour rassembler, l'objectif est aussi d'effectuer un sous échantillonnage afin de réduire la taille du vecteur caractérisant les données à ce niveau. Cette couche cachée apporte également un niveau d'abstraction supplémentaire. La couche de sortie rassemble les données des deux directions pour prendre sa décision lorsqu'elle est précédée par un étage LSTM comme c'est le cas sur la figure 1.7.

Ces réseaux BLSTM sont peu sensibles au phénomène de disparition du gradient grâce à la cellule LSTM, et permettent de modéliser les contextes gauche et droite d'un élément d'une séquence. Cependant, l'apprentissage d'un réseau récurrent nécessite de connaître précisément la vérité terrain de chaque trame. Cette exigence pose un problème majeur, dans la mesure où il est très onéreux d'étiqueter chaque trame d'une séquence. De plus l'étiquetage peut s'avérer complexe car certaines trames sont difficiles à étiqueter notamment lorsqu'elles sont situées entre 2 caractères.

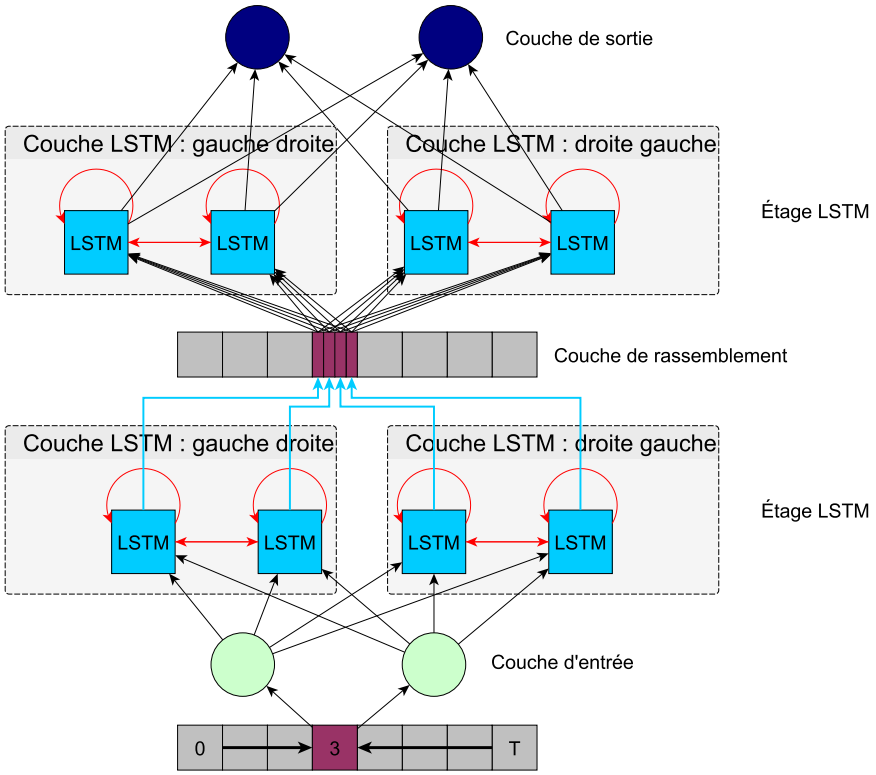


FIGURE 1.7 – BLSTM

1.5.3.3 Classification temporelle connexionniste (CTC)

La classification temporelle connexionniste [Graves and Schmidhuber, 2005] a été introduit dans le but de s'affranchir de l'étiquetage au niveau trame des séquences, pour les réseaux récurrents. Les équations de le CTC ont été redéfini par A. Graves dans [Graves, 2012] et y sont particulièrement claires, nous reprenons ces équations dans ce paragraphe. La CTC est l'une des principales raisons de la popularisation des BLSTM dans le domaine de l'apprentissage. Cette technique permet de calculer une erreur au niveau trame en réalisant une transcription, c'est-à-dire qu'elle résout à la fois le problème de la segmentation et celui de la classification.

Pour résoudre ce problème dual, la proposition originale de la CTC implique la présence d'une couche softmax en sortie avec une étiquette supplémentaire. Cette étiquette supplémentaire correspond à une étiquette blanche ou joker : $-$, c'est-à-dire l'absence d'une étiquette que l'on cherche à prédire. Pour l'exemple de la reconnaissance de l'écriture, le joker correspond à la non présence de caractère de l'alphabet considéré. Le joker permet de stopper la prédiction d'un caractère sans avoir à débuter la prédiction d'un autre caractère. L'ensemble des étiquettes incluant le joker constitue l'ensemble A', et on définit par A'^T l'ensemble des séquences de longueur T sur A'. Avec l'introduction du joker, il est nécessaire d'avoir une fonction F permettant de passer de l'espace des sorties du réseaux à des étiquettes avec ou sans joker. Cette fonction supprime d'abord les doublons puis les jokers. Par exemple $F(tt - o - t - t - -o) = totto$.

Dans le cas d'un réseau de neurones récurrent, nous avons des sorties y pour plusieurs instants t d'une séquence. Le problème est de trouver le meilleur chemin de sorties y passant par tous les instants t. Pour trouver le meilleur chemin π il faut optimiser l'équation 1.34 représentant la probabilité d'un chemin particulier qui est le produit des sorties y pour chaque instant t.

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{1.33}$$

Cependant, cette optimisation requiert une connaissance de la vérité terrain pour chaque instant. Or, lors de l'évaluation CTC, on cherche à trouver la meilleure étiquette à travers les différents chemins sans pour autant connaître l'étiquette à chaque instant. Pour remédier à cette difficulté, la probabilité que les étiquettes z d'une séquence peut se calculer par la somme de toutes les probabilités de chemins π alignés par la fonction F sur elles.

$$p(z|x) = \sum_{\pi \in F^{-1}(z)} p(\pi|x) \tag{1.34}$$

C'est la réunion des probabilités des différents chemins sur la même étiquette qui permet à la CTC d'utiliser des données non segmentées, et ce car elle permet au réseau de prédire une étiquette sans en connaître la position.

La CTC s'inspire alors de l'algorithme forward-backward utilisé pour apprendre un modèle de Markov caché pour le calcul de la probabilité conditionnelle $p(z|x)$. Elle s'inspire de ce modèle markovien afin d'inférer à la fois la position et la classe d'un élément (phonème, caractère, etc). L'algorithme forward-backward permet de calculer les deux variables forward α et backward β par les équations 1.35 et 1.36 respectivement où l'on définit les ensembles $V(t,u) = \{\pi \in A'^t : F(\pi) = z_{1:u/2}, \pi_t = z'_u\}$ et $W(t,u) = \{\pi \in A'^{T-t} : F(\pi^* + \pi) = z \ \forall \pi^* \in V(t,u)\}$.

$$\alpha(t,u) = \sum_{\pi \in V(t,u)} \prod_{i=1}^{t} y_{\pi_i}^t \tag{1.35}$$

$$\beta(t,u) = \sum_{\pi \in W(t,u)} \prod_{i=1}^{T-t} y_{\pi_i}^{t+i} \tag{1.36}$$

Après avoir réalisé l'algorithme forward-backward, on peut alors calculer la fonction d'erreur CTC qui est la vraisemblance logarithmique de classifier correctement tous les exemples x d'étiquette z d'un jeu de données S :

$$\mathcal{L}(S) = \sum_{(x,z) \in S} \mathcal{L}(x,z) = -\sum_{(x,z) \in S} \ln p(z|x) \tag{1.37}$$

La probabilité de classifier correctement un exemple se calcule alors à l'aide des variables forward α et backward β suivant l'équation 1.38.

$$\mathcal{L}(x,z) = -\ln \sum_{u=1}^{|z'|} \alpha(t,u)\beta(t,u) \tag{1.38}$$

Une fois ces calculs réalisés, l'apprentissage est identique à celle du RNN. Le gradient est calculé en fonction de l'erreur de sortie obtenue, puis il est rétropropagé à travers le réseau.

Lors de la phase de reconnaissance, un algorithme de décodage est utilisé afin de trouver la solution optimale. L'algorithme le plus efficace est le prefix search decoding [Russell and Norvig, 1995], il réalise une recherche du meilleur chemin dans l'arbre des étiquettes. Cependant, l'algorithme qui nous intéresse davantage est celui du "best path decoding", car il permet un décodage très rapide de la séquence de caractères à prédire. Son principe est de prendre l'étiquette de probabilité maximale pour chaque trame. Puis la séquence d'étiquettes est passée à travers la fonction F donnant ainsi le résultat final.

Ce décodage est très intéressant car il permet d'obtenir des performances brutes très élevées comme nous le verrons par la suite dans les contributions de cette thèse. Cette particularité est due aux sorties très tranchées et nettes obtenues à l'issue d'un entraînement avec la CTC comme on peut le constater sur la figure 1.8. On peut voir sur la figure 1.8 que chacun des caractères à reconnaître a une probabilité proche de 1 sur une durée très courte formant des pics. Hors de ces pics le joker voit généralement sa probabilité proche de 1.

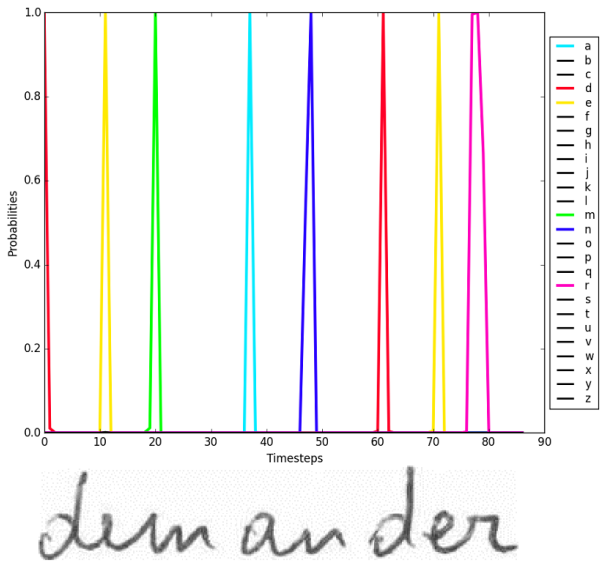


FIGURE 1.8 – Sorties d'un réseau LSTM à l'issue d'un entrainement avec la CTC. Les sorties forment des pics nets pour chaque caractère.

Grâce à l'introduction de la Classification Temporelle Connexionniste, les réseaux BLSTM ont pu être appris sans qu'il soit nécessaire d'annoter les trames, rendant ainsi la tâche moins pénible et augmentant leur attractivité.

1.5.3.4 Sous échantillonnage des séquences

Le sous échantillonnage hiérarchique [LeCun et al., 1998] vise à réunir les éléments d'une séquence (trames) tout au long de la passe en avant du réseau. Les trames aug-

mentent alors en taille et diminuent en nombre. Cette nouvelle couche est une couche dite de bloc. Elle consiste à concaténer les vecteurs issus d'un nombre défini de trames pour former une seule nouvelle trame. On peut aussi parler de couche de pooling au lieu de bloc bien qu'elle ne réalise pas d'opérations de moyenne ou de maximum comme les couches de pooling des réseaux convolutifs (présentés dans la sous-section 1.6.1) mais une concaténation. C'est pour cette raison que nous choisissons le terme de bloc afin de la différencier du terme pooling. Le fait d'ajouter ces couches de bloc permet d'avoir une structure pyramidale des données. En effet, pour l'exemple de la reconnaissance de texte, sur les premières couches l'analyse est fine (largeur de trame de 1 ou 2 pixels), alors que sur les dernières couches l'analyse est plus large (largeur de trame de 6,8,12 pixels). L'utilisation de ces couches permet de considérer des séquences longues, tout en réduisant la disparité des données.

Pour d'autres architectures, ce sous-échantillonnage peut être réalisé à l'aide d'un pas inférieur à la taille de l'élément regroupé afin de rester précis dans la description. Par exemple, en regroupant deux à deux 4 trames de largeurs 1 pixels avec un pas de 1 pixel, on obtient 3 trames de largeur 2 pixels. Pour un réseau récurrent, il n'est pas nécessaire de prendre un tel pas car le contexte est déjà pris en compte par la récurrence.

Dans la figure 1.9, on peut voir un exemple de réseau BLSTM comprenant un empilement hiérarchique des traitements.

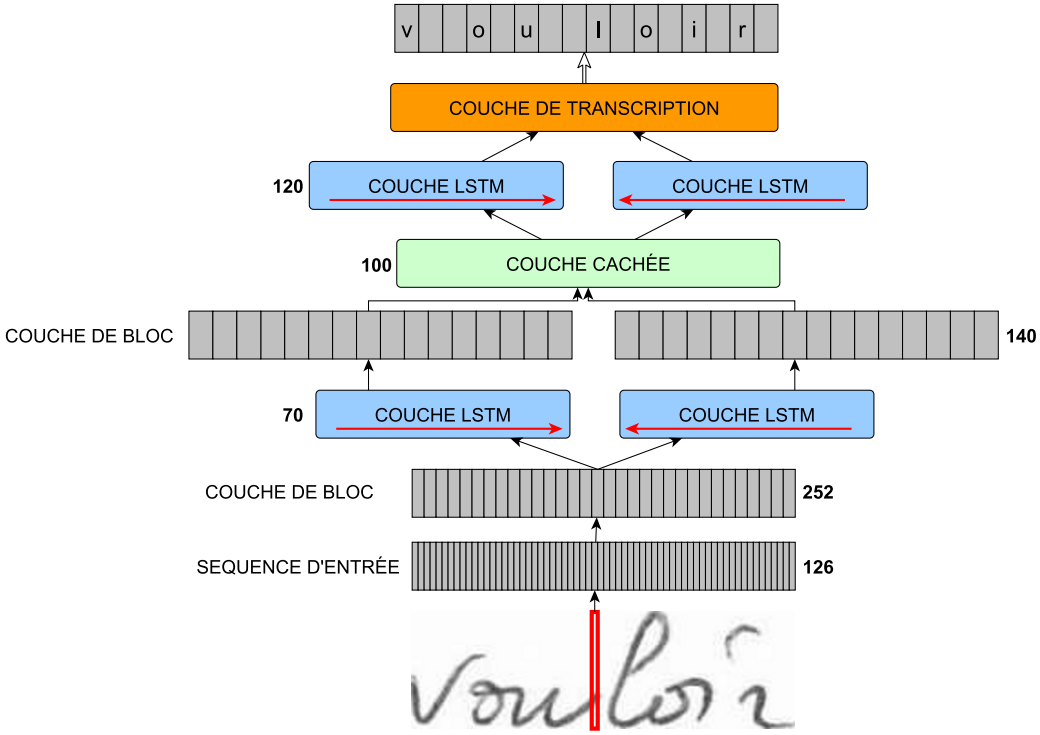


FIGURE 1.9 – Réseau BLSTM avec couches de bloc et de rassemblement.

En résumé, la couche de rassemblement regroupe les données de plusieurs couches représentant les directions d'analyse de la séquence sans modifier le découpage en trames.

La couche de rassemblement peut être une couche cachée : elle a alors un rôle de sous-échantillonnage. Dans notre exemple sur la figure 1.9, elle rassemble deux vecteurs de taille 140 en une sortie de couche cachée de taille 100. La couche de bloc agit sur le nombre de trames en les regroupant par concaténation. En revanche, elle n'agit pas sur les données. Sur l'exemple de la figure figure 1.9, la deuxième couche de bloc réunit pour chaque direction les deux vecteurs de sorties de couche LSTM de taille 70 de deux trames en un seul vecteur de taille 140.

Ces couches ne sont pas souvent mises en avant mais ont pourtant été utilisées avec succès [Graves, 2012, Mioulet et al., 2015] et notamment pour les réseaux LSTM multi-dimensionnel [Graves and Schmidhuber, 2009, Pham et al., 2014].

1.5.3.5 LSTM multidimensionnel (MDLSTM)

Les réseaux BLSTM sont très intéressants pour le traitement des séquences 1D. Néanmoins, une grande partie des problèmes actuels concerne la vision par ordinateur et donc des signaux à deux dimensions. A. Graves a voulu étendre les réseaux LSTM au cas multidimensionnel [Graves and Schmidhuber, 2009] notamment pour la reconnaissance de l'écriture manuscrite qui a pour point de départ des images 2D d'un texte mais où le but est d'obtenir une séquence de caractères 1D.

La figure 1.10 présente un MDLSTM 2D pour la reconnaissance de l'écriture. On peut voir des changements sur le nombre de couches par niveau. Un réseau à d dimensions aura alors généralement 2^d directions et, comme énoncé précédemment, il aura par conséquent 2^d couches par niveau LSTM. Il peut y avoir moins de directions, cependant la méthode perd alors de son intérêt, dans la mesure où la mémorisation dans certains sens ne se fera pas et de l'information sera perdue.

Sur cette figure 1.10, les directions liées a un problème 2D de type image sont représentées par le passage des carrés oranges à $t-1$ vers les rouges à t. La couche de rassemblement est représentée différemment par rapport au BLSTM vu précédemment, mais fonctionne de la manière identique. La couche de rassemblement réunit les vecteurs de sorties des couches LSTM représentant les quatre directions en un vecteur de taille $4 \times 2 = 8$, pour chaque morceau de la matrice considérée.

En plus des changements au niveau des couches, les formulations du bloc de mémoire LSTM ont été modifiées et étendues au cas multidimensionnel. Le premier élément est le passage d'une porte d'oubli à d portes d'oubli (figure 1.11), où d est le nombre de dimensions. Ensuite le calcul de la récurrence (en vert précédemment) est sommé pour toutes les dimensions du problème. C'est-à-dire que dans un cas 2D, par exemple, la récurrence suivant la valeur précédente d'une dimension sera sommée à la récurrence suivant la valeur précédente de l'autre dimension (symbolisé par un lien cyan et magenta sur la figure 1.11). Le nombre de poids de peephole (en orange précédemment) dépend du nombre de dimensions. La porte d'entrée est calculée suivant toutes les dimensions pour les poids d'entrée et récurrents. Le mécanisme de la porte d'oubli devient plus complexe car chaque porte d'oubli a ses propres poids d'entrée et ses propres poids pour les peephole. Lors du calcul lié aux peephole, seul le calcul dans la dimension liée à la porte d'oubli est considéré. La récurrence multiple suivant chaque dimension s'applique également au calcul de l'entrée de la cellule et à la porte de sortie. Le peephole de la porte de sortie

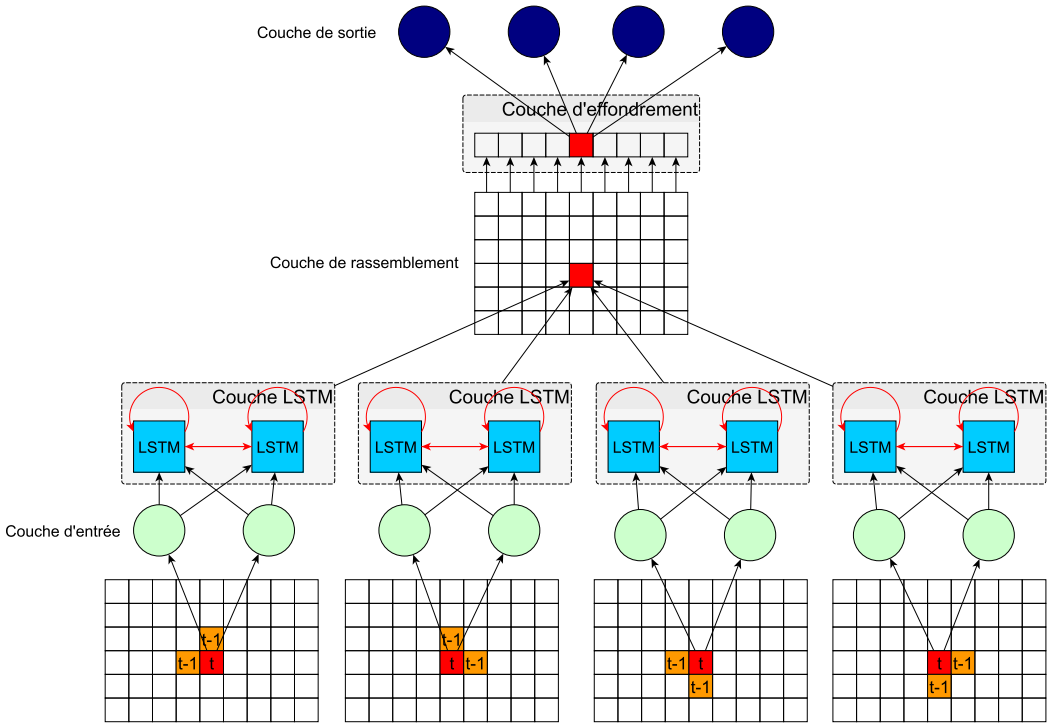


FIGURE 1.10 – 2DLSTM avec une couche d'effondrement pour le passage d'un problème 2D à 1D.

reste inchangé car il observe l'état courant de la cellule.

Pour la reconnaissance d'écriture, l'ajout d'une couche d'effondrement est nécessaire (collapse layer en anglais) pour passer d'une représentation à deux dimensions à une séquence. Cette couche d'effondrement est réalisée en deux étapes et est schématisée sur la figure 1.10 qui n'est pas une représentation idéale de la couche. La première étape consiste à rassembler avec une couche cachée classique comme pour la couche de rassemblement, mais sans fonction sur l'activation. Par définition, le nombre de neurones de la couche d'effondrement est égal au nombre de sorties du réseau. Lorsque l'activation a été calculée, l'opération d'effondrement a lieu et les activations sont sommées suivant la dimension que l'on souhaite faire disparaître (effondrer). En n'appliquant pas de fonctions d'activation et en ayant un nombre de neurones égal aux nombres de sorties, on peut appliquer directement une fonction d'activation softmax sur le vecteur de sortie de la couche d'effondrement. Par exemple, La dimension d'effondrement pour la reconnaissance d'écriture est la verticale, dans la mesure où la plupart des écritures se lisent selon l'horizontale. Néanmoins, c'est l'inverse pour le chinois ou bien encore le japonnais.

Une fois l'effondrement effectué, le nombre total d'activations est donc égal au nombre de sorties du réseau. Comme cela est illustré sur la figure 1.10, chaque carré de la couche d'effondrement possède 4 activations. La fonction softmax est ensuite directement appliquée sur ces activations. Il est possible d'appliquer le sous échantillonnage hiérarchique au MDLSTM afin de de ne pas utiliser la couche d'effondrement dans le cas de problèmes de taille fixe. Aujourd'hui, l'utilisation des MDLSTM est remise en question ([Puigcerver,

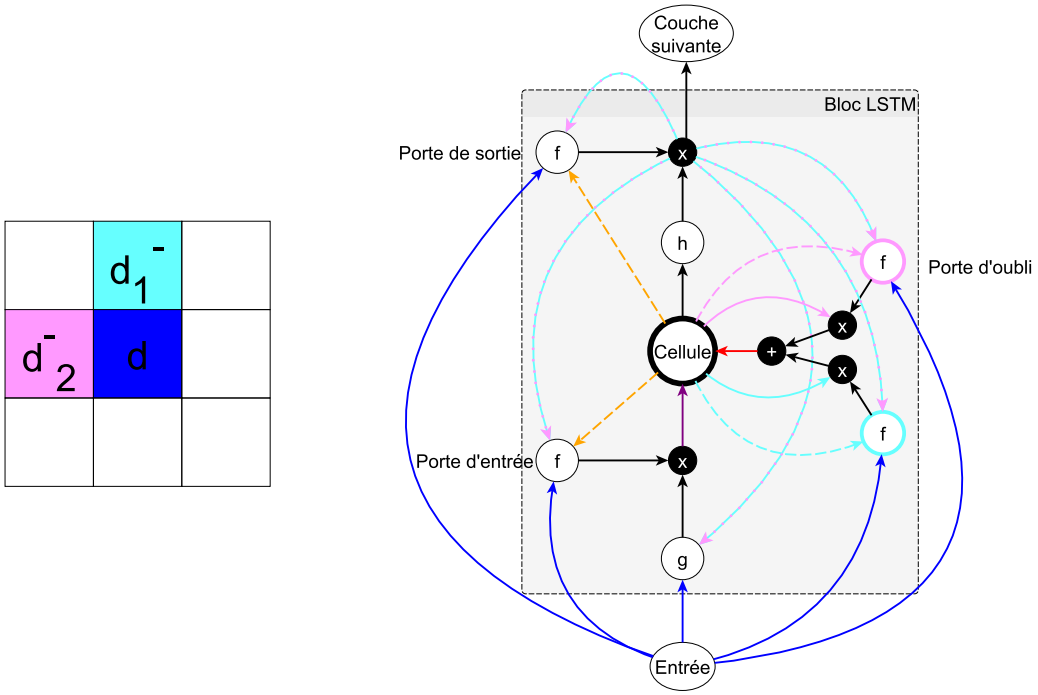


FIGURE 1.11 – Bloc de mémoire LSTM 2D

2017]). Celle-ci s'explique du fait que les réseaux convolutifs sont plus efficaces pour extraire des caractéristiques des images et donc pour prendre en compte la nature 2D des images, tout en étant plus légers en terme de nombre de paramètres.

1.6 Autres architectures profondes

1.6.1 Réseau de neurones à convolutions (CNN)

Les réseaux convolutifs sont conçus pour l'extraction de caractéristiques d'une image et sont essentiellement utilisés en vision par ordinateur. Le premier réseau apparenté aux réseaux convolutifs est le néocognitron [Fukushima and Miyake, 1982]. Ils ont été introduits dans leur forme actuelle en 1990 pour la classification de chiffres manuscrits [LeCun et al., 1990]. Ils sont utilisés pour de nombreuses applications où ils excellent comme par exemple : la classification d'images [Krizhevsky et al., 2012], la segmentation d'images bio-médicales [Ronneberger et al., 2015, Belharbi et al., 2017], la coloration d'images noir et blanc en couleur [Zhang et al., 2016] ou bien encore pour jouer en temps réel à des jeux vidéos Atari à partir de l'image vidéo du jeu [Mnih et al., 2013]. Une équipe de Google Deepmind a très récemment battu les meilleurs joueurs mondiaux au jeu de GO avec *AlphaGo Zero* [Silver et al., 2017] une architecture profonde basée sur des convolutions.

1.6.1.1 Principe

Un réseau convolutif possède une architecture similaire à un réseau de neurones profond et le principe de calcul est identique. Les réseaux convolutifs sont des architectures utilisées principalement pour des problèmes de traitement d'image et de vision par ordinateur. Ils peuvent être utilisés pour le traitement de la langue naturelle ou pour des problèmes séquentiels, mais ils n'y sont pas directement adaptés. Les réseaux convolutifs permettent de prendre en entrée une image en niveau de gris ou en couleur ou bien un ensemble de matrices de même taille. Grâce à leurs filtres ou noyaux, ils créent un ensemble d'images filtrées représentant des cartes de caractéristiques de l'image. À partir de ces cartes de caractéristiques, des couches cachées sont ajoutées. L'objectif est de séparer les caractéristiques afin de réaliser une classification ou une régression.

Un réseau convolutif se compose généralement d'une succession de couches convolutives et de couches de pooling, comme montré sur la figure 1.12. Une couche de convolution est constituée d'un ensemble de filtres qui sont appliqués à l'image. Sur la figure 1.12, la première couche est composée de 6 filtres. Un filtre est un neurone identique à ceux présentés pour le MLP avec un biais. L'entrée de ce neurone est un patch rectangulaire 2D d'une image. Les valeurs des pixels du patch sont multipliées par les poids qui sont représentés sous la même forme. L'image est alors parcourue avec ou sans recouvrement et donne lieu à de nouvelles images "filtrées". Les filtres [Zeiler and Fergus, 2014] des premières couches sont généralement plus grands en taille, peu nombreux et ils extraient des caractéristiques géométriques (lignes, courbes,etc pour la première couche par exemple). On peut alors parler de champs réceptifs. Plus on évolue dans les couches plus les filtres sont petits, nombreux et extraient des formes complexes. Les couches de pooling fonctionnent de manière similaire aux couches de bloc vues pour les réseaux récurrents. Elles ont pour objectif de réduire la taille de l'image. Le pooling applique un patch aux points de l'image avec ou sans chevauchement. Toutefois, contrairement au couche de bloc, il ne concatène pas les valeurs dans un vecteur. Le pooling applique une fonction qui peut être de plusieurs natures, les plus utilisés étant le maximum (max pooling) ou la moyenne (mean pooling) des valeurs, permettant un sous échantillonnage.

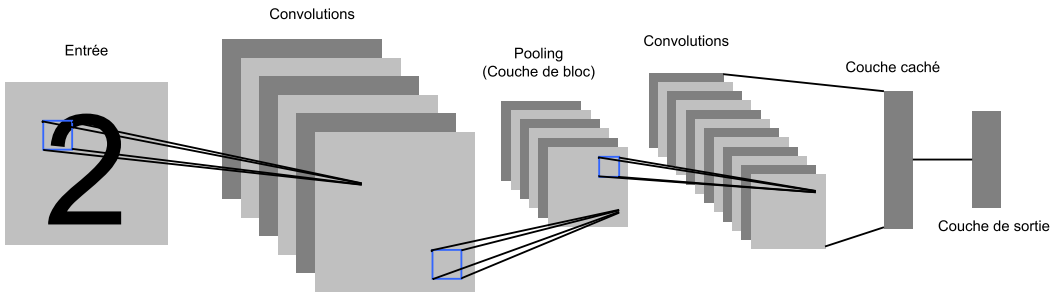


FIGURE 1.12 – Réseaux de neurones convolutifs : CNN.

Par ces propriétés, les CNN sont donc utilisés pour de nombreuses applications de vision par ordinateur. Ils sont également combinés à des modèles RNN pour résoudre des problèmes alliant vision par ordinateur et traitement de la langue naturelle, tels que la reconnaissance de textes dans les images ou la génération de description à partir d'images.

1.6.2 Réseaux hybrides DNN, CNN, RNN

Il existe de nombreuses problématiques qu'un réseau profond comme un DNN, un RNN ou un CNN seul ne peut pas résoudre. Comme nous l'avons vu, chacune de ces architectures possède ses particularités. Les réseaux profonds permettent de construire des caractéristiques dans un espace non linéaire. Les réseaux de neurones récurrents sont des réseaux qui ont une mémoire autorisant une modélisation séquentielle des signaux. Les réseaux convolutifs produisent une représentation, une caractérisation de signaux à plusieurs dimensions.

Dans la littérature, de nombreux articles utilisent ces différentes propriétés pour construire des réseaux complexes capables de traiter des problèmes combinant traitement de la langue naturelle et vision par ordinateur dans des espaces de représentation de grande dimension. Dans cette thèse, nous nous intéressons à la reconnaissance de l'écriture qui est un problème de traitement de la langue naturelle et de vision par ordinateur. Pour résoudre ce type de problèmes, les premières architectures alliant CNN et RNN ont été utilisées pour la description des images [Vinyals et al., 2015, Donahue et al., 2015]. Ces réseaux sont composés d'une partie convolutive capable d'interpréter l'image et d'une partie récurrente capable de générer une séquence de mots qui décrit l'image. Une architecture de réseau appelée CLDNN [Donahue et al., 2015] a donné des résultats à l'état de l'art pour la reconnaissance de parole. Elle est composée de trois parties : d'abord une convolutive, puis une récurrente avec LSTM, enfin des couches profondes.

Un autre exemple d'architecture hybride basée sur des couches CNN et MDLSTM entrelacées a été proposé pour résoudre le problème de la reconnaissance de l'écriture [Voigtlaender et al., 2016]. Un schéma de cette architecture est présenté sur la figure 1.13. Cette architecture fait partie des architectures les plus performantes à l'état de l'art pour la reconnaissance de l'écriture car elle permet d'extraire des caractéristiques grâce aux convolutions tout en exploitant des couches MDLSTM pour une analyse contextuelle de l'image. Cette architecture fonctionne de la même manière que l'architecture MDLSTM présentée précédemment, des couches convolutives et de pooling s'insèrent entre les couches LSTM. Le CNN MDLSTM présenté dans [Voigtlaender et al., 2016] utilise également des couches de rassemblement et une couche d'effondrement.

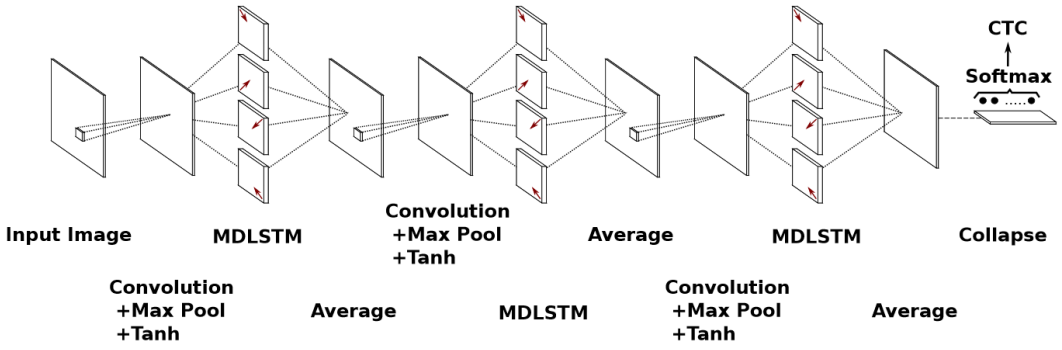


FIGURE 1.13 – Schéma de l'architecture CNN MDLSTM proposé dans [Voigtlaender et al., 2016].

1.7 Conclusion

Depuis la fin des années 2000, les réseaux de neurones ont fait une véritable avancée grâce à l'apprentissage d'architectures profondes. Ils ont permis de grands progrès dans de nombreux domaines comme le médical, la vision par ordinateur, ou encore le traitement de la langue naturelle. Lors d'un séminaire à Carnegie Mellon Univ. le 18-Nov-2016, Y. LeCun a identifié trois problèmes comme frontières de l'intelligence artificielle des réseaux de neurones. Ces trois problématiques sont : i) la puissance de calcul nécessaire pour les apprentissages, ii) les données étiquetées nécessaires pour les apprentissages et iii) les capacités de mémoire insuffisantes.

Malgré les limites techniques actuelles, ces réseaux profonds ont aujourd'hui pris une place importante et sont largement utilisés dans la vie quotidienne : reconnaissance vocale, traduction, filtres anti-spam, etc. Ils se ont également développés dans le domaine de l'analyse et de la reconnaissance de documents et tout particulièrement pour la reconnaissance de l'écriture. Notre prochain chapitre s'intéresse donc à la reconnaissance de l'écriture avec des réseaux récurrents.

Chapitre 2

Reconnaissance de l'écriture manuscrite

Contents

2.1 Introduction

La reconnaissance de l'écriture est une problématique liée à la numérisation des écrits pour leur conservation et pour faciliter leur consultation. Le papier est un support qui a pour inconvénient de ne pas être pérenne car sensible à l'environnement extérieur. De plus, le papier nécessite de grands espaces de stockage car il est volumineux. En opposition, le document numérique ne prend pas de place et autorise la recherche d'informations. En effet, on ne peut pas faire une recherche automatique d'un mot sur du papier, sa consultation est parfois peu aisée.

La reconnaissance de l'écriture est le processus de transcription d'une écriture en une chaîne de caractères numériques. Les deux grands types d'écritures sont l'écriture dactylographiée produite par impression à l'aide d'une machine et l'écriture manuscrite provenant de l'homme. La reconnaissance de l'écriture dactylographiée est facilité par la régularité des caractères et la présence d'espaces entre les caractères. La reconnaissance consiste généralement à réaliser une segmentation en caractères puis une reconnaissance des caractères. Aujourd'hui, les principales difficultés liées à la reconnaissance de l'imprimé sont dues aux documents dégradés, aux documents anciens ou de mauvaise qualité. Dans cette thèse, nous nous concentrons sur la reconnaissance de l'écriture manuscrite car elle présente, en plus des difficultés énoncées précédemment, des difficultés dues aux nombreux styles d'écriture, ou encore aux irrégularité de formes. Nous nous inspirons en partie de l'état de l'art de la thèse de T. Bluche [Bluche, 2015] et notamment des nombreuses références citées sur l'histoire de la reconnaissance de l'écriture manuscrite.

La reconnaissance de l'écriture est un domaine d'application qui a été abordé dès les années 60 [Earnest, 1960]. Il a par la suite connu un essor dans les années 90 [Plamondon and Srihari, 2000, Steinherz et al., 1999], date à laquelle les premiers ateliers internationaux (IWFHR devenu ICFHR) sur la reconnaissance de l'écriture manuscrite ont eu lieu. Les raisons du fort intérêt de la recherche pour la reconnaissance de l'écriture dans les années 90 étaient motivées par deux applications industrielles majeures : la reconnaissance d'adresses pour le tri automatique du courrier et la lecture automatique de chèques. L'objectif premier était la reconnaissance des chiffres : codes postaux pour les adresses et montants pour les chèques. De nombreux articles traitent de la reconnaissance de lignes d'adresses [Gilloux, 1993, Srihari, 1993, Srihari and Kuebert, 1997, El-Yacoubi et al., 2002] et d'autres de la reconnaissance des chèques [Paquet and Lecourtier, 1993, Guillevic and Suen, 1995, Le Cun et al., 1997, Gorski et al., 1999]. Avec le progrès des méthodes de reconnaissance, elles permettent actuellement de traiter des documents complexes. La complexité peut alors provenir de la langue avec des lexiques de taille conséquente ou de la forme avec des documents dégradés. On peut citer parmi ces documents complexes les courriers manuscrits de la compétition RIMES [Grosicki and El Abed, 2009] ou les documents anciens de la compétition READ [Sánchez et al., 2016]. Les applications de la reconnaissance de l'écriture ont aussi évolué et traitent par exemple du repérage de mots [Fischer et al., 2010, Thomas et al., 2010] ou de l'extraction d'informations [Chatelain et al., 2006a].

Concernant la reconnaissance de l'écriture manuscrite, on distingue la reconnaissance dite en-ligne de la reconnaissance dite hors-ligne [Plamondon and Srihari, 2000]. Bien que

divisée en deux, l'objectif est identique et vise à transformer des ensembles de pixels en chaîne de caractères. Les mêmes techniques tendent à être appliquées, mais la reconnaissance hors ligne est réputée plus difficile du fait de l'absence de l'information temporelle. La distinction entre les deux reconnaissances se fait à travers le mode de capture de l'écriture qui est sous forme de vecteurs 1D acquis au fil du temps pour la reconnaissance en-ligne, et sous forme d'images 2D pour la reconnaissance hors-ligne.

La reconnaissance en-ligne réalise l'acquisition de données sous forme d'un vecteur 1D. Les données peuvent provenir d'un stylo ayant des capteurs ou interagissant avec une dalle électronique. Les informations contenues dans ces données sont diverses comme la position du stylo ou du doigt, ou encore la pression exercée. La majorité des applications de la reconnaissance en ligne se fait autour des smartphones, des tablettes et autres écrans tactiles.

La reconnaissance de l'écriture manuscrite hors-ligne est la reconnaissance de l'écriture uniquement à partir de l'image de l'écriture produite. Il faut donc au préalable effectuer une numérisation de l'écriture en une image de texte. Elle se prête à des applications comme la reconnaissance de chèques ou d'adresses postales. Dans cette thèse, nous nous intéressons uniquement à la reconnaissance de l'écriture manuscrite hors-ligne, bien que certaines des contributions peuvent être utiles à la reconnaissance en ligne.

La première difficulté vient du fait qu'une image numérisée est organisée dans une matrice en deux dimensions de valeurs entières. La reconnaissance consiste à interpréter un signal en deux dimensions et à le transformer en une chaîne de caractères. La qualité avec laquelle la numérisation est effectuée est aussi importante. Il est difficile d'interpréter lors de la reconnaissance des pixels qui ne représentent pas bien le caractère car acquis en basse résolution ou dont le contraste n'est pas bon.

Les systèmes de reconnaissance automatique de l'écriture manuscrite sont confrontés à la difficulté de devoir reconnaître des styles d'écritures différents, irréguliers, y compris lorsqu'elle provient d'un même scripteur. Il existe deux styles d'écriture bien déterminés que sont l'écriture cursive (ou attachée) et l'écriture en script. L'écriture cursive pose un problème sur la segmentation en caractères car il n'y a plus d'espace entre les caractères. À cette difficulté s'ajoute le style de chaque scripteur conduisant à une grande variabilité des formes, des caractères, et par conséquent des mots, rendant ainsi plus difficile leur segmentation et leur reconnaissance. La figure 2.1 représente différents styles d'écriture du mot salutations.

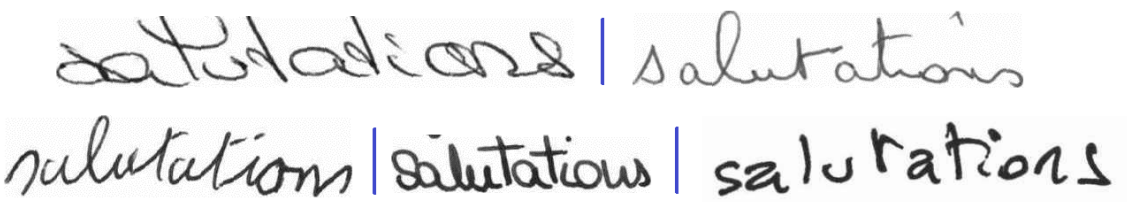


FIGURE 2.1 – Différentes écritures du mot "salutations" par différents scripteurs sur la base RIMES.

De plus, un scripteur peut écrire un même caractère de différentes façons. Le cas le

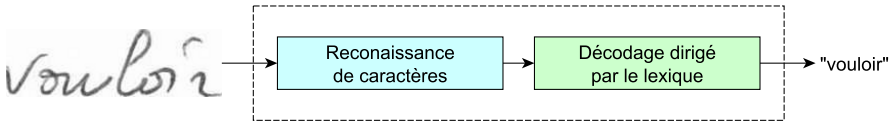


FIGURE 2.2 – Le paradigme de reconnaissance de l'écriture actuel.

plus difficile apparaît lorsqu'un même scripteur représente deux caractères différents par la même forme. Un tel exemple est observable sur le mot "salutations" au milieu de la ligne du bas de la figure 2.1, où les lettres 'u' et 'n' sont très similaires. De ce fait, il est important d'être capable de reconnaître le mot afin de reconnaître le caractère et de pouvoir le segmenter également. L'écriture d'une personne peut être difficile à déchiffrer, c'est alors le contexte qui nous aide à comprendre et à reconnaître.

Le contexte prend alors de l'importance lors de la reconnaissance des caractères. Cependant, la connaissance et la prise en compte du contexte présentent également des difficultés. Pour la reconnaissance de l'écriture, il est important de connaître la langue et le vocabulaire liés au contexte de la reconnaissance car ils permettent d'aider la reconnaissance des caractères. Il est en effet plus facile pour un homme de lire et reconnaître des mots dans une langue qui lui est familière.

Dans ce chapitre, nous allons voir succinctement les méthodes de reconnaissance de l'écriture qui ont été les plus utilisées et qui sont actuellement à l'état de l'art. La majorité des méthodes utilisées suivent le même paradigme de reconnaissance. Ce paradigme consiste à réaliser une reconnaissance optique de caractères, puis à intégrer le contexte linguistique adapté à l'aide de méthodes dirigées par le lexique ou de modèles de langage. La reconnaissance de caractères donne des hypothèses de caractères qui sont ensuite réévaluées par les méthodes dirigées par le lexique ou les modèles de langage. Un schéma de ce paradigme de reconnaissance est présenté de manière simplifiée sur la figure 2.2.

Tout d'abord, nous présentons les étapes préliminaires à la reconnaissance de l'écriture. En effet, la reconnaissance de l'écriture fait partie de l'analyse de documents. Nous présentons donc un aperçu des méthodes utilisées pour l'extraction des zones de textes, la segmentation en lignes, en mots, mais aussi les pré-traitements possibles.

La section suivante présente les premières étapes de la reconnaissance de l'écriture qui sont la segmentation en caractères et l'extraction de caractéristiques.

Ensuite nous présentons les méthodes de reconnaissance de caractères, c'est-à-dire la reconnaissance optique des caractères (OCR). Ce sont les méthodes qui transforment les données des pixels ou caractéristiques en une séquence d'hypothèses de caractères matérialisée par des probabilités de caractères.

Des méthodes de décodage dirigé par le lexique incluant une modélisation de la langue sont par la suite utilisées afin d'exploiter le contexte pour résoudre les ambiguïtés et fournir la meilleure reconnaissance possible. Nous présentons ces méthodes de décodage dirigé par le lexique qui constituent un élément important dans le paradigme de reconnaissance actuel des systèmes à l'état de l'art.

Lorsque le système de reconnaissance est constitué, il faut être capable de l'évaluer. Nous présentons donc les différentes métriques d'évaluation de la reconnaissance de l'écri-

ture. Enfin, nous présentons les bases de données publiques que nous utiliserons pour évaluer les performances des propositions que nous faisons dans cette thèse.

2.2 Étapes préliminaires à la reconnaissance de l'écriture

Nous disposons des images de documents numérisés et nous souhaitons reconnaître les textes manuscrits qu'elles contiennent. Il faut identifier les zones de textes et les segmenter en lignes. Nous n'abordons pas cette question au cours de cette thèse mais elle reste une étape importante. L'analyse de l'image débute par une étape d'extraction des lignes d'écriture.

2.2.1 Détection des zones d'écriture

Dans des applications de reconnaissance de l'écriture comme la lecture automatique de chèques, il n'y a pas de difficulté liée à l'extraction des lignes de texte et notamment du montant car les chèques sont tous formatés de la même façon et l'information se trouve toujours au même endroit. En revanche, pour des applications comme la lecture d'adresses, une localisation du bloc d'adresse est nécessaire avant sa lecture [Yeh et al., 1987, Viard-Gaudin and Barba, 1991, Palumbo et al., 1992, Wolf et al., 1997]. Il existe une grande variabilité de documents pour lesquels une localisation des zones de texte est nécessaire. Une compilation de documents variés nécessitant une extraction de zones de texte est présentée dans le corpus de la base de la campagne Maurdor d'analyse de documents [Brunessaux et al., 2014]. Une image d'un formulaire de la base Maurdor est présentée à gauche de la figure 2.3. Elle montre les difficultés d'extraction des zones de texte, sachant qu'il n'y pas de formulaire type dans la base. Il existe de nombreuses méthodes permettant d'identifier les différents blocs [Barlas et al., 2014], dont les zones de textes. Ceux-ci constituent la structure physique d'un document.

Dans [Wong et al., 1982], un algorithme de segmentation de documents est proposé. Cet algorithme est le RLSA (Run-Length Smoothing Algorithm). Il permet de noircir les espaces blancs entre des pixels noirs si la distance est inférieure à un seuil, construisant ainsi des blocs. Une autre méthode se base sur l'agrégation de composantes connexes [O'Gorman, 1993]. Cette agrégation se fait au niveau pixel pour former des caractères, puis de ces caractères sont formés des lignes et de ces lignes, des blocs textes. L'avantage de cette méthode est son indépendance à la rotation et à l'espacement du texte. Elle fonctionne également avec différentes orientations sur une même page. [Breuel, 2002] propose une méthode de recherche d'espaces blancs permettant la segmentation en colonnes. Cette méthode a un double intérêt qui est sa simplicité d'implémentation et son absence d'heuristique. Elle se base sur la recherche de rectangles blancs maximaux. Dans [Kumar et al., 2007], les auteurs utilisent une méthode fondées sur les ondelettes comme caractéristiques d'un classifieur de Fisher. Puis, pour affiner les résultats au niveau pixel, un champ de Markov caché est utilisé. Dans [Antonacopoulos et al., 2011], il est montré que les méthodes les plus performantes sont celles basées sur les espaces blancs et les composantes

connexes. Aujourd'hui, les méthodes se tournent vers l'utilisation de réseaux convolutifs profonds pour réaliser la segmentation du document [Viana and Oliveira, 2017, Chen et al., 2017].

Lorsque les zones de textes sont identifiées, l'étape suivante consiste à isoler les lignes de textes. Ce processus est important car il impacte la qualité de la reconnaissance [Romero et al., 2015]. Une segmentation en lignes d'un document ancien est représentée sur la figure 2.3. C'est un processus difficile pour lequel de nombreuses méthodes ont été proposées, et des revues du sujet publiées [Likforman-Sulem et al., 2007, Louloudis et al., 2009]. Les principales difficultés résident dans la superposition partielle des lignes d'écriture du fait de la présence de hampes et jambages, mais aussi des lignes penchées rendant difficile leur extraction. Les méthodes à l'état de l'art actuelles utilisent des réseaux profonds convolutifs ou entièrement convolutifs [Diem et al., 2017]. Après avoir été extraites, ces lignes peuvent s'avérer difficiles à reconnaître, nécessitant ainsi des pré-traitements.

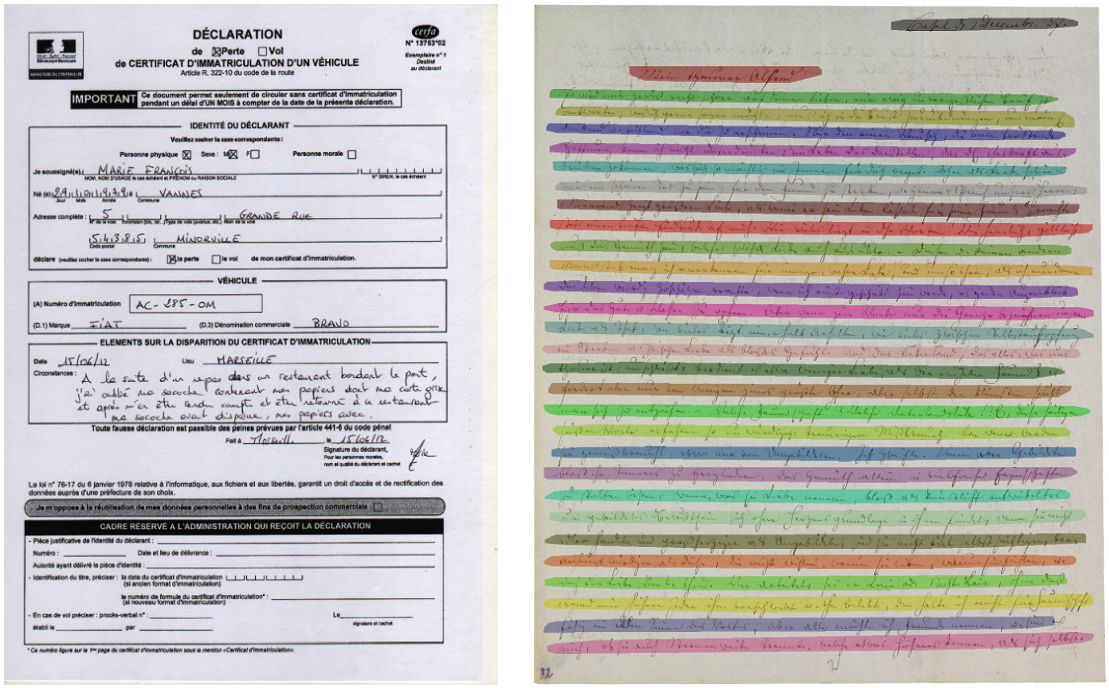


FIGURE 2.3 – À gauche : image d'un formulaire issu de la base Maurdor sur lequel il faut trouver les zones de textes. À droite : Segmentation en lignes d'une page de texte issue de la base Read.

2.2.2 Prétraitements

Il existe de nombreuses perturbations qui peuvent apparaître sur une ligne de texte à reconnaître. Les principales perturbations sont : le contraste de l'écriture par rapport au fond, une écriture penchée (en italique), une ligne d'écriture inclinée. L'objectif des prétraitements est de diminuer la variabilité et d'améliorer la qualité des lignes d'écriture afin de faciliter la reconnaissance.

Les problèmes de contraste de l'écriture pouvant apparaître dans une image sont nombreux. En effet, l'écriture peut être présente sur un fond qui n'est pas uniformément blanc, sur des fonds colorés ou texturés. L'écriture peut également avoir terni ou elle peut être visible au travers de la feuille. Toutes ces difficultés amènent au problème de nettoyage de l'image où l'on cherche à séparer les pixels sombres qui correspondent à l'écriture, des pixels clairs qui correspondent au fond. Idéalement, on souhaite binariser l'image en espérant parvenir à séparer ces deux informations. L'approche naïve consiste à définir un seuil de binarisation au dessus duquel les pixels sont transformés en pixels blancs et en dessous en noirs. Cette méthode pose un problème lorsqu'il y a des documents ayant des contrastes différents. La méthode de binarisation d'Otsu [Otsu, 1979] permet de le résoudre en utilisant un seuil variable basé sur l'histogramme des valeurs des pixels. Cependant cette méthode est globale à une image et ne prend pas en compte les variations locales de contraste. Des méthodes de binarisation locale ont donc été proposées dans ce sens [Sauvola and Pietikäinen, 2000, Wolf et al., 2002, Gatos et al., 2006]. Ces méthodes sont notamment efficaces pour les lignes de textes de documents abîmés ou anciens. Une autre méthode [Pesch et al., 2012] propose une augmentation de contraste. L'image reste en niveau de gris et deux seuils sont utilisés : l'un pour les pixels les plus sombres tous mis à 0 et l'autre pour les plus clairs mis à 255 ; les valeurs intermédiaires sont lissées. De plus en plus, les méthodes de reconnaissance n'utilisent plus la binarisation et s'appliquent directement aux images en niveau de gris.

En plus du problème de contraste, les personnes peuvent écrire de manière penchée, comme on peut le voir sur la figure 2.4. Afin de faciliter la reconnaissance, il est préférable que toutes les écritures soient droites. On peut mesurer le degré d'inclinaison de l'écriture notamment grâce aux hampes et aux jambages. Pour y parvenir, il suffit d'isoler toutes les barres verticales longues en supprimant toutes les composantes longues horizontales. Les barres verticales délimitent alors des zones dont l'angle est calculé par rapport à l'horizontale et la verticale au centre de gravité de la zone. L'angle d'inclinaison est la moyenne des angles de toutes les zones [Bozinovic and Srihari, 1989, Papandreou and Gatos, 2012]. Le texte est ensuite redressé par cisaillement ("shearing" en anglais) de l'angle trouvé. Il existe de nombreuses autres méthodes comme celles basées sur des histogrammes effectuant des cisaillements locaux [Vinciarelli and Luettin, 2001], ou des projections [Pastor et al., 2004] ou encore des méthodes qui exploitent une combinaison de plusieurs méthodes de projections [Kozielski et al., 2013a].

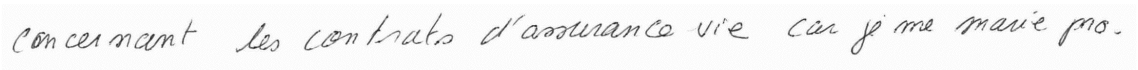


FIGURE 2.4 – Image d'une ligne de la base RIMES dont l'écriture est penchée.

L'un des derniers problèmes de la reconnaissance de l'écriture est l'inclinaison des lignes. Sans repères sur une feuille blanche, il est en effet difficile de maintenir son écriture horizontale. Ce travers peut avoir une influence importante sur la ligne à reconnaître. C'est ce que montre la figure 2.5 où des morceaux des mots des lignes voisines sont présents. La reconnaissance est plus facile lorsque la ligne d'écriture est rectiligne, car cela permet d'éviter de prendre des hampes et jambages d'autres lignes. Afin d'estimer l'angle

d'inclinaison d'une ligne de texte, des méthodes de recherche utilisent la ligne de base de la segmentation en ligne comme dans [Senior and Robinson, 1998, Vinciarelli and Luettin, 2001]. Une revue est également disponible [Hull, 1998] et présente des méthodes basées sur le document entier. Certaines méthodes appliquent ces techniques sur des segments de lignes car il est fréquent qu'une ligne soit courbée [Bunke et al., 2004, Pesch et al., 2012].

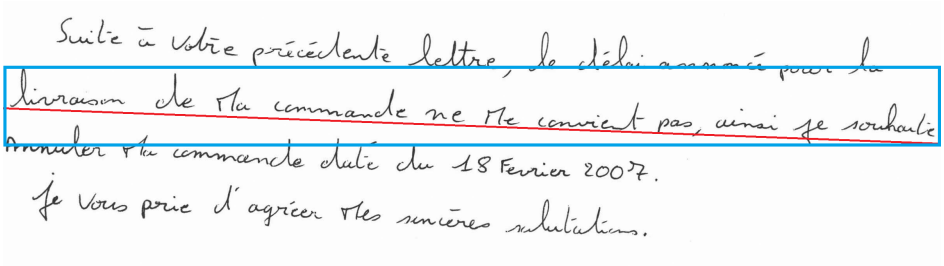


FIGURE 2.5 – Image d'une personne ayant une ligne d'écriture particulièrement inclinée soulignée en rouge et encadrée en bleu.

Lorsque les pré-traitements ont été effectués, l'étape suivante est d'extraire des caractéristiques locales de l'image (matrice de pixels).

2.3 Extractions des caractéristiques locales

La première étape de l'extraction des caractéristiques locales consiste à réaliser une segmentation explicite ou implicite de la ligne de texte en caractères ou en sous unités plus petites appelées graphèmes. À partir des ces graphèmes, deux stratégies sont envisageables : soit l'extraction de vecteurs de caractéristiques, soit l'utilisation direct des pixels.

2.3.1 Segmentation en caractères

Les méthodes fondées sur une segmentation en caractères sont des méthodes historiques. La segmentation en caractères est rendue compliquée par la non connaissance du mot, la cursivité de l'écriture et par les différents styles d'écritures. Pour contourner ce problème, de nombreuses méthodes procèdent à une sur-segmentation en sous-unités appelées graphèmes et à une combinaison de plusieurs graphèmes pour former des caractères lors de la reconnaissance. Les méthodes de la littérature cherchent généralement à analyser le contour où sont en rapport avec. Dans [Edelman et al., 1990], des points sont extraits à partir des différents traits du contour. Il existe plusieurs méthodes de segmentation en caractères ou graphèmes, leur objectif est de trouver les connexions entre caractères à l'aide généralement d'heuristiques. On peut trouver de multiples références sur cette question : [Casey and Lecolinet, 1996, Lu and Shridhar, 1996]. [Kim and Govindaraju, 1997] cherche les ligatures entre les caractères en analysant l'épaisseur des traits, les concavités et convexités. Dans [Morita et al., 2001, El-Yacoubi et al., 1999], les auteurs segmentent en analysant la convexité du contour supérieur.

Cette segmentation en caractères donne cependant lieu à des erreurs, notamment de sous segmentation, tout en sur-découpant l'écriture pour garantir un taux de présence suffisant des séparateurs entre caractères. C'est l'utilisation d'une fenêtre glissante [Kaltenmeier et al., 1993] qui a été retenue car elle permet une segmentation limitant les erreurs de sous segmentation, au prix d'un sur-découpage important. La fenêtre glissante extrait des caractéristiques suivant sa largeur et en respectant un pas de déplacement. Une forme de fenêtre glissante est aussi utilisée par les méthodes neuronales.

2.3.2 Caractéristiques

Avant l'avènement de l'apprentissage profond, les méthodes de reconnaissance de l'écriture s'appuyaient sur l'extraction de caractéristiques. Les techniques étaient variées, et les caractéristiques étaient extraites soit sur des caractères segmentés [Casey and Lecolinet, 1996] soit à l'aide d'une fenêtre glissante [Kaltenmeier et al., 1993] de largeur fixée. Le niveau d'abstraction des caractéristiques est bas quand elles sont proches du pixels et de l'image et de plus haut niveau quand elles s'en éloignent.

Les caractéristiques proches du pixel sont extraites par des méthodes utilisant de l'analyse d'image, des histogrammes ou des transformations géométriques. Il y en a une très variété nombre dans la littérature et il est difficile de toutes les répertorier. On citera parmi ces caractéristiques utilisées dans [Knerr et al., 1998, Graves et al., 2009, Morillot et al., 2013] :

- La densité des pixels noirs ;
- La moyenne des valeurs des pixels gris ;
- La taille et la largeur des graphèmes (morceau de caractères) ;
- La position du centre de gravité ;
- Les profils des pixels vus des 4 côtés ;
- Les transitions dans les 4 directions (passage pixel noir à blanc et vice versa) ;
- La densité de pixels suivant les directions ;
- Le compte des pixels par colonne et par ligne (analyse "run-length" en anglais).

Par la suite, des caractéristiques de plus haut niveau ont été utilisées par la suite. Celles-ci facilitent l'apprentissage des classifieurs en intégrant des niveaux d'abstraction supérieurs dans leur extraction. Elles proviennent pour la plupart de la vision par ordinateur et de la robotique. Parmi ces caractéristiques, on peut citer notamment :

- Les histogrammes de gradients orientés (HOG) [Dalal and Triggs, 2005] ;
- Les caractéristiques robustes accélérées (SURF) [Bay et al., 2006] ;
- ORB pour "Oriented FAST and Rotated BRIEF" [Rublee et al., 2011].

Certaines de ces caractéristiques ont notamment été utilisées avec succès dans [Mioulet et al., 2015] comme HOG et ORB.

Ces nombreuses caractéristiques ont prouvé leur efficacité pour la reconnaissance de l'écriture cursive. Cependant, aujourd'hui, avec la forte influence des algorithmes d'apprentissage profond, nous avons pu voir que des systèmes de reconnaissance suffisamment forts comme les MDLSTM permettent d'analyser directement les pixels d'une image [Graves and Schmidhuber, 2009, Bluche et al., 2014a]. Plus récemment, ce sont les propriétés des réseaux convolutifs [Poznanski and Wolf, 2016] et des réseaux hybrides CNN

/ LSTM [Voigtlaender et al., 2016] qui ont été utilisées afin d'exploiter les pixels.

Nous nous intéressons maintenant à ces systèmes de reconnaissance de caractères utilisant des caractéristiques ou directement les pixels de l'image.

2.4 Reconnaissance de caractères

Pour reconnaître des lignes de textes, les premières méthodes se sont appuyées sur une segmentation en mots [Seni and Cohen, 1994, Louloudis et al., 2009], puis en caractères. La segmentation des lignes en mots facilitent la reconnaissance car les méthodes peuvent s'appuyer sur un lexique de mots. Il existe deux types de méthodes de reconnaissance de caractères : les méthodes fondées sur la segmentation en caractères ou graphèmes et les méthodes sans segmentation.

2.4.1 Reconnaissance de graphèmes ou de caractères segmentés

La première famille de méthodes de reconnaissance utilise une reconnaissance fondée sur une segmentation en caractères.

La reconnaissance de caractères joue un double rôle dans les méthodes de reconnaissance fondées sur une segmentation car elles reconnaissent à la fois les caractères mais aussi valident les points de segmentation. Les deux grands types d'approches sont d'une part les approches créant plusieurs possibilités de segmentation en caractères en associant des graphèmes, et d'autre part les approches modélisant directement les graphèmes dans la reconnaissance.

Dans les méthodes où plusieurs choix de segmentation sont analysés, les hypothèses sont mises dans un graphe et la reconnaissance permet de choisir le meilleur chemin de segmentation dans ce graphe. Dans [Edelman et al., 1990], les agrégations de graphèmes sont reconnues par calcul de l'alignement le plus proche à des prototypes de caractères, sorte de masque représentant les 26 caractères de l'alphabet latin. D'autres reconnaisseurs sont utilisés comme dans [Favata and Srikantan, 1996] où les auteurs utilisent une méthode de k plus proches voisins. Dans [Bengio et al., 1995, Tay et al., 2001], c'est un réseau de neurones qui est utilisé. Ces méthodes peuvent prendre en compte des contraintes lexicales afin d'améliorer le résultat de reconnaissance.

Les méthodes modélisant directement les graphèmes utilisent quasi exclusivement une représentation à l'aide de modèles de Markov cachés (HMM) [Rabiner, 1989]. Les modèles de Markov modélisent les mots comme une séquence de graphèmes tous reliés entre eux. Ce modèle émet alors une probabilité de caractère pour chaque graphème. Des HMMs discrets ont été utilisés avec des caractéristiques simples dans [El-Yacoubi et al., 1999, Morita et al., 2001]. Dans [Knerr et al., 1998], les HMMs ne modélisent pas les caractères mais directement les mots du lexique des montants de chèque. [Chen et al., 1995] utilisent des HMM continus qui modélisent les relations entre chaque symbole et son état à l'aide de mélanges de gaussiennes. Généralement, ces méthodes utilisent un décodage de Viterbi [Viterbi, 1967] ou l'algorithme de Baum-Welch [Welch, 2003].
Par

L'inconvénient de ces méthodes provient de la recherche de points de segmentation dans une étape de pré-traitement où l'omission d'un point de segmentation engendre une erreur irrécupérable.

2.4.2 Reconnaissance de caractères sans segmentation

Les méthodes de reconnaissance sans segmentation ne cherchent pas à segmenter explicitement les images à reconnaître, même s'il est vrai que la segmentation en caractères peut être un sous produit de ces méthodes une fois la reconnaissance réalisée. Ces méthodes sont devenues très populaires et ont permis d'obtenir des résultats à l'état de l'art dans de nombreuses compétitions. La majorité de ces méthodes s'appuie sur les modèles de Markov cachés (HMM) et les réseaux de neurones.

Le principe des méthodes utilisant des HMMs réside dans l'utilisation d'une fenêtre glissante [Kaltenmeier et al., 1993]. Les vecteurs de caractéristiques extraits avec la fenêtre sont alors donnés en entrée de HMMs qu'ils représentent à l'aide de modèles de mélanges de gaussiennes (GMM pour "Gaussian Mixture Models"). Les HMMs modélisent chacun un caractère et calculent chacun la vraisemblance du caractère, ce qui permet ensuite de modéliser la chaîne de caractères. Il existe de très nombreuses méthodes utilisant les HMM continus ou GMM-HMM ; une revue de ces méthodes est d'ailleurs disponible dans [Plötz and Fink, 2009].

Les modèles de mixtures de gaussiennes peuvent être remplacés par des réseaux de neurones comme les MLP ou les RNN [Senior and Robinson, 1996]. Les méthodes basées sur les réseaux de neurones utilisent une façon similaire de procéder à la reconnaissance par fenêtre glissante. Les colonnes de pixels (cas 1D) ou les patchs de pixels (cas 2D) de l'image sont considérés comme étant des points où le réseau s'applique. À l'image d'une fenêtre glissante, le réseau de neurones s'applique à tous ces points et réalise autour les calculs de vraisemblances des caractères utilisées par le HMM. Les réseaux utilisés peuvent être soit des réseaux convolutifs [Bengio et al., 1995, LeCun et al., 1998] soit des réseaux de neurones récurrents LSTM [Graves et al., 2009, Graves and Schmidhuber, 2009, Voigtlaender et al., 2016]. Dans ces méthodes, il est important d'effondrer la dimension verticale (la hauteur de l'image) à l'aide de couches de pooling pour les CNN afin d'obtenir une séquence 1D. Cette contrainte est imposée par le fait qu'on ne peut émettre de probabilité pour les caractères qu'à une position horizontale de l'image. En effet, il n'y a pas deux caractères l'un au dessus de l'autre. Pour les réseaux MDLSTM nous avons vu la couche d'effondrement qui permet d'effondrer la dimension verticale et de ramener l'image à une séquence. Les réseaux BLSTM [Moulet et al., 2015] fonctionnent comme les HMMs avec l'extraction de vecteurs de caractéristiques à l'aide d'une fenêtre glissante.

Néanmoins, toutes ces méthodes, même les plus performantes utilisant des CNN et MDLSTM [Voigtlaender et al., 2016], n'émettent en général que des hypothèses de caractères. En effet, elles n'intègrent pas de connaissance linguistique permettant d'apporter des informations contextuelles de haut niveau : lexique et modèle de langage permettant de choisir les bonnes hypothèses.

2.5 Reconnaissance dirigée par le lexique

Pour avoir un système de reconnaissance complet avec de bonnes performances, il faut ajouter un traitement linguistique à la reconnaissance de caractères. En effet, lorsqu'on connaît le langage à reconnaître, l'utilisation d'une modélisation linguistique permet de contraindre la reconnaissance aux seuls éléments de la langue et ainsi de corriger des erreurs. Les deux grands types de méthodes sont les méthodes fondées sur l'utilisation d'un vocabulaire ou lexique et les méthodes fondées sur une modélisation de la langue qui nécessitent de modéliser tout d'abord un lexique. Les deux méthodes nécessitent une recherche de la meilleure solution en s'appuyant sur un lexique donné ou modélisé. Ainsi, une séquence de caractères est associée à une probabilité (ou score) pour chaque mot du lexique ou du vocabulaire modélisé par la langue. Nous appelons méthodes dirigées par le lexique tous les systèmes de reconnaissance utilisant des hypothèses de caractères afin de contraindre la sortie à un lexique, ou à une modélisation de la langue. Ces méthodes cherchent le mot dont le score est le plus élevé. Les méthodes similaires à une recherche en faisceau de Viterbi [Fissore et al., 1988] sont les plus utilisées pour réaliser la recherche du mot ou de la phrase de plus grand score (ou probabilité).

2.5.1 Reconnaissance globale de mots

Les méthodes de reconnaissance globale de mots sont des méthodes anciennes. Elles cherchent à reconnaître directement le mot sans chercher à le segmenter en caractères, en s'appuyant généralement sur un petit lexique. L'idée réside dans le fait que la forme ou le profil d'un mot est moins sensible aux variations locales qui peuvent affecter les caractères. Dans [Parisse, 1996], les auteurs proposent d'extraire deux profils de points (haut et bas du mot) représentant la forme du mot et de s'en servir pour classifier directement les mots. D'autres méthodes se basent sur des caractéristiques holistiques [Madhvanath and Govindaraju, 1996] ou des caractéristiques encodant la position relative des boucles, jambages et hampes [Guillevic and Suen, 1998] pour reconnaître les mots en entier. La principale application de ces méthodes de reconnaissance globale est la reconnaissance de montant sur des chèques. En effet, les mots sont des entités indivisibles modélisées et reconnues comme telles. Il faut donc beaucoup d'exemples de chaque mot pour apprendre un classifieur à les distinguer. Il est en pratique difficile, voire impossible, d'obtenir une telle quantité d'exemples couvrant les centaines de milliers de mots d'une langue. C'est la raison pour laquelle ces méthodes se limitent à la reconnaissance de montants de chèques.

Pour de nombreuses applications, le champ lexical à couvrir est trop grand et nécessite une reconnaissance de caractères suivie d'un décodage dirigé par le lexique. On parle de méthodes analytiques en opposition aux méthodes globales.

2.5.2 Méthodes fondées sur un lexique

L'intégration d'un lexique est une méthode importante lors de la reconnaissance qui pose des problèmes liés à son choix et sa taille. En effet, un système de reconnaissance n'est capable de reconnaître que les mots présents dans son lexique. Il faut donc augmen-

ter la taille des lexiques à des millions d'entrées lorsque le langage est peu contraint. Il y a par exemple 336531 mots dans le dictionnaire français Gutenberg et plus de 3 millions de mots sur les pages françaises de Wikipedia et du Wiktionnaire. Cependant, l'augmentation du nombre de mots dans un lexique n'est pas sans conséquence. En effet, plus le lexique est grand et plus le temps de la recherche est important, rendant impossible la recherche du mot de score le plus élevé en un temps raisonnable. À cette problématique s'ajoute un problème de précision car le nombre de mots ayant des écritures proches (par exemple "mange", "Mange", "Monge" et "mangé") augmente. C'est le problème des grands lexiques, dont une revue du problème et solutions a été proposée dans [Koerich et al., 2003]. Nous discutons plus en détails de ce problème dans le chapitre suivant, section 3.2.

En raison du problème des grands lexiques, les méthodes fondées sur un lexique cherchent à trouver un équilibre entre le taux de couverture et le taux de mots hors vocabulaires (OOV pour "Out Of Vocabulary"). Il n'est pas toujours facile d'évaluer ces taux car les éléments à reconnaître sont par nature inconnus. Plus le lexique est grand, plus le taux de couverture augmente et plus le taux d'OOV diminue. Un système est optimal lorsque le taux de couverture est à 100% pour un lexique le plus petit possible. Les systèmes s'appuyant sur un décodage dirigé par le lexique cherchent systématiquement à lier la séquence de caractères au mot du lexique de plus grand score. Contraindre un résultat par des méthodes dirigées par le lexique implique une erreur minimale incompressible sur les mots hors vocabulaires que l'on ne peut pas reconnaître avec cette stratégie.

Il existe de nombreuses modélisations d'un vocabulaire mais la plus utilisée consiste à utiliser un transducteur à états finis ("Finite State Transducer" FST) comme il a été proposé dans [Mohri, 1996]. Le principe d'un transducteur à états finis est de transformer à l'aide d'opérations un ou plusieurs mots d'un alphabet d'entrée en un ou des mots d'un alphabet de sortie. Dans le cas de la modélisation à l'aide de FST, l'alphabet d'entrée est l'ensemble des caractères et l'alphabet de sortie le lexique. Les librairies les plus connues permettant de faire de la reconnaissance dirigée par le lexique sont HTK [Young et al., 2002] et Kaldi [Povey et al., 2011] utilisant des FST pondérés ("Weighted FST" WFST).

L'utilisation des transducteurs pondérés à états finis est motivée par la possibilité d'insérer les probabilités d'un modèle de langage au niveau des opérations. Un modèle de langage ayant appris un lexique, une syntaxe ou une grammaire peut alors se modéliser par un transducteur à états finis. On peut alors modéliser tout le traitement linguistique à l'aide d'un transducteur complet.

2.5.3 Modélisation de la langue

Un modèle de langage est une distribution probabiliste sur des séquences de mots ou de caractères. Les modèles de langage sont utilisés dans de nombreux domaines du traitement de la langue naturelle afin de vérifier la qualité d'une phrase, que ce soit en reconnaissance de l'écriture et de la parole ou en traduction automatique. Les modèles de langage peuvent modéliser soit des séquences de caractères soit des séquences de mots. Un modèle de langage de caractères permet de modéliser des mots hors vocabulaires par des probabilités. Un modèle de langage de mots utilise un lexique et modélise une syntaxe

et une grammaire de la langue. L'objectif de ces modèles est de déterminer la séquence de mots ou de caractères la plus probable. Un modèle de langage évalue la probabilité d'observer la séquence de mots $w_i, ..., w_m$ par l'équation 2.1 suivante :

$$P(w_i, ..., w_m) = \prod_{i=1}^{m} P(w_i | w_1, ..., w_{i-1}) \tag{2.1}$$

Les modèles de langage utilisés sont des modèles statistiques [Rosenfeld, 2000] appris sur des textes (ou corpus) [Marti and Bunke, 2001, Bunke et al., 2004] de la langue permettant d'établir une relation entre les mots ou les caractères. Y compris avec de très grands corpus, il est impossible de modéliser toutes les combinaisons d'enchaînements de mots ou de caractères d'une langue du fait que le nombre de possibilités croît exponentiellement avec la taille du lexique. Plus le lexique est grand, plus la combinatoire d'enchaînements possibles est grande, ce qui rend difficile la modélisation probabiliste.

Pour résoudre ce problème, les n-grams ont été introduits par [Shannon, 1948]. Les n-grams ne modélisent pas une relation sur tous les mots (ou caractères), mais ils modélisent une relation sur les n mots précédents (ou caractères). Pour ce faire, la méthode de n-grams s'appuie sur une hypothèse de Markov à l'ordre n avec une dépendance aux seuls $n-1$ mots précédents ou suivants. On peut alors estimer $P(w_i, ..., w_m)$ par l'équation suivante :

$$P(w_i, ..., w_m) = \prod_{i=1}^{m} P(w_i | w_{i-n+1}, ..., w_{i-1}) \tag{2.2}$$

Lors de la reconnaissance, on cherche alors à trouver la meilleure séquence de mots (ou n-grams) optimisant cette probabilité. Les n-grams peuvent être utilisés au niveau mots, ou au niveau caractères. Lorsque les n-grams sont utilisés sur des caractères, on a alors une décomposition en sous unités lexicales de n caractères des textes. Toutes les sous unités sont gardées par la décomposition. Afin d'estimer la probabilité des n-grams, un histogramme des séquences de mots ou sous-unités lexicales est réalisé sur le corpus. Cependant, cet histogramme donne lieu a un très grand nombre d'observations à 0. Pour résoudre ce problème, il existe plusieurs méthodes de lissage [Chen and Goodman, 1996].

Il existe aussi de nombreuses autres méthodes comme les modèles de langages utilisant des classes de mots [Brown et al., 1992] ou les skip-grams [Guthrie et al., 2006]. Parmi les autres modélisations de la langue, il y a également les réseaux de neurones [Zamora-Martinez et al., 2014]. En effet l'utilisation de corpus plus grands entraîne davantage de mots dans le vocabulaire, ce qui engendre une augmentation exponentielle du nombre de séquences et donc de valeurs nulles qu'il faut gérer par des méthodes de lissage. Or, les réseaux de neurones peuvent modéliser une distribution de probabilités. Un réseau de neurones apprend alors sur des données représentées par une séquence de vecteurs. Les réseaux de neurones récurrents se prêtent particulièrement bien à l'analyse séquentielle et ont donc été utilisés pour modéliser la langue avec succès [Graves et al., 2013] pour la reconnaissance de la parole. Des réseaux de neurones ont aussi été utilisés [Mikolov et al., 2013a, Mikolov et al., 2013b] pour représenter de manière vectorielle une langue avec *word2vec*.

La modélisation de la langue est la dernière étape de la reconnaissance de l'écriture et permet d'améliorer les résultats de reconnaissance. Nous allons à présent voir comment les performances d'un système de reconnaissance sont évaluées.

2.6 Évaluation de la qualité de la reconnaissance de l'écriture

Il est important de pouvoir évaluer les performances d'un système de reconnaissance d'écriture et d'en assurer son bon fonctionnement. C'est valable aussi bien pour la reconnaissance de mots isolés que de lignes de texte. La reconnaissance de caractères isolés peut s'évaluer à l'aide d'une précision. Dans cette thèse, nous ne nous intéressons qu'à la reconnaissance de mots isolés et de lignes de texte, pour lesquelles l'évaluation du taux d'erreur caractère (CER) et du taux d'erreur mot (WER) sont les deux métriques les plus répandues. Dans les deux cas, nous avons une séquence de caractères qu'il faut comparer à une séquence de caractères représentant la vérité terrain (GT).

Pour la reconnaissance de mots isolés, la vérification de l'égalité de chaque séquence de caractères avec sa vérité terrain permet d'obtenir le taux d'erreur mot (c'est une précision). Cependant, la seule évaluation du taux d'erreur mot (WER) n'est pas suffisante pour permettre d'évaluer correctement la qualité de la reconnaissance. En effet, des erreurs de reconnaissance caractères peuvent se produire sans pour autant gêner la lecture du résultat de reconnaissance. En prenant l'exemple de la langue française, on peut citer un "s" pluriel oublié ou ajouté, une accentuation manquante, ou encore un caractère majuscule confondu avec un caractère minuscule. Il est donc intéressant d'évaluer les performances de reconnaissance d'un mot au niveau caractère. La difficulté réside dans le fait qu'un caractère peut être substitué, supprimé ou ajouté.

La même difficulté se pose lorsque l'on veut évaluer les performances de reconnaissance d'une ligne de texte. À ce niveau, il n'est pas adapté d'évaluer une précision car on devrait considérer une égalité parfaite entre la séquence de caractères reconnus et la séquence de caractères de la vérité terrain. Avec un très grand nombre de caractères dans la séquence, il est fréquent d'avoir une erreur sur un caractère, notamment de ponctuation. Une seule erreur caractère induirait alors une erreur de séquence bien que la reconnaissance de la ligne serait parfaitement compréhensible. Pour évaluer le taux de reconnaissance mot, il faut tenir compte du fait qu'un mot peut être ajouté, supprimé ou substitué, de même que pour les caractères et l'évaluation du taux d'erreur caractère (CER).

La méthode la plus utilisée pour calculer le taux d'erreur caractère et le taux d'erreur mot est basée sur des distances d'édition comme la distance de Levenshtein [Levenshtein, 1966]. La distance de Levenshtein compte le nombre d'opérations sur les éléments (caractères ou mots) pour transformer une séquence en une autre. Les trois opérations possibles sont :

- La substitution : un élément est remplacé par un autre ;
- La délétion : un élément est supprimé ;
- L'insertion : un élément est ajouté.

Afin d'obtenir le nombre minimal d'opérations et d'en déduire le taux d'erreur caractère

ou mot, il faut utiliser un algorithme de programmation dynamique. Le taux d'erreur caractère et le taux d'erreur mot se calculent alors comme suit :

$$CER = \frac{nb_{substitution} + nb_{deletion} + nb_{insertion}}{nb_{caracsGT}} \tag{2.3}$$

$$WER = \frac{nb_{substitution} + nb_{deletion} + nb_{insertion}}{nb_{motsGT}} \tag{2.4}$$

Dans cette thèse, nous utilisons principalement ces deux métriques pour évaluer les performances sur différents jeux de données publiques.

2.7 Jeux de données et lexiques

Les jeux de données utilisés dans cette thèse sont issus de la communauté de la reconnaissance de l'écriture manuscrite et ont fait l'objet d'une compétition. Les deux principaux sont les jeux de données RIMES [Augustin et al., 2006] et IAM [Marti and Bunke, 2002]. Nous avons également utilisé le jeu de données Read [Sánchez et al., 2016] lors de notre participation à cette compétition. Dans cette thèse, tous les résultats sont présentés sur les différentes bases de test. Les meilleurs résultats sont sélectionnés sur les bases de validation, sauf mention contraire.

2.7.1 Le jeu de données RIMES

Le jeu de données RIMES [Augustin et al., 2006] est un jeu de données constitué de courriers factices en langue française rédigés par des particuliers adressés à des entreprises ou des administrations. Les participants devaient écrire le courrier en suivant l'un des scénarios de courrier fourni. Puis, le texte a été scanné en niveaux de gris à une résolution de 300dpi. Ces courriers ont comme thématique générale une requête formelle. Il y a 9 thèmes comme par exemple : une demande d'information, une modification de contrat ou de commande, une plainte, une déclaration de dommages,etc. On retrouve alors un champ lexical assez spécialisé avec notamment beaucoup de première personne, des formules de politesse mais aussi des noms propres, des dates, des numéros. L'alphabet utilisé est composé de caractères latins en majuscules et minuscules, ainsi que de chiffres, de caractères accentués et de ponctuation. Deux exemples de courriers sont présentés en figure 2.6. Ces images ne sont pas directement utilisées pour la reconnaissance de l'écriture. Une extraction des mots ou des lignes avec leur transcription est donnée. Dans cette thèse, les bases de mots et de lignes isolées utilisées proviennent du découpage donné pour la compétition RIMES à ICDAR 2011 [Grosicki and El-Abed, 2011].

2.7.1.1 Base de mots isolés

La base de mots isolés a été construite en deux temps. Une première base a été présentée lors de la compétition ICDAR 2009 [Grosicki and El Abed, 2009], puis celle-ci a été enrichie à l'occasion de la compétition ICDAR 2011 [Grosicki and El-Abed, 2011].

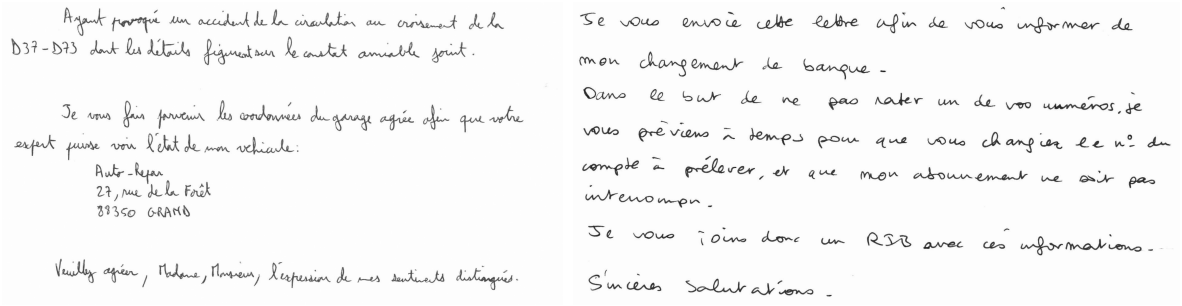


FIGURE 2.6 – Exemples de courriers issus de la base RIMES.

Dans cette thèse, nous utilisons la base RIMES dans sa dernière version. Des exemples de mots issus de cette base sont présentés figure 2.7. Cette base est divisée en trois parties : apprentissage (51737 images), validation (7464 images) et test (7776 images). Sur cette base, deux tâches ont été proposées : la tâche WR2 où le lexique est composé des 1692 mots de la base de test, et la tâche WR3 où le lexique contient les 5744 mots de la base.

FIGURE 2.7 – Exemples de mots isolés issus de la base RIMES.

2.7.1.2 Base de lignes isolées

La base de lignes isolées n'est présentée qu'à partir de la compétition ICDAR 2011 [Grosicki and El-Abed, 2011]. Des exemples de lignes sont présentés figure 2.8. La base est composée de 11328 lignes en apprentissage et 778 en test. De cette base d'apprentissage, nous avons tiré 1426 lignes au hasard pour constituer notre base de validation. Le lexique utilisé avec la base ligne de RIMES est l'ensemble des mots de la base d'apprentissage, soit 8578 mots.

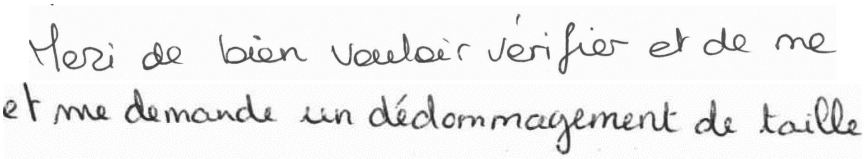


FIGURE 2.8 – Exemples de lignes issues de la base RIMES.

2.7.2 Le jeu de données IAM

Le jeu de données IAM [Marti and Bunke, 2002] est un jeu de données constitué de phrases en langue anglaise tirées du LOB corpus [Johansson et al., 1978]. Les participants ont recopié un passage de texte de ce corpus dans un formulaire sans contrainte. Puis, les

formulaires ont été scannés à une résolution de 300dpi en niveaux de gris. Il y a 15 catégories de documents représentées dans le jeu de données IAM et le style est très littéraire. On y retrouve notamment des articles de presse ainsi que des fictions. Le champ lexical et les phrases obéissent donc à certaines règles linguistiques. L'alphabet utilisé est composé de caractères latins en majuscules et minuscules, ainsi que de chiffres et de caractères de ponctuation. Deux exemples de formulaires sont présentés figure 2.9. De façon similaire à RIMES, ces images ne sont pas directement utilisées pour la reconnaissance de l'écriture. À l'inverse, on utilise des mots isolés ou des lignes extraites dont la transcription est fournie. Dans cette thèse, les bases de mots et de lignes isolées IAM sont utilisées. Cependant, le découpage en lignes fournis dans la dernière version de la base ne correspond pas à celui utilisé par la communauté (comme dans [Pham et al., 2014, Voigtlaender et al., 2016]). C'est pourquoi, nous utilisons alors les deux.

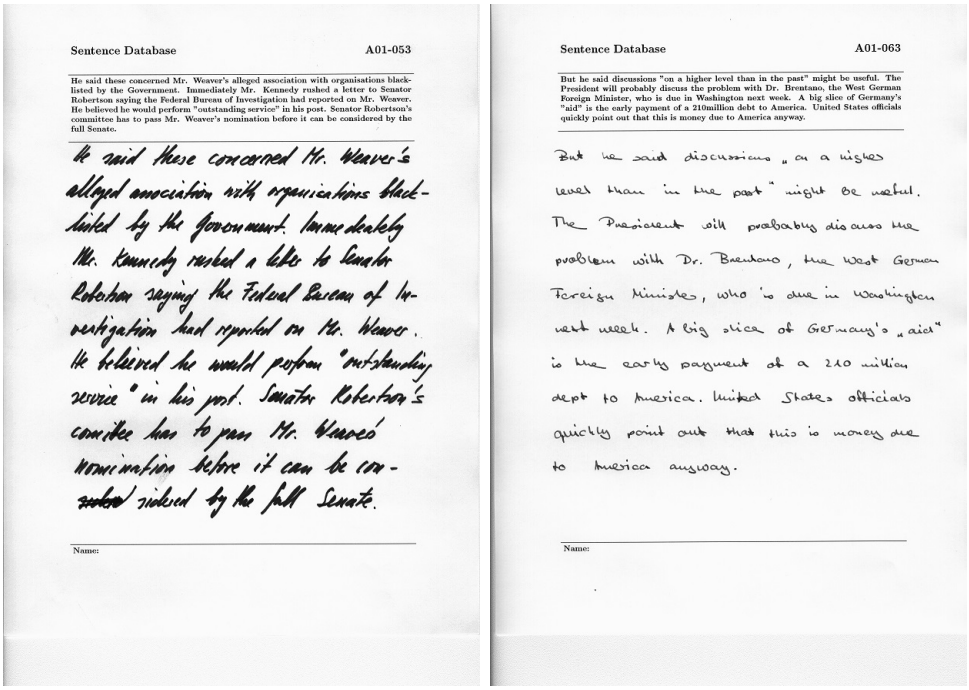


FIGURE 2.9 – Exemples de documents issus de la base IAM.

2.7.2.1 Base de mots isolés

La base de mots isolés IAM sert de référence depuis de nombreuses années. Des exemples extraits de la base sont donnés figure 2.10. Elle est composée de plus de 115 000 mots. Cependant, certains mots sont identifiés comme contenant une erreur de segmentation. Nous écartons donc ces mots lors de la création du découpage fourni en base d'apprentissage, de validation et de test. Le lexique comprend tous les mots du jeu de données pour un total de 12202 mots.

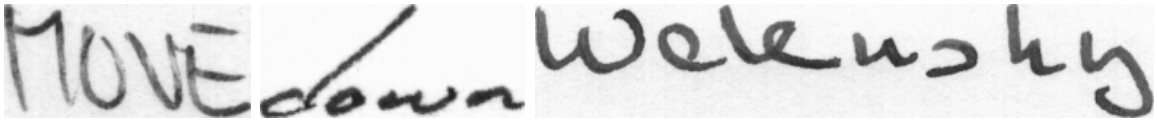


FIGURE 2.10 – Exemples de mots isolés issus de la base IAM.

2.7.2.2 Base de lignes isolées

La base de lignes isolées possède plusieurs séparations possibles : une officielle [Marti and Bunke, 2002] et une utilisée par la communauté jusqu'à présent(comme dans [Pham et al., 2014, Voigtlaender et al., 2016]). Des exemples de lignes extraites sont données figure figure 2.11. Afin d'effectuer une comparaison, nous utilisons le découpage de la communauté qui est constitué de 6482 lignes en apprentissage, 976 en validation et 2915 en test. Le lexique est alors composé de tous les mots de la base d'apprentissage et de validation, soit 13373 mots.

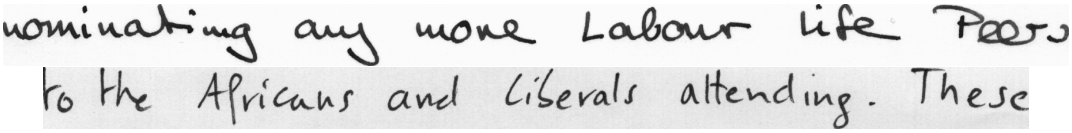


FIGURE 2.11 – Exemples de lignes issues de la base IAM.

2.7.3 Le jeu de données READ 2016

Le jeu de données READ 2016 est un jeu de données préparé pour une compétition à l'occasion de la conférence ICFHR 2016 par le centre de recherche PRHLT de l'université de Valence [Sánchez et al., 2016]. Ce jeu de données comprend des images de documents anciens de la base *Ratsprotokolle* composés de comptes rendus de conseils municipaux tenus entre 1470 et 1805. Les difficultés de ce jeu de données résident à la fois dans l'image avec du texte penché et qui apparaît par transparence mais aussi avec le champ lexical ; la langue étant de l'allemand ancien. L'alphabet utilisé est composé de caractères allemands en majuscules et minuscules, ainsi que de chiffres, de caractères accentués et de caractère spéciaux. Le nombre de scripteurs n'est pas connu. Cependant, les images fournies présentent des documents d'une même année et nous supposons qu'il n'y a qu'un seul scripteur. Des exemples des images du jeu de données sont présentées figure 2.13.

Les images sont en couleurs et la résolution est de 300dpi. Des fichiers XML au format page sont fournis avec la segmentation en lignes ainsi que la transcription pour les données d'apprentissage et de validation. Des exemples de lignes sont donnés figure 2.12. La base est découpée en 350 pages d'apprentissage soit 8367 lignes, 50 pages de validation et de test, soit respectivement 1043 et 1140 lignes. Le lexique extrait de la base d'apprentissage et de validation contient 8511 mots et le taux de mots hors vocabulaire est de 6.20% sur la base de test.

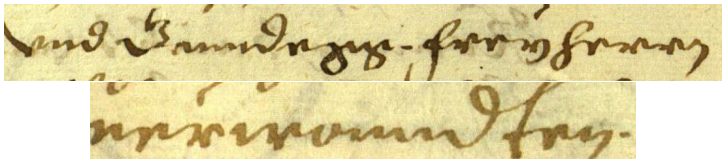


FIGURE 2.12 – Exemples de lignes issues de la base READ 2016.

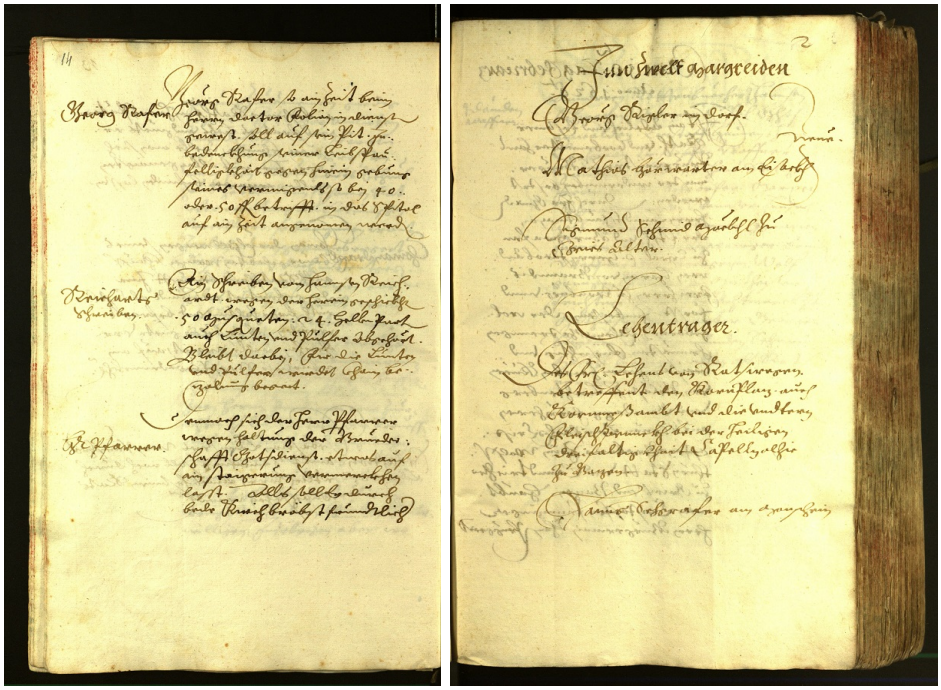


FIGURE 2.13 – Exemples de documents issus de la base READ 2016.

2.8 Conclusion

Dans ce chapitre, nous avons brièvement présenté les méthodes principales et problématiques qui ont marqué l'histoire de la reconnaissance de l'écriture manuscrite au cours de ces 40 dernières années, l'évaluation des méthodes de reconnaissance ainsi que les jeux de données publiques. Les méthodes de reconnaissance de l'écriture ont été utilisées dans de nombreux domaines, comme la lecture automatique de chèques, d'adresses, ou encore la numérisation de documents et leur indexation. Les méthodes avec segmentation ont pendant longtemps été préférées, puis elles ont laissé place à des méthodes sans segmentation. Aujourd'hui, il est même envisagé de reconnaître l'écriture d'un document sans le segmenter en ligne à l'aide de modèles à attention [Bluche et al., 2016]. Néanmoins, les méthodes avec segmentation en lignes basées sur des architectures profondes comme les CNN-LSTM [Voigtlaender et al., 2016] couplées avec un décodage dirigé par le lexique avec modélisation de la langue constituent l'état de l'art.

En dépit des progrès réalisés ces dernières années en reconnaissance de l'écriture grâce aux réseaux de neurones récurrents, l'architecture et la stratégie de reconnaissance des systèmes n'a pas évolué. Les systèmes de reconnaissance procèdent toujours par une re-

connaissance dirigée par le lexique et la langue. En contraignant la reconnaissance par une méthode de décodage dirigé par le lexique, on crée des problèmes liés à la taille du lexique et aux temps de décodage. Si nous ne sommes pas entrés dans les détails de cette problématique c'est parce qu'elle constitue l'objet du chapitre suivant (3). Dans ce 3ème chapitre, nous proposons un nouveau paradigme de reconnaissance de l'écriture utilisant les lexiques.

Deuxième partie

Contributions

Chapitre 3

Vérification lexicale et cohorte de réseaux LSTM

Contents

3.1 Introduction

Nous avons vu précédemment dans l'état de l'art que la reconnaissance de l'écriture se décompose traditionnellement en deux étapes [Plamondon and Srihari, 2000] : la reconnaissance optique de caractères et les traitements linguistiques. La reconnaissance optique de caractères est difficile du fait de la variabilité des formes. En effet, chaque individu a son propre style d'écriture. C'est pourquoi, la seule utilisation de réseaux LSTM à l'état de l'art pour reconnaître les caractères [Graves and Schmidhuber, 2009] n'est pas suffisante. En effet, un nombre non négligeable d'erreurs subsiste (taux d'erreur caractère brut : environ 10% sur la base mot de RIMES). Le traitement linguistique a pour objectif de combiner des hypothèses de caractères afin de déterminer la séquence la plus probable en accord avec des règles linguistiques. Il y a deux types de connaissances linguistiques : les lexiques et les modèles de langages. Un modèle de langage est une modélisation probabiliste d'une langue qui fournit généralement une vraisemblance au niveau mot ou phrase permettant de ré-estimer les probabilités des hypothèses de reconnaissance données par le modèle optique. La question du corpus à utiliser pour apprendre le modèle de langage se pose alors. Les méthodes dirigées par le lexique ont pour objectif de reconnaître les mots grâce à l'utilisation d'un lexique. Ces méthodes cherchent le mot du lexique correspondant le mieux à la concaténation des hypothèses de caractères. Pour la reconnaissance d'écriture manuscrite, on cherche à aligner les hypothèses de caractères aux mots du lexique. Il n'y actuellement aucune alternative efficace à l'utilisation de méthodes dirigées par le lexique. La question du lexique à utiliser est difficile à appréhender et affecte directement les performances de reconnaissance. Un lexique trop petit ne peut pas couvrir suffisamment les données à reconnaître, manquant ainsi des solutions. Cependant l'utilisation d'un lexique trop grand (1000 mots ou plus) conduit à des temps de calcul plus élevés et à une baisse de la précision [Koerich et al., 2003]. À notre connaissance, le plus grand lexique ayant été utilisé dans la littérature est composé de 200k mots [Hamdani et al., 2014] (60k mots pour [Bluche et al., 2014a]). Cependant, il y subsiste toujours des mots hors vocabulaire (entités nommées, nombres, etc).

Dans le chapitre 2, nous avons vu que les principaux progrès dans le domaine de la reconnaissance de l'écriture sont dus aux avancées réalisées dans le domaine de l'apprentissage profond [LeCun et al., 2015] avec notamment les réseaux récurrents à cellules de mémoire à long et court terme (LSTM) [Hochreiter and Schmidhuber, 1997]. Les performances de reconnaissance atteintes par les systèmes complets incluant réseaux LSTM et ressources linguistiques s'expliquent par les performances brutes élevées du modèle optique, c'est-à-dire sans utiliser de ressources linguistiques supplémentaires [Bluche, 2015]. Par exemple, les performances brutes d'un réseaux BLSTM seul sur la base RIMES sont de l'ordre de 35-40% de taux d'erreur mot comme rapporté dans [Pham et al., 2014] . Nous pensons que ces performances brutes des réseaux LSTM sont sous-exploitées et donnent des idées sur une exploitation différente de tels réseaux. Ils ne seraient alors plus utilisés comme de simples modèles de reconnaissance de caractères imbriqués dans une approche dirigée par le lexique, comme c'est le cas dans la plupart des études récemment rapportées dans la littérature (Cf. chapitre 2).

En rupture avec l'utilisation standard des réseaux LSTM comme un simple classifieur

couplé à un processus de décodage dirigé par le lexique, ce chapitre propose un nouveau paradigme de reconnaissance améliorant les performances à l'état de l'art pour la reconnaissance de mots isolés.

Ce nouveau paradigme s'appuie sur la combinaison en cascade de nombreux classifieurs de mots isolés et d'une règle de décision au niveau mot. Nous introduisons une combinaison de classifieurs utilisant une architecture en cascade. Cette combinaison en cascade est basée sur deux points clés : i) un ensemble de classifieurs complémentaires, et ii) un processus de décision rapide. Le premier point est résolu en introduisant une nouvelle stratégie très efficace permettant de générer des centaines de reconnaisseurs complémentaires en un temps d'apprentissage très raisonnable. En se basant sur les derniers résultats théoriques en terme d'apprentissage profond [Choromanska et al., 2015], nous proposons de générer de multiples réseaux complémentaires lors d'un seul apprentissage. Nous exploitons ces résultats théoriques afin de générer à l'aide d'un apprentissage unique des centaines de réseaux LSTM complémentaires. Nous nommons cet ensemble de réseaux par le terme de **cohorte**. Le second point concerne le processus de décision pour lequel nous proposons la **vérification lexicale**. Elle consiste à accepter un mot si l'hypothèse appartient au lexique et à le rejeter dans le cas contraire. L'idée sous-jacente est qu'il est très peu probable qu'une hypothèse erronée de mot appartienne au lexique. L'avantage majeur de cette stratégie est qu'elle décide extrêmement rapidement, particulièrement quand on la compare à un processus de décodage dirigé par le lexique telle qu'une recherche en faisceau de Viterbi [Fissore et al., 1988]. Nous montrons par ailleurs que la combinaison proposée atteint des performances élevées quelle que soit la taille du lexique. En conséquence, cette approche n'a pas de limitation de taille de lexique, comme nous le montrons dans ce chapitre, avec les résultats obtenus en utilisant un lexique de plus de 3 millions de mots.

Ce chapitre est organisé comme suit : Nous nous intéressons d'abord aux limitations des méthodes dirigées par le lexique et des grands vocabulaires qui ont conduit à ces recherches. Puis, nous présentons notre approche de cascade de réseaux LSTM avec vérification lexicale. Enfin, l'implémentation de notre méthode et les résultats que nous obtenons sont présentés et discutés avant de conclure.

3.2 Méthodes dirigées par le lexique et le problème des grands vocabulaires

Nous avons vu à travers le chapitre 2 que la majorité des systèmes procèdent de manière identique en utilisant des décodages dirigés par le lexique. Nous rappelons que nous appelons méthodes dirigées par le lexique tous les systèmes de reconnaissance utilisant des hypothèses de caractères afin de contraindre la sortie à un lexique, ou à une modélisation de la langue. Ces systèmes reportent les décisions de classification de caractères au niveau de la phrase afin de trouver la séquence de mots appartenant au lexique la plus probable, en utilisant un algorithme de complexité au moins égale celle d'un algorithme de programmation dynamique. Ce schéma de reconnaissance est présenté figure 3.1a. Si les approches dirigées par le lexique permettent de corriger les erreurs de reconnaissance de

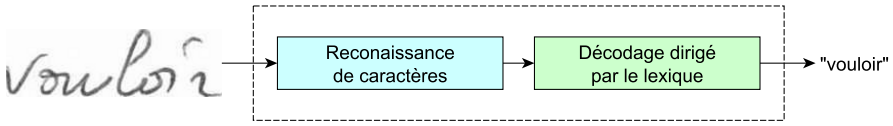
caractères, elles nécessitent la constitution de grands lexiques afin d'avoir une couverture lexicale suffisante et de minimiser les mots hors vocabulaire (OOV). Cependant, l'utilisation d'un grand lexique nécessite un élagage de celui-ci lors de la phase de reconnaissance afin d'avoir des temps de calculs raisonnables. Dans [Grosicki and El Abed, 2009], les auteurs présentent les résultats obtenus pour la compétition RIMES 2009 de reconnaissance de mots isolés, dans laquelle la majorité des participants ont implémenté des approches basées sur les HMM. On peut y observer une nette différence des taux de reconnaissance entre un petit, un moyen et un grand lexique allant jusqu'à 5334 mots. La taille du lexique joue également un rôle majeur dans des applications réelles où celui-ci est lié à une langue et peut contenir des centaines de milliers de mots. Dans des cas spécifiques, l'utilisation de telles ressources linguistiques pour améliorer la reconnaissance n'est plus compatible avec des contraintes de traitement en temps réel.

Le problème des grands vocabulaires a été décrit et identifié dans [Koerich et al., 2003], où un lexique de taille 1000 est déjà considéré comme grand. Plus récemment, dans [Pham et al., 2014] les auteurs utilisent des lexiques de 50k, 12k et 95k mots pour les jeux de données IAM, RIMES et OpenHaRT respectivement. Toutefois, aucun de ces lexiques ne couvre entièrement leur base respective. Les plus grand lexiques ayant été utilisés dans la littérature sont composés de 200k mots [Hamdani et al., 2014] et 60k mots pour [Bluche et al., 2014a]. Néanmoins, des problèmes persistent car il y a toujours des mots hors vocabulaire. De plus des applications plus générales et pratiques nécessitent l'utilisation de lexiques moins contraints. Ces lexiques peuvent contenir des centaines de milliers de mots (e.g. le dictionnaire français Gutenberg contient 336k mots) afin d'atteindre des taux de couverture acceptables, sans pour autant couvrir les entités nommées et les nombres.

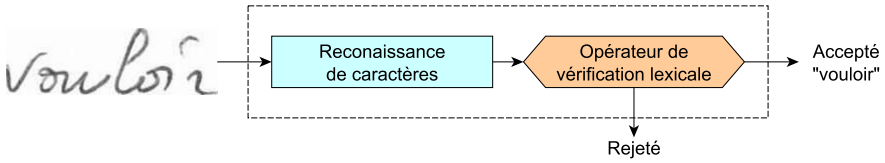
Pour dépasser ces limitations, des solutions ont été étudiées :

- L'élagage supprime les entrées d'un lexique qui correspondent le moins à une image à reconnaître et permet de réduire la complexité de l'exploration des solutions [Madhvanath and Govindaraju, 1996, Madhvanath et al., 2001, Gilloux, 1998]. Cependant, l'élagage peut être excessif et induire une augmentation des erreurs.
- Certaines méthodes [Brakensiek et al., 2002, Bharath and Madhvanath, 2012] permettent un décodage sans lexique grâce à des modèles de Markov cachés avec d'autres techniques comme les sacs de symboles. Toutefois, ces méthodes n'égalent pas les performances des méthodes dirigées par le lexique. Signalons que de telles approches ont aussi été utilisées pour la reconnaissance de champs numériques sans lexique [Shridhar et al., 1997, Chatelain et al., 2006a, Chatelain et al., 2006b] bien que les chiffres soient plus simples à reconnaître.
- Enfin, dans [Brakensiek et al., 2002, Hamdani et al., 2013, Kozielski et al., 2013b, Poznanski and Wolf, 2016, Swaileh et al., 2016], les auteurs proposent une décomposition en préfixes et suffixes de mots du lexique puis un modèle de n-gram de ces unités permettant de représenter des lexiques ouverts. Ces méthodes sont basées sur des statistiques extraites de corpus d'apprentissage. Les modèles n-grams atteignent l'état de l'art pour des tâches ayant des mots hors lexiques [Hamdani et al., 2013, Kozielski et al., 2013b].

Comme nous venons de le voir, la problématique des grands lexiques est toujours ouverte. Nous cherchons donc à traiter ce problème en proposant un nouveau paradigme



(a) Le paradigme de reconnaissance de l'écriture classique



(b) Notre nouveau paradigme de reconnaissance de l'écriture

FIGURE 3.1 – Paradigmes de reconnaissance de l'écriture.

pour la reconnaissance de mots isolés qui ne s'appuierait plus sur le paradigme actuel de reconnaissance dirigée par le lexique comme il est illustré sur la figure 3.1a. Ce nouveau paradigme de reconnaissance de l'écriture s'inscrit comme une alternative fiable à la reconnaissance dirigée par le lexique. Ce paradigme repose sur le principe de vérification lexicale comme montré figure 3.1b.

3.3 Cascade, vérification lexicale et cohorte de réseaux LSTM

Le paradigme proposé fait apparaître des rejets qui n'existaient pas auparavant. Cette particularité permet un traitement séquentiel des images grâce à un mécanisme de cascade qui combine des réseaux de neurones complémentaires grâce au mécanisme fiable de rejet basé sur une vérification lexicale. Dans cette section, nous décrivons d'abord le principe de cascade et l'architecture que nous mettons en place avec le mécanisme principal de rejet : la vérification lexicale. Enfin, nous décrivons la génération efficace de centaines de réseaux complémentaires.

3.3.1 Cascade de réseaux récurrents pour la reconnaissance de mots isolés

Avant de présenter notre schéma de cascade, une revue de la littérature des systèmes de cascade est présentée.

3.3.1.1 La cascade dans la littérature

La cascade de classifieurs est une méthode de combinaison de classifieurs qui combine séquentiellement les décisions prises par les classifieurs en exploitant leur complémentarité, afin d'affiner progressivement la décision de reconnaissance.

La contribution la plus célèbre est celle de Viola et Jones [Viola and Jones, 2001]. Les auteurs décrivent un processus en cascade pour la détection de visage. Ce processus de cascade est basé sur un grand ensemble de classifieurs faibles, permettant un processus de décision rapide tout en introduisant des étages de décisions fiables. Bien que les performances de chaque classifieur soient faibles, ils sont associés à un étage de décision à forte confiance qui accepte ou rejette l'hypothèse. Les performances des classifieurs de la cascade augmentent progressivement avec la profondeur. L'objectif est, au début, de rejeter rapidement et de manière fiable des résultats simples. Si un objet n'est pas rejeté, il est alors passé au classifieur suivant. Chaque classifieur est appris à partir des éléments non traités par le classifieur précédent, ce qui rend l'apprentissage des classifieurs long et difficile. La cascade est profonde car il y a 32 étages de classification dans la proposition originale.

Dans [Zhang, 2013] et [Zhang et al., 2007], un schéma différent de cascade est proposé, où sont combinés des classifieurs forts ayant des architectures et des caractéristiques d'entrée différentes. Ici, ce sont les objets rejetés qui sont passés au classifieur suivant, contrairement au schéma précédent. Chaque classifieur reconnaît un nombre important d'objets avec un faible taux d'erreur, tout en s'appuyant sur un étage de décision permettant le transfert des rejets au classifieur suivant. Ces deux méthodes s'appuient sur un très petit nombre de décision de classification. La profondeur de la cascade utilisée par ces deux systèmes se limite à une combinaison de deux classifieurs.

Les deux schémas, bien qu'utilisant des classifieurs différents et ayant une profondeur différente, présentent un point commun : la présence d'un étage de décision introduit après chaque classifieur. Dans les deux cas, la puissance de la combinaison en cascade vient de la fiabilité de cet étage décisionnel mais aussi de la complémentarité des réseaux utilisés.

3.3.1.2 Schéma de cascade proposé

Nous venons de voir qu'il existe deux schémas de cascade qui combinent : i) soit des classifieurs faibles et la cascade est profonde ii) soit des classifieurs forts et la cascade est peu profonde. Dans ce chapitre, nous proposons une cascade de classifieurs forts qui est profonde où nous combinons des centaines de réseaux LSTM. Dans [Viola and Jones, 2001], la cascade combine de nombreux classifieurs faibles qui sont forts dans le rejet tandis que notre schéma combine des classifieurs forts qui sont forts dans l'acceptation. L'objectif de "Viola et jones" est de rejeter au plus vite, le notre est de décider au plus vite. Pour construire la cascade, nous enchaînons des réseaux LSTM chacun suivi d'un opérateur de vérification lexicale. Les rejets de l'opérateur de vérification lexicale à un niveau de la cascade servent à alimenter le réseau LSTM au niveau suivant. Lorsqu'une hypothèse est acceptée par l'opérateur de vérification lexicale, le processus s'arrête permettant ainsi de ne pas solliciter tous les classifieurs dans la plupart des cas. Le schéma d'une telle cascade est présenté figure 3.2. Le mécanisme de décision est essentiel aux performances de la cascade et permet aussi d'améliorer significativement la rapidité du processus quand un grand nombre de classifieurs est utilisé.

Le deuxième élément le plus important de la cascade est la complémentarité des classifieurs. En effet, des classifieurs adoptant un comportement similaire ne permettent pas d'affiner la décision à travers la cascade. Nous proposons dans ce chapitre une nouvelle

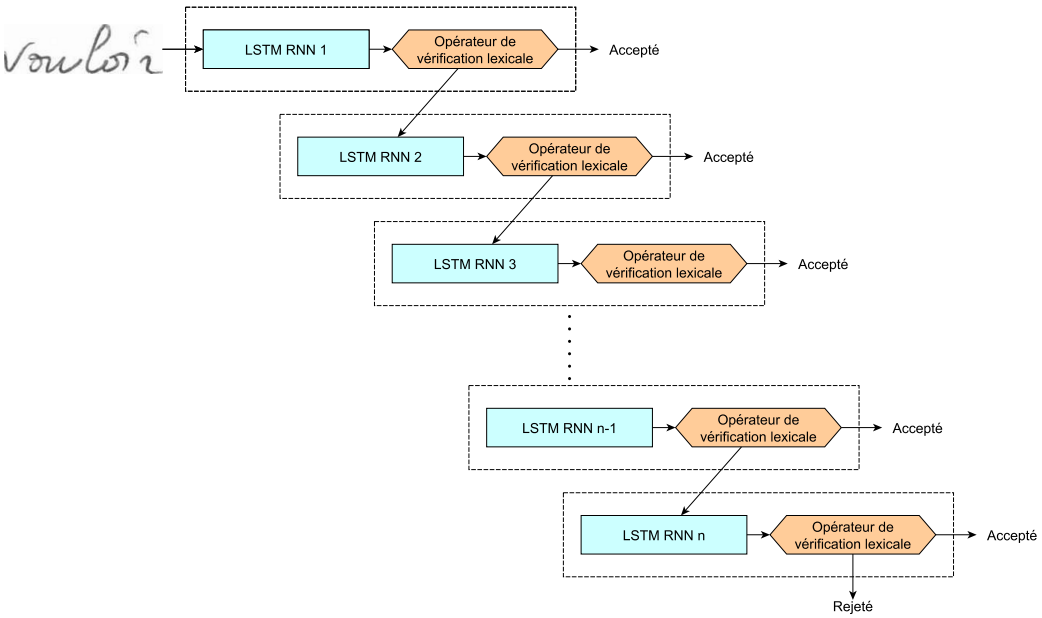


FIGURE 3.2 – La cascade de BLSTM proposée. Chaque réseau traite les rejets de l'étage précédent.

méthode permettant d'obtenir des centaines de réseaux complémentaires lors de l'apprentissage d'un unique réseau récurrent, en s'appuyant sur les propriétés des réseaux de neurones profonds. Les classifieurs sont ordonnés dans l'ordre croissant du taux d'erreur (c'est-à-dire par fiabilité décroissante) afin de diminuer les erreurs au début qui sont ensuite irrécupérables lorsque la reconnaissance est validée.

Nous discutons de cette nouvelle méthode de génération de réseaux complémentaires dans la sous-section 3.3.3. Mais, dans un premier temps, nous étudions les propriétés de l'opérateur de vérification lexicale.

3.3.2 Opérateur de vérification lexicale

L'étage de décision est composé d'une règle de vérification lexicale qui consiste à accepter une hypothèse de chaîne de caractères si elle appartient au lexique. Dans le cas contraire, elle est rejetée. Une règle de décision aussi simple ne peut servir d'étage de décision dans une cascade qu'à condition qu'elle donne des décisions fiables avec un taux de fausse acceptation suffisamment faible.

Une fausse acception de la règle de vérification lexicale se produit lorsqu'une mauvaise séquence d'hypothèses de caractères correspond à un mot du lexique. La probabilité de fausse acceptation P_{FA} pour un reconnaisseur donné s'exprime à l'aide de l'équation 3.1 où W est une hypothèse de mot, L un lexique, et *Reco* un reconnaisseur. "W **est** erronée" est *vrai* lorsque l'hypothèse de mot W est erronée. $W \in L$ est *vrai* lorsque l'hypothèse de mot appartient au lexique.

$$P_{FA} = P(W \textbf{ est } \text{erronée } \wedge W \in L \mid Reco) \tag{3.1}$$

La figure 3.3 montre l'estimation de la probabilité P_{FA} d'un seul reconnaisseur en fonction de la longueur des mots n. La probabilité a été estimée en prenant les résultats d'un réseau LSTM appris sur la base RIMES mot. On peut observer que la probabilité P_{FA} (courbe bleue) est relativement élevée pour les mots courts (moins de 4 caractères) avec plus de 2% de fausse acceptation. En conséquence, on peut conclure que la règle de vérification lexicale n'est pas assez fiable dans ce cas. Cependant, nous voyons que pour les mots longs, cette probabilité diminue rapidement pour atteindre une valeur qui semble acceptable. Pour affaiblir la probabilité d'erreurs sur les mots courts, nous introduisons le Nombre Minimal d'Accord de Décision (NMAD), permettant ainsi de réduire drastiquement la probabilité de fausse acceptation à un niveau de confiance très acceptable. Cette valeur NMAD est le nombre minimal de classifieurs ayant pris la même décision parmi tous les classifieurs ayant déjà donné une décision dans la cascade. Comme on peut l'observer sur les courbes cyan et magenta de la figure 3.3, l'utilisation d'un NMAD de 2 ou 3 permet de réduire fortement la probabilité estimée (en dessous d'un demi pour cent) et ce même pour les mots courts. Dans ce cas, l'acceptation d'une hypothèse de reconnaissance par un classifieur ne met pas fin au processus de cascade.

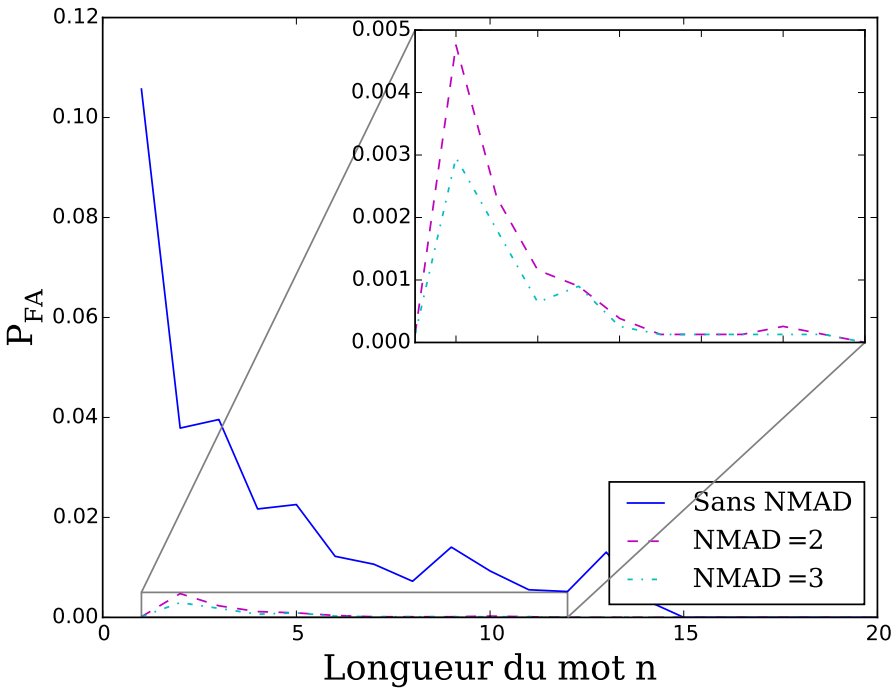


FIGURE 3.3 – P_{FA} estimé sur la longueur de mot n pour un réseau LSTM donné entraîné sur la base RIMES avec et sans Nombre Minimum d'Accord de Décision. La probabilité estimée décroît significativement avec le NMAD.

Analysons maintenant l'effet de la vérification lexicale seule (NMAD = 0) sur une tâche de reconnaissance de mots. À cet égard, la table 3.1 montre les performances obtenues par un réseau LSTM, avec ou sans la règle de vérification, sur le jeu de données RIMES. On peut voir qu'ajouter la règle de vérification conduit à une réduction drastique du taux d'erreur mot de 93%, passant ainsi de 33.63% à 2.25%. En s'intéressant aux confusions

faites, on remarque que la plupart sont des erreurs de casse, d'accentuation ou de pluriel. Cette expérience préliminaire démontre bien la force d'un réseau LSTM et la capacité de la vérification lexicale.

Réseau	**WRR**	**WER**	**WJR**
BLSTM	66.37	33.63	0
BLSTM + vérification	66.37	**2.25**	31.38

TABLE 3.1 – Taux de reconnaissance mot (WRR), taux d'erreur mot (WER) et taux de rejet mot (WJR) d'un BLSTM sans lexique appris sur RIMES mot, avec et sans la stratégie de vérification lexicale. La plupart des erreurs de reconnaissance sont rejetées grâce à la vérification, réduisant ainsi fortement le taux d'erreur mot.

Nous avons montré que la vérification lexicale permet de réduire significativement les erreurs de reconnaissance, et que la proportion de fausse acceptation peut être maîtrisée avec l'ajout du Nombre Minimum d'Accord de Décision (NMAD). Dans l'architecture de cascade proposée, nous introduisons tout naturellement une règle de décision alliant la vérification lexicale et le NMAD. Cet opérateur de vérification lexicale est une règle fiable et efficace permettant le contrôle des décisions à chaque étage de la cascade. Cependant, les résultats de reconnaissance ne sont pas à l'état de l'art en raison du grand nombre de rejets. La profondeur de la cascade permet ainsi d'agir sur ces résultats. Dans la section suivante, nous montrons comment obtenir facilement des centaines de réseaux LSTM complémentaires pour améliorer les performances du système.

3.3.3 Cohorte de réseaux récurrents

Nous cherchons à générer un grand nombre de classifieurs pour les combiner dans le schéma de cascade. Pour y parvenir, nous avons vu qu'ils doivent être complémentaires. Il y a de nombreuses manières de générer des réseaux LSTM complémentaires. Les plus élémentaires sont l'utilisation de différentes architectures (BLSTM vs MDLSTM ; changer le nombre de couches/de neurones ; etc.) et l'utilisation de caractéristiques d'entrée différentes (pixels, histogramme de gradients, etc.). Cependant, ces différentes stratégies sont limitées par leurs coûts en terme de conception et de temps d'apprentissage.

Des expériences précédentes [Menasri et al., 2012] ont montré que des réseaux LSTM ayant des architectures identiques mais des initialisations aléatoires des poids différentes peuvent être combinées avec succès. Ces réseaux ont des taux de reconnaissance similaires, mais les poids sont différents et ont des propriétés complémentaires. Cependant, l'apprentissage de centaines de réseaux avec des initialisations aléatoires différentes requiert un temps considérable (il faut plusieurs jours pour entraîner un seul réseau sur une machine avec CPU sur la base RIMES).

Pour obtenir en un temps raisonnable de conception et d'apprentissage le maximum de réseaux complémentaires, nous proposons de contrôler l'apprentissage d'un réseau profond. Pour ce faire, nous nous sommes inspirés des travaux de Choromanska et al. dans l'article [Choromanska et al., 2015]. Ces travaux font un parallèle entre la fonction de coût des réseaux de neurones entièrement connectés et les polynômes aléatoires de haut

degré qui ont un nombre important de minimum locaux de magnitude de même ordre. Les auteurs en déduisent que "la descente de gradient stochastique converge vers cette bande de points locaux critiques, et que tous ces points trouvés ici sont des minimums locaux de haute qualité mesurée par l'erreur sur la base de test". Les auteurs concluent également que lors de l'apprentissage d'un réseau de neurones suffisamment profond, atteindre un minimum local est équivalent à atteindre un minimum global. Ils ajoutent qu'il est souvent préférable de procéder ainsi car la recherche à tout prix du minimum global est de plus en plus difficile avec la taille du réseau et mène souvent à un sur-apprentissage. En conséquence, l'exploration de ces minimums locaux lors d'un apprentissage semble être une stratégie efficace pour obtenir un grand ensemble de réseaux complémentaires ayant des performances équivalentes. Il faut donc converger vers cet espace de minimums de haute qualité et l'explorer afin d'obtenir des réseaux complémentaires et bons. Nous appelons l'ensemble des réseaux obtenus par cette stratégie une cohorte et nous utilisons la cohorte ainsi obtenue afin de construire une cascade. Sur la figure 3.4, nous illustrons l'exploration de l'espace et la manière dont nous voulons extraire les réseaux. Nous ne souhaitons pas converger vers un minimum global car comme montré sur la courbe rouge la valeur du critère est identique induisant l'absence de complémentarité. À la place, nous souhaitons explorer la zone de la courbe verte ayant des valeurs de critère différentes et de haute qualité, permettant d'obtenir des réseaux complémentaires et performants.

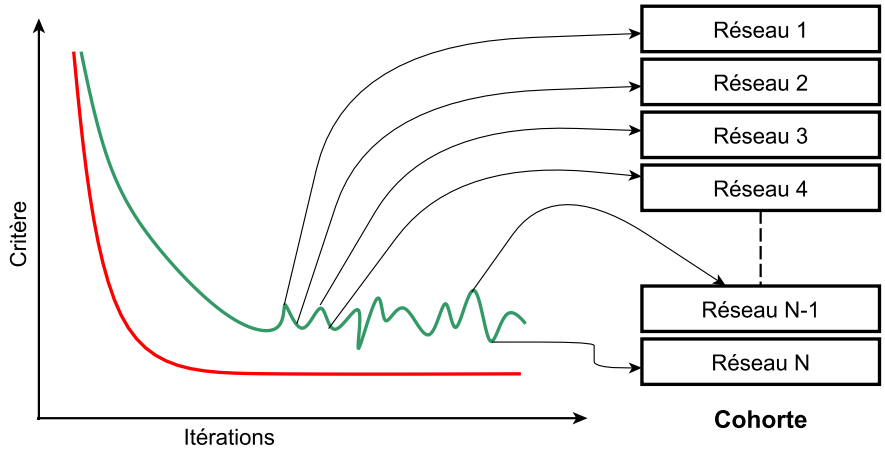


FIGURE 3.4 – Courbe d'apprentissage théorique d'un réseau de neurones que nous désirons (en vert) permettant le processus d'extraction d'une cohorte. La courbe rouge présente un apprentissage qui converge vers un minimum ce que nous ne souhaitons pas.

Cependant, cette stratégie rapide pour obtenir des réseaux complémentaires nécessite de prêter attention aux différents paramètres d'apprentissage afin d'obtenir les propriétés voulues. En effet, lors de l'apprentissage, nous souhaitons explorer l'espace des minimums locaux de haute qualité et éviter de rester piégés dans un minimum local afin d'obtenir des réseaux complémentaires. Une descente de gradient d'ordre 1 est contrôlée par trois paramètres importants : le moment, le mélange aléatoire des données à chaque itération et le taux d'apprentissage. Le moment est le facteur contrôlant la mise à jour des poids et

donc la convergence de la procédure d'apprentissage. Nous utilisons un moment de valeur 0.9 couramment utilisée dans la littérature pour aider l'apprentissage à s'échapper des minimum locaux. Lors de l'apprentissage de réseaux, les données sont mélangées aléatoirement entre chaque itération afin d'obtenir une meilleure convergence et d'éviter les cycles. Puisque les données sont mélangées aléatoirement entre chaque itération, on peut considérer que l'état du réseau (les valeurs de ses poids) a été atteint aléatoirement d'une itération à l'autre. Le dernier mais probablement le plus important des paramètres est le taux d'apprentissage. Le taux d'apprentissage est le coefficient associé à la valeur du gradient, servant ainsi de règle de mise à jour des poids. La figure 3.5 montre l'impact du taux d'apprentissage sur l'apprentissage d'un réseau BLSTM défini dans la sous-section 3.4.1 sur le jeu de données RIMES mot. Un taux d'apprentissage trop élevé ne permet pas la convergence du réseau, comme cela est montré sur la courbe cyan de la figure 3.5 où le réseau n'arrive pas à converger correctement avec un taux d'apprentissage de valeur 10^{-3}. Inversement, un taux d'apprentissage trop petit conduit à modifier faiblement la valeur des poids et ne permet donc pas d'atteindre d'autres minimum locaux. Par conséquent, cet apprentissage ne produira pas de réseaux complémentaires d'une itération à l'autre. Comme montré sur la courbe magenta de la figure 3.5, la diminution du taux d'apprentissage à une valeur de 10^{-5} à l'itération 96 provoque une diminution des fluctuations du taux d'erreur mot. Cette diminution engendre ainsi moins de variations et quasiment plus de complémentarité (ce point est analysé dans la sous-section 3.4.4).(L'itération 96 a été choisie aléatoirement, mais tout autre point aurait convenu.) Le comportement voulu est obtenu pour un taux d'apprentissage d'une valeur de 10^{-4}.

En choisissant ces paramètres, les réseaux extraits à chaque itération d'une unique procédure d'apprentissage peuvent constituer une cohorte. Le rectangle rouge sur la figure 3.5 montre la région où les réseaux peuvent être sélectionnés pour constituer la cohorte. Cette région débute au moment où l'apprentissage a convergé vers un état sans diminution significative du taux d'erreur mot mais avec des fluctuations qui soulignent la présence de différents minimums locaux, donnant ainsi des réseaux aux propriétés locales différentes. Lors de l'apprentissage si le nombre de données est faible et qu'entre deux itérations il n'y a quasiment pas de variations du critère, il faut utiliser un pas de prélèvement des réseaux plus élevé pour lequel la variation du critère est significative. Dans la section suivante, nous montrons expérimentalement que cette stratégie se révèle très efficace pour obtenir des centaines de réseaux complémentaires à l'aide d'un seul apprentissage. Des expérimentations sont analysées dans la sous-section 3.4.4 afin d'évaluer la complémentarité des réseaux d'une cohorte ainsi obtenus.

3.4 Expérimentations et résultats

Nous avons présenté dans les paragraphes précédents un nouvel opérateur de vérification lexicale pouvant servir d'étage décisionnel fiable, ainsi qu'une stratégie permettant d'obtenir des réseaux LSTM complémentaires en un seul apprentissage. En se basant sur ces principes méthodologiques, nous détaillons maintenant l'implémentation de la cascade et présentons les résultats à l'état de l'art que nous avons obtenus.

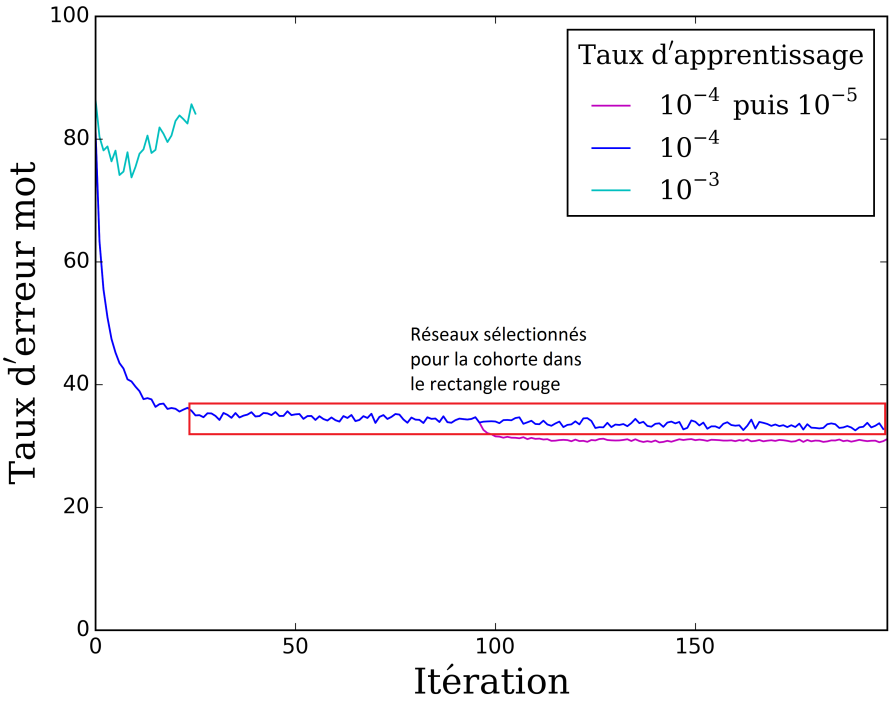


FIGURE 3.5 – Taux d'erreur mot en fonction du nombre d'itérations d'un réseau LSTM appris sur RIMES avec un taux d'apprentissage fixé à 10^{-4} (bleu), à 10^{-4} puis 10^{-5} à l'itération 96 (magenta) et à 10^{-3} (cyan). Sur la courbe bleue, on peut observer des fluctuations locales de l'erreur, qui diminuent en réduisant le taux d'apprentissage (courbe magenta). Sur la courbe cyan, le taux d'apprentissage est trop élevé pour que l'apprentissage puisse converger.

3.4.1 Architectures des réseaux utilisés

Pour les expérimentations, nous utilisons deux architectures de réseaux récurrents à cellules LSTM qui ont démontré leur efficacité.

La première architecture est un réseau LSTM bidirectionnel (BLSTM) similaire à celle utilisée dans [Mioulet et al., 2015]. C'est un BLSTM à trois couches cachées comme montré figure 3.6. Il est composé de deux couches récurrentes, de 70 et 120 cellules LSTM, séparées par une couche de rassemblement de 100 neurones sans biais ayant pour activation la fonction tangente hyperbolique. Le réseau est aussi composé de deux couches de bloc permettant chacune la réduction par deux du nombre de trames en les concaténant deux à deux. La couche de sortie est une couche softmax classique ayant le même nombre de neurones que de caractères à reconnaître. Les caractéristiques d'entrée utilisées sont les histogrammes de gradients orientés (HOG) [Dalal and Triggs, 2005] ; ces derniers ayant démontré leur efficacité pour la reconnaissance de l'écriture [Bideault et al., 2015]. Les images sont normalisées en hauteur à 64 pixels. Une fenêtre glissante de largeur 8 pixels extrait les caractéristiques HOG avec un pas de 1 pixel. Cette architecture a été sélectionnée pour ses caractéristiques équilibrées. Elle présente à la fois des performances légèrement supérieures à l'architecture de référence (présenté dans [Graves and Schmidhuber, 2009])

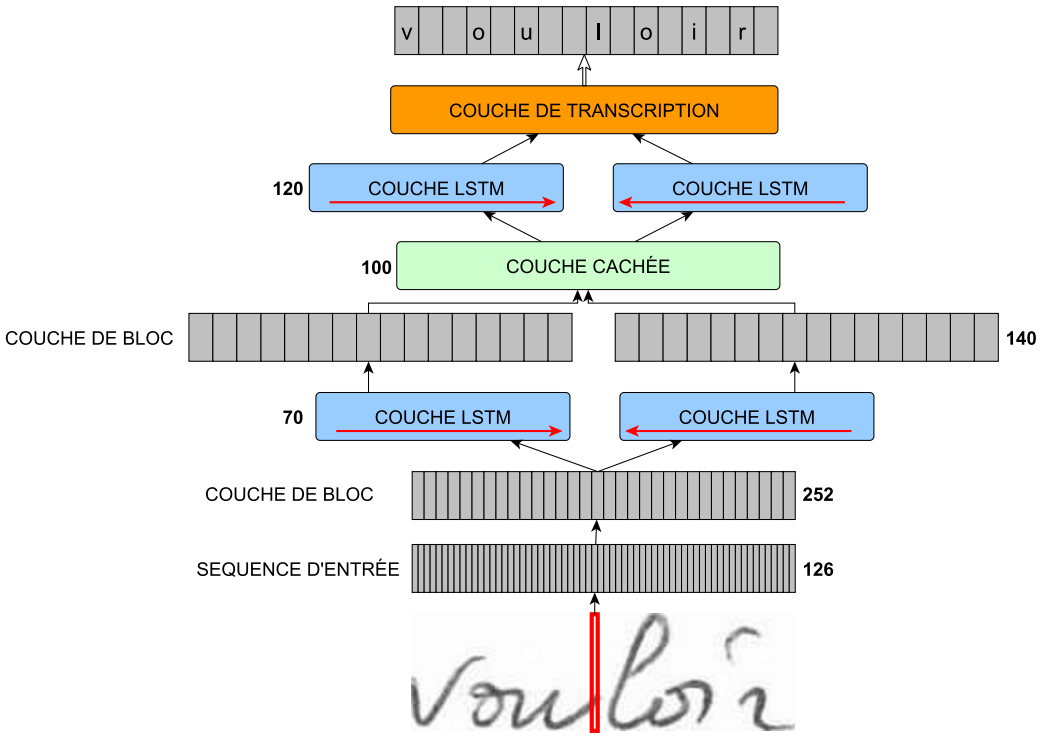


FIGURE 3.6 – Réseau BLSTM utilisé.

et un temps de décodage rapide d'environ 15 millisecondes par mot en moyenne (temps mesuré sur un processeur intel i7-3740QM).

La seconde architecture utilisée est un MDLSTM et est similaire à l'architecture de référence [Graves and Schmidhuber, 2009]. Son architecture se compose de 3 couches récurrentes de 2, 10 et 50 blocs LSTM et de deux couches de rassemblement de 6 et 20 neurones intercalées entre la première et deuxième couche et la deuxième et troisième respectivement. La couche de sortie est également une softmax. Les valeurs des pixels des images d'entrée sont centrées, réduites et directement passées au réseau. Le temps de décodage est de 10 millisecondes en moyenne (sur un processeur intel i7-3740QM).

Pour les deux architectures, nous utilisons l'algorithme de décodage sans lexique : "best path decoding" [Graves, 2012]. Les apprentissages et décodages ont été réalisés avec la librairie RNNLIB [Graves, 2013b].

3.4.2 Génération de la cohorte de LSTM RNN

Afin d'obtenir une cohorte de réseaux LSTM, nous utilisons les trois stratégies suivantes : (i) notre nouvelle stratégie décrite dans la sous-section 3.3.3 permettant d'obtenir un réseau par itération ; (ii) différentes initialisations aléatoires des poids ; (iii) deux architectures différentes le BLSTM et le MDLSTM.

De plus, nous utilisons des déformations sur les données d'apprentissage, afin d'augmenter la taille du jeu de données et sa variabilité. En utilisant des rotations et étirements, nous multiplions la taille de la base par 3. Comme on peut l'observer sur la figure 3.7, le

taux d'erreur mot est plus faible lorsqu'on utilise les déformations. De plus, nous observons que les déformations produisent de plus grandes fluctuations lors de l'apprentissage en raison de l'introduction d'un plus grand nombre d'exemples entre chaque itération. Celles-ci impliquent un plus grand nombre de mises à jour des poids et donc une augmentation potentielle des différences entre deux réseaux obtenus lors de deux itérations consécutives.

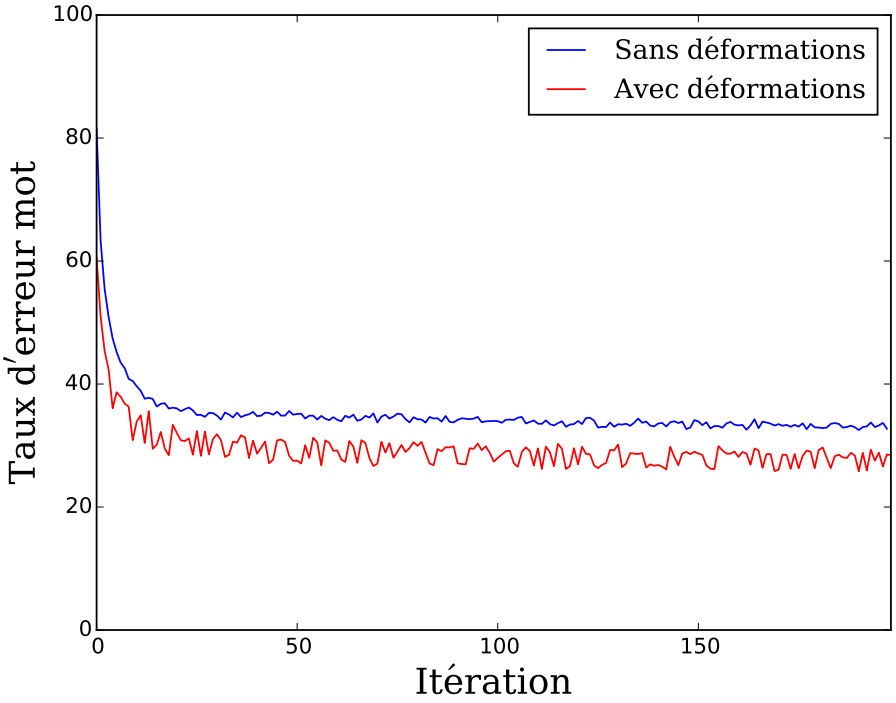


FIGURE 3.7 – Comparaison du taux d'erreur mot sur le nombre d'itérations avec et sans déformations pour un taux d'apprentissage fixé à 10^{-4}. Les erreurs fluctuent avec plus d'amplitude pour le réseau appris avec des déformations.

Nous avons lancé nos expérimentations sur les deux bases publiques les plus répandues : RIMES et IAM.

2100 réseaux LSTM différents ont été obtenus à partir de 10 apprentissages seulement sur la base de données RIMES avec différentes initialisations aléatoires :

- 2 apprentissages BLSTM ;
- 1 apprentissage MDLSTM ;
- 4 apprentissages BLSTM avec déformations ;
- 3 apprentissages MDLSTM avec déformations.

L'apprentissage de ces 10 réseaux sur 3 machines a duré deux semaines, alors que nous estimons que l'apprentissage de 2100 réseaux avec des initialisations différentes aurait duré plus de deux ans.

En ce qui concerne le jeu de données IAM, nous avons appris quatre réseaux avec déformations (triplant la base d'apprentissage), deux BLSTM et deux MDLSTM, donnant un total de 1039 réseaux pour cette tâche.

En réalisant plusieurs apprentissages, le but est triple, il permet : (i) d'obtenir plus de réseaux en réalisant des apprentissages en parallèle, (ii) de vérifier que l'efficacité de la méthode ne dépend pas de l'initialisation aléatoire des poids ou de l'architecture sélectionnée, et (iii) d'améliorer les performances grâce aux deux architectures et différentes caractéristiques.

3.4.3 Résultats expérimentaux

Dans toutes les expériences, nous mesurons les performances avec le taux d'erreur caractère (CER) et le taux d'erreur mot (WER). Nous mesurons également le taux de reconnaissance mot (WRR) et le taux de rejet mot (WJR). Dans l'évaluation pour la base de données RIMES, nous ne prenons pas en compte les erreurs de casse afin de nous comparer aux autres méthodes de la littérature [Menasri et al., 2012, Poznanski and Wolf, 2016] qui ne les considèrent pas. Les mots rejetés ne sont pas pris en considération pour l'évaluation du CER pour la cascade seule.

Nous nous intéressons d'abord aux résultats obtenus pour une cohorte (issue d'un seul apprentissage) puis pour plusieurs cohortes (issues de plusieurs apprentissages).

3.4.3.1 Performance de la cascade construite à partir d'une unique cohorte

Dans cette section, les cascades sont construites à partir d'une seule cohorte de réseaux LSTM issus d'un unique apprentissage sur le jeu de données RIMES.

Nous considérons trois cascades issues de trois cohortes : BLSTM, BLSTM avec déformations et MDLSTM. Nous sélectionnons 100 réseaux de chaque apprentissage pour former les cohortes et construire la cascade avec l'opérateur de vérification lexicale.

Lors de la combinaison des réseaux dans une cascade, les performances dépendent de l'ordre des classifieurs dans la cascade ainsi que de la valeur du Nombre Minimum d'Accord de Décision dans l'étage de décision (NMAD). Nous choisissons d'ordonner les réseaux dans la cascade selon le taux d'erreur d'omission croissant (omission d'un caractère) sur la base de validation. Le NMAD a été paramétré selon la longueur des mots (longs ou courts).

Comme vu précédemment sur la figure 3.3, on peut observer que les mots courts, d'une taille égale ou inférieur à 3, sont plus sujets aux erreurs, justifiant ainsi le fait que les mots soient considérés dans la cascade selon leur longueur. Dans nos expérimentations nous avons choisi un NMAD de 3 pour les mots longs et de 10 pour les mots courts. Cependant, de petites variations de ces valeurs n'ont pas d'impact significatif sur les performances. Par exemple, en utilisant un NMAD entre 2 et 10 pour les mots longs et entre 5 et 40 pour les mots courts, on n'observe quasiment aucune modification du CER (environ 0.01 point) tandis que le WER n'est impacté que de 0.25 point de pourcentage. Il est aussi possible d'optimiser ces paramètres par rapport à la base de validation.

Dans la table 3.2, nous rappelons d'abord à la première ligne les performances d'un BLSTM seul qui atteint un taux de reconnaissance mot de 66%. Le MDLSTM seul obtient un taux de reconnaissance légèrement inférieur à celui du BLSTM. En utilisant des déformations, ce score s'améliore jusqu'à 71%. La combinaison de 100 BLSTM en cascade

permet d'obtenir un WRR de 84.5%. De plus, la même expérience avec 100 BLSTM appris avec déformations atteint un WRR de 89.5%. Finalement, l'amélioration ne dépend pas du type de réseau ou des caractéristiques d'entrée puisque l'on observe aussi une amélioration significative du WRR pour la cascade de MDLSTM.

Système	**WRR**	**WER**	**WJR**	**CER**
BLSTM seul	66.37	33.63	0	11.24
BLSTM seul (+déformations)	71.78	28.22	0	8.89
BLSTM seul + vérification	66.37	2.25	31.38	-
Cascade de 100 BLSTM	84.53	3.13	12.34	0.99
Cascade de 100 BLSTM (+déformations)	89.56	2.74	7.70	0.82
Cascade de 100 MDLSTM	80.27	4.09	15.64	1.40

TABLE 3.2 – Taux de reconnaissance mot (WRR), taux d'erreur mot(WER), taux de rejet mot (WJR) et taux d'erreur caractère (CER) de différents systèmes sur le jeu de données RIMES avec un lexique de 5744 mots. Les réseaux de chaque cascade sont extraits d'un seul apprentissage (i.e. une unique cohorte).

Par cette première expérimentation, nous montrons d'une part que notre schéma de cascade fonctionne et permet d'améliorer les performances, d'autre part que l'opérateur de vérification lexicale est fiable et enfin qu'il y a une complémentarité entre les réseaux d'une cohorte. Cependant, les performances obtenues par une cascade composée d'une seule cohorte ne permettent pas d'obtenir des résultats à l'état de l'art. Nous analysons maintenant les résultats pour des cascades construites à partir de multiples cohortes, afin d'augmenter la complémentarité au sein de la cascade.

3.4.3.2 Performance de la cascade construite à partir de plusieurs cohortes

Nous évaluons dans cette section les performances d'une cascade de réseaux LSTM sur les bases RIMES et IAM, en s'appuyant cette fois ci sur plusieurs cohortes. En plus des données RIMES, nous avons extrait un lexique gigantesque de 3.275.855 mots à partir du dictionnaire français Gutenberg, des pages du Wiktionnaire et de Wikipedia français. Nous faisons de même pour IAM pour lequel nous avons extrait un lexique gigantesque de 2.439.432 mots à partir de la base de données "Google bilion words" [Chelba et al., 2013].

Dans un premier temps, nous rapportons les résultats obtenus pour RIMES en combinant les 2100 réseaux LSTM issus des 10 cohortes décrites dans la sous-section 3.4.2. Comme montré dans la table 3.3, nous atteignons de très bonnes performances pour un lexique de 5744 mots. La différence de reconnaissance entre le petit lexique (1692 mots) et le grand lexique (5744 mots) est faible (0.48 points) en comparaison de la perte de performance que l'on a pu observer dans [Grosicki and El Abed, 2009]. L'écart de performance s'accentue, avec une perte de 2.19 points de WRR, lorsque l'on ajoute le dictionnaire français Gutenberg au lexique de RIMES passant à une taille de lexique de 342275 mots. La différence entre le grand lexique (5744 mots) et le lexique gigantesque (> 3M mots) est encore plus importante mais cependant limitée à 5.94 points soit 5.71%. Ces résultats

montrent bien la faible sensibilité de notre méthode à la taille du lexique. À propos du lexique gigantesque, on peut remarquer que le taux de reconnaissance mot est très intéressant (90.14%) alors que le lexique est 600 fois plus grand que le lexique RIMES. Nous n'avons pas pu nous comparer à d'autres résultats car, à notre connaissance, aucune autre référence dans la littérature n'utilise de lexique aussi grand.

Cascade	**Taille du lexique**	**WRR**	**WER**	**WJR**	**CER**
2100 réseaux	*1692*	96.08	2.32	1.60	0.76
2100 réseaux	*5744*	95.60	3.00	1.40	0.99
2100 réseaux	*342 275*	93.41	5.47	1.12	1.72
2100 réseaux	*3 276 994*	90.14	9.18	0.68	2.67

TABLE 3.3 – Résultats pour une cascade comprenant plusieurs cohortes sur la base RIMES. Nous évaluons les performances de la cascade pour différentes tailles de lexique. (WRR : taux de reconnaissance mot, WER : taux d'erreur mot, WJR : taux de rejet mot, CER : taux d'erreur caractères)

Afin de nous comparer aux autres travaux à l'état de l'art pour lesquels il n'y a pas de rejet, nous avons utilisé un algorithme de Viterbi afin de décoder à l'aide du lexique les rejets à la fin de la cascade. Pour y parvenir, nous utilisons les probabilités moyennées de 10 réseaux LSTM tirés au hasard dans la cohorte. Réaliser la moyenne des probabilités de plusieurs réseaux s'avère plus efficace que de prendre les probabilités d'un seul réseau. il faut noter que le nombre de réseaux pris au hasard n'a quasiment aucun impact sur les performances à partir d'un certain seuil. Sélectionner 5, 10, 20 ou 40 réseaux donne quasiment les mêmes résultats. En testant aléatoirement plusieurs tirages de 5, 10, 20 ou 40 réseaux, on obtient un écart type inférieur à 0.03 sur le CER et le WER. Nous présentons nos résultats dans la table 3.4. Nous nous comparons d'abord à un de nos meilleurs réseaux BLSTM appris avec déformations pour lequel nous avons appliqué un décodage de Viterbi. Nous pouvons voir que l'utilisation de la cascade permet d'augmenter très nettement les performances obtenues. C'est particulièrement flagrant pour le lexique gigantesque combinant le lexique de RIMES et celui du dictionnaire français Gutenberg pour lequel la combinaison en cascade permet de diviser par plus de deux le taux d'erreur caractères et mots. On observe également, que notre méthode est moins sensible à la taille du lexique avec une perte de reconnaissance de 2.56 points contre 4.24 points et une augmentation du CER de 0.77 contre 2.23 pour un décodage dirigé par le lexique De plus, la cascade est beaucoup plus rapide pour décoder un élément qu'un Viterbi appliqué sur un dictionnaire de taille 342k. La différence de performance entre nos résultats et les résultats à l'état de l'art présentés dans [Poznanski and Wolf, 2016] pour le même lexique de taille 5744, aussi bien en terme de CER que de WER est significative. Il y a une diminution absolue de 0.42 (11%) du WER et de 0.56 (29%) du CER.

Les résultats sur le jeu de données IAM sont présentés dans la table 3.5. Les mêmes comportements que sur RIMES sont globalement observés. Le WER et le CER sont inférieurs de 0.52 (baisse de 8%) et de 0.66 (baisse de 19%) respectivement vis à vis des résultats à l'état de l'art. Les résultats pour le lexique gigantesque sont aussi intéressants

Système	Taille du lexique	WRR	WER	CER
1 réseau + Viterbi	*5744*	91.48	8.52	2.77
1 réseau + Viterbi	*342 275*	87.24	12.76	5.00
Cascade de 2100 réseaux + Viterbi	*5744*	**96.52**	**3.48**	**1.34**
Cascade de 2100 réseaux + Viterbi	*342 275*	93.96	6.04	2.11
[Menasri et al., 2012]	*5744*	95.25	4.75	-
[Poznanski and Wolf, 2016]	*5744*	96.10	3.90	1.90

TABLE 3.4 – Résultats pour une cascade comprenant plusieurs cohortes sur la base RIMES. Comparaison des performances à l'état de l'art et à un décodage dirigé par le lexique pour un réseau. (WRR : taux de reconnaissance mot, WER : taux d'erreur mot, WJR : taux de rejet mot, CER : taux d'erreur caractères)

avec 85% des mots reconnus pour un lexique 200 fois plus grand que celui de la base IAM.

Système	Taille du lexique	WRR	WER	WJR	CER
Cascade de 1039 réseaux	*12202*	92.46	**5.04**	2.50	2.08
Cascade de 1039 réseaux	*2 439 432*	85.51	13.30	1.19	4.77
Cascade de 1039 réseaux + Viterbi	*12202*	**94.07**	5.93	-	**2.78**
[Poznanski and Wolf, 2016]	*12202*	93.55	6.45	-	3.44

TABLE 3.5 – Résultats pour une cascade comprenant plusieurs cohortes sur la base IAM. La cascade atteint des performances à l'état de l'art. (WRR : taux de reconnaissance mot, WER : taux d'erreur mot, WJR : taux de rejet mot, CER : taux d'erreur caractères)

Avec la mise en cascade de plusieurs cohortes nous avons obtenu des performances à l'état de l'art renforçant l'intérêt de notre méthode. De plus nous avons montré que la cascade avec vérification lexicale peut utiliser des lexiques de taille gigantesque au prix d'une faible diminution du taux de reconnaissance. De tels lexiques gigantesques, comme le lexique extrait des pages de Wikipédia, contiennent un grand nombre d'entités nommées mais aussi des mots mal orthographiés et des mots d'autres langues. Cette grande variété d'éléments permet d'aborder des problèmes plus complexes contenant un grand nombre d'entités nommées ou des documents multilingues. Nous analysons maintenant deux éléments de notre cascade : la complémentarité des réseaux et les temps de calcul.

3.4.4 Analyse de la complémentarité des réseaux

Dans cette section, nous analysons la complémentarité des réseaux issus d'une cohorte. Pour cela, nous estimons la Similarité des Sorties de Mots des Classifieurs (SSMC) de chaque réseau lors de la phase d'apprentissage. SSMC mesure le pourcentage de réponses identiques au niveau mot entre deux classifieurs. Le SSMC peut s'exprimer par l'équation

suivante :

$$SSMC = \frac{\sum_{i,j}^{N} \delta_{ij}}{N} \tag{3.2}$$

Nous calculons le SSMC pour 100 réseaux sélectionnés sur des itérations consécutives d'une cohorte. Les résultats sont présentés figure 3.8.

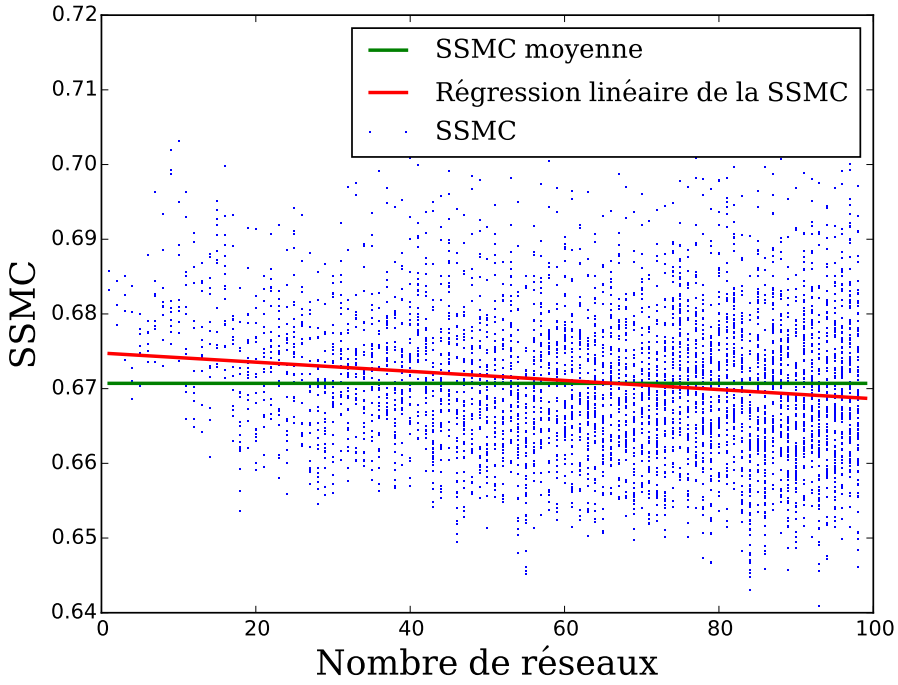


FIGURE 3.8 – Similarité des Sorties de Mots des Classifieurs pour 100 réseaux sélectionnés sur des itérations consécutives d'un apprentissage sur le nombre d'itérations (points bleus) avec la moyenne du SSMC (droite verte) et la régression linéaire du SSMC (droite rouge).

Nous mesurons une SSMC moyenne de 67% avec un écart type de 0.009 seulement, montrant ainsi expérimentalement une complémentarité satisfaisante des réseaux sélectionnés par notre stratégie. On peut noter également que pour un taux d'apprentissage de 10^{-5} la similarité moyenne est de 90%. Ce résultat conforte les choix des paramètres d'apprentissage que nous avons réalisé pour ces expérimentations et notre argumentaire. Comme montré sur la figure 3.8, la régression linéaire sur la SSMC a une pente descendante, signifiant que la complémentarité entre réseaux s'améliore avec le nombre de réseaux.

Pour aller plus loin dans l'analyse des réseaux obtenus, nous calculons le pourcentage des cas où au moins une hypothèse correcte est présente dans la cohorte. Cette mesure permet de calculer la borne supérieure théorique que peut atteindre la cohorte. La figure 3.9 montre ce taux de présence par rapport au nombre de réseaux qui a été calculé pour trois cohortes différentes : BLSTM, MDLSTM et BLSTM avec déformations (ce sont les mêmes cohortes que pour les résultats présentés dans la table 3.2). Pour un réseau BLSTM (courbe bleue), ce pourcentage est de 66%. Pour 10 réseaux il est de 83% et pour 100 BLSTM il est de 90%. L'évolution positive de ce pourcentage de reconnaissance où il n'y

a aucune stagnation pour chaque cohorte sur la figure 3.9 montre bien que plus il y a de réseaux dans la cohorte, plus la probabilité de présence de la bonne solution est grande.

Un autre aspect de l'analyse de ce taux représenté par la figure 3.9 montre des résultats pour la cohorte de BLSTM avec deux règles de décision simples : un vote à la majorité et un vote à la majorité avec vérification lexicale. Le vote à la majorité sélectionne l'hypothèse de mot la plus fréquente parmi les séquences d'hypothèses des classifieurs, alors que le vote à la majorité avec lexique sélectionne l'hypothèse la plus fréquente appartenant au lexique. La figure 3.9 montre que le simple vote à la majorité produit un résultat de reconnaissance mot très limité, alors que lorsqu'on ajoute la règle de vérification, les performances s'améliorent sensiblement de plus de 10 points. Cette observation illustre bien la fiabilité et les capacités intéressantes de la vérification lexicale.

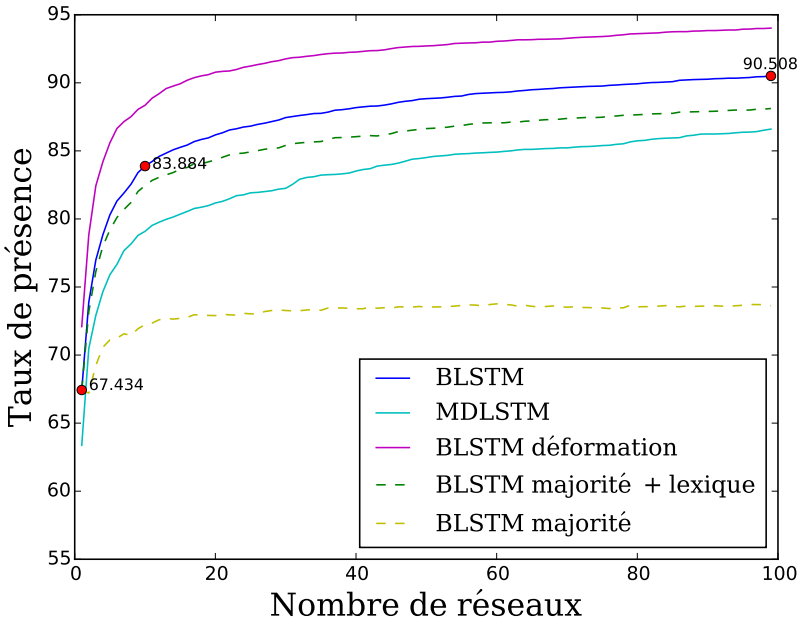


FIGURE 3.9 – Évolution du taux de présence optimal théorique sur le nombre de réseaux pour trois différentes cohortes. La reconnaissance optimale théorique augmente avec la taille de la cohorte.

Cette analyse montre ainsi la complémentarité des réseaux à travers plusieurs aspects. Cependant, le nombre de réseaux utilisés reste important et donc les temps de traitement peuvent représenter un point faible de notre proposition. C'est ce que nous examinons dans le paragraphe suivant.

3.4.5 Temps de calcul

Nous analysons maintenant un inconvénient potentiel de notre approche : elle nécessite un grand nombre de réseaux, ce qui implique des temps de calcul importants. Cependant, l'architecture de la cascade s'avère très efficace en ce qui concerne le coût de calcul, car

dès lors qu'un exemple est validé par l'étage décisionnel, il ne parcourt pas le reste de la cascade. En ce qui concerne le temps de calcul de la vérification lexicale, il est en dessous de la microseconde y compris en considérant un lexique gigantesque car cela se résume à procéder à un accès dans une table de hachage qui est de complexité $O(1)$ en moyenne. Si l'on considère une cascade à 2100 réseaux, le temps moyen de calcul est de 729 millisecondes par mot. 80% des mots sont traités en moins de 0.175s après 14 réseaux ou moins, et 90% en moins d'une demie seconde (après 41 réseaux ou moins). [1] Dans la mesure où notre implémentation n'est pas optimisée ni parallélisée, nous pensons qu'il reste des améliorations possibles.

En ce qui concerne les aspects de mémoire, les 2100 réseaux utilisés dans les expériences nécessitent 6GB de mémoire en les encodant en précision double (64bits).

Ces considérations nous amènent à nous interroger sur la nécessité de mobiliser tous les réseaux de la cohorte ou si un sous ensemble pourrait suffire à obtenir des performances similaires, et ainsi alléger les temps de calcul. Nous nous intéressons à ce point dans la section suivante.

3.4.6 Élagage de classifieurs de la cascade

L'objectif de l'élagage est de limiter les temps de calcul en réduisant le nombre de réseaux de l'ensemble des cohortes de classifieurs pour ne conserver qu'un sous ensemble des réseaux les plus efficaces. L'élagage est effectué en retirant simplement les réseaux ayant les plus mauvaises performances sur la base de validation. Les réseaux sont ordonnés dans l'ordre d'erreur croissante. Les réseaux ne reconnaissant pas de nouveaux mots par rapport aux réseaux précédents ou ayant un taux de fausse acceptation supérieur au taux de bonne acceptation sont retirés. En procédant ainsi, le nombre de réseaux a été réduit de 2100 à 118. Comme montré dans la table 3.6, nous observons une baisse de performances tout à fait acceptable par rapport aux 2100 réseaux. Des résultats très similaires sont d'ailleurs obtenus avec le décodage de Viterbi sur les rejets finaux : le WER est de 3.64% (précédemment 3.48%) et le CER est de 1.49% (précédemment 1.34%). Cette architecture réduite améliore ainsi toujours les résultats à l'état de l'art. Le temps moyen de traitement d'un mot avec cette cohorte élaguée est ramené à 197 millisecondes (précédemment 729ms).

Système	**Taille du lexique**	**WRR**	**WER**	**WJR**	**CER**
Cascade de 118 réseaux	*5744*	92.96	2.28	4.76	0.69
Cascade de 118 réseaux	*3 276 994*	88.82	8.04	3.14	2.20
Cascade de 118 réseaux + Viterbi	*5744*	96.36	3.64	-	1.49

TABLE 3.6 – Résultats de la cascade élaguée à 118 réseaux sur le jeu de données RIMES. Les résultats sont proches de ceux obtenus avec 2100 réseaux avec 18 fois moins de réseaux.

1. Ces temps ont été réalisé sur un processeur i7-3740QM.

3.5 Conclusion

Dans ce chapitre, nous avons introduit un nouveau paradigme de reconnaissance de l'écriture qui remplace la reconnaissance dirigée par le lexique traditionnelle par une combinaison avec un opérateur de vérification lexicale. Ce paradigme est utilisé dans une cascade combinant des centaines de réseaux LSTM. Nous appelons cet ensemble de réseaux une cohorte. La cohorte de réseaux récurrents est obtenue à partir de l'apprentissage d'un seul réseau, nécessitant donc peu d'optimisation des paramètres et un temps d'apprentissage très raisonnable. En analysant la Similarité des Sorties de Mots des Classifieurs sur le jeu de données RIMES, nous avons montré que la complémentarité entre réseaux peut être obtenue grâce à cette stratégie et augmente en fonction du nombre de réseaux.

Une autre raison importante du succès de la combinaison en cascade vient de la combinaison de la vérification lexicale et du Nombre Minimal d'Accord de Décision. La combinaison de ces deux éléments permet d'obtenir une probabilité de fausse acceptation faible tandis que la taille du lexique n'exerce que peu d'influence. Ainsi, la méthode proposée permet d'utiliser des lexiques gigantesques sans qu'il n'y ait d'impact sur les temps de calcul ; ce qui, à notre connaissance n'a jamais été réalisé auparavant. Si les performances de la stratégie diminuent avec l'augmentation de la taille du lexique, le NMAD peut alors être augmenté afin de réduire les confusions générées. Nous avons obtenu dans ce chapitre des résultats à l'état de l'art sur les jeux de données RIMES et IAM de mots isolés avec peu de paramètres à régler et une bonne robustesse par rapport à ces réglages. De plus, le système peut être personnalisé en fonction des contraintes de l'application.

Dans le chapitre suivant, nous nous intéressons à une perspective de ce travail qui est l'extension de cette méthode à la reconnaissance de lignes d'écritures. En effet, la reconnaissance de mots isolés ne suffit pas à résoudre les problèmes réels.

Chapitre 4

Combinaison LV-ROVER de réseaux LSTM

Contents

4.1 Introduction

Dans ce chapitre, nous nous intéressons à la reconnaissance de lignes d'écriture en tentant d'exploiter l'approche de combinaison de classifieurs proposée dans le chapitre 3. Nous avons vu dans le chapitre 2 que la reconnaissance de l'écriture manuscrite est une tâche difficile en raison de la complexité de la langue mais aussi des différents styles d'écriture. La reconnaissance de l'écriture se décompose ainsi en deux parties. Une reconnaissance optique de caractère (OCR) produit une séquence de caractères ou des hypothèses de caractères qui sont ensuite corrigées par un processus linguistique. Dans le chapitre 3, nous avons obtenu des performances à l'état de l'art en combinant des centaines de réseaux LSTM complémentaires. Cette performance a été obtenue en s'appuyant sur deux principes novateurs : la cohorte de réseaux récurrents LSTM et la vérification lexicale. Ces deux principes ont alors été utilisés dans un processus de combinaison en cascade pour la reconnaissance de mots isolés.

Dans la littérature, les méthodes de reconnaissances avec les réseaux LSTM associés à une méthode de décodage dirigé par le lexique avec une modélisation de la langue à partir de n-grams ont permis des progrès significatifs dans le domaine de la reconnaissance de l'écriture. Ces progrès ont permis d'adresser directement le problème de reconnaissance de lignes d'écriture ; la reconnaissance de lignes basée sur une segmentation en mot étant moins performante en raison des erreurs de segmentation difficilement corrigeables. La reconnaissance de l'écriture manuscrite procède d'abord à une segmentation en lignes de documents, puis à la reconnaissance des lignes d'écriture sans introduire d'étape de segmentation en mots. C'est le processus de reconnaissance des lignes qui produit la segmentation en mots. Le processus de reconnaissance de lignes suit le même paradigme que la reconnaissance de mots isolés : c'est-à-dire une reconnaissance de caractères suivie d'un décodage dirigé par le lexique avec modélisation de la langue.

Les résultats à l'état de l'art en reconnaissance de l'écriture de lignes sont obtenus en combinant plusieurs systèmes de reconnaissance (plusieurs modèles optiques et/ou plusieurs modèles de langage) permettant de fiabiliser cette reconnaissance. La technique de combinaison la plus utilisée et donnant les meilleurs résultats s'appelle ROVER (Recognizer Output Voting Error Reduction) [Fiscus, 1997]. ROVER permet de combiner de nombreux systèmes complets (c'est-à-dire comprenant une reconnaissance de caractères suivi d'un décodage dirigé par le lexique) dès lors qu'ils sont complémentaires. Cette combinaison s'appuie sur deux modules : un module d'alignement et un module de vote. Cependant, il est difficile d'entraîner de nombreux systèmes complémentaires, tant pour l'apprentissage des modèles optiques que des modèles linguistiques. C'est pour cette raison que les travaux se limitent à la combinaison de quelques systèmes dans la littérature [Bluche et al., 2014b, Sánchez et al., 2016, Bertolami and Bunke, 2005]. Ces observations nous laissent penser que le principe de combinaison ROVER est sous exploité et qu'il serait possible de l'utiliser pour combiner un grand nombre de classifieurs, en s'appuyant sur les cohortes de réseaux LSTM présentées dans le chapitre précédent. C'est ce que nous proposons d'examiner ici.

Dans ce chapitre, nous présentons une nouvelle méthode exploitant les deux principes novateurs que sont la cohorte et la vérification lexicale dans un contexte de reconnaissance

de lignes d'écriture. L'algorithme que nous proposons s'intitule LV-ROVER pour Lexicon Verified Recognizer Output Voting Error Reduction. LV-ROVER s'inspire de la méthode ROVER tout en s'en démarquant par l'introduction d'un opérateur de vérification lexicale. Cette variante de l'algorithme ROVER permet de combiner des centaines voir des milliers de réseaux issus d'une cohorte sans aucun modèle de langage mais sans toutefois l'interdire. LV-ROVER propose un module d'alignement plus rapide que ROVER en évitant l'utilisation d'une programmation dynamique sensible aux erreurs de reconnaissance de caractères des systèmes n'introduisant pas de contraintes lexicales et linguistiques. Le module de vote, quant à lui, intègre la vérification lexicale pour une décision également rapide. LV-ROVER permet d'obtenir des performances à l'état de l'art, tout en permettant une combinaison plus rapide.

Ce chapitre présente d'abord la combinaison ROVER, puis la combinaison LV-ROVER. Enfin, l'implémentation et les résultats qui en découlent sont étudiés.

4.2 ROVER

ROVER (en anglais Recognizer Ouput Voting Error Reduction) proposé par J.G. Fiscus [Fiscus, 1997] est une méthode de combinaison par vote sur les sorties de reconnaisseurs permettant de réduire les erreurs pour la reconnaissance de la parole ou de lignes de texte. La méthode se décompose en deux modules : le module d'alignement et le module de vote. La figure 4.1 présente ces deux éléments. Dans un premier temps, les sorties de plusieurs systèmes de reconnaissance sont combinées pour produire un treillis de mots. Dans un second temps, le module évalue chaque branche du treillis de mots en utilisant un vote afin d'en extraire la meilleure transcription.

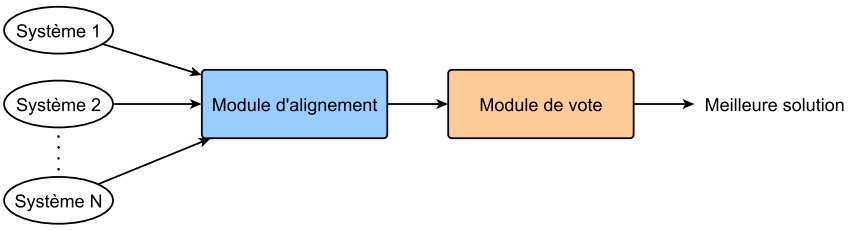


FIGURE 4.1 – Processus de combinaison ROVER présenté dans [Fiscus, 1997].

Dans l'article fondateur de la méthode ([Fiscus, 1997]), un seul module d'alignement est présenté alors que différents schémas de vote sont envisagés.

Le module d'alignement consiste à créer un treillis de mots représentant le meilleur alignement entre les différentes sorties des classifieurs. Ce meilleur alignement entre les N classifieurs est calculé par programmation dynamique en introduisant une heuristique. Le treillis de mots peut être vu comme un tableau à deux dimensions où chaque ligne est une séquence de mots alignée par rapport à toutes les autres et chaque colonne contient les mots alignés à cette position dans le treillis. L'alignement de ROVER se base sur des systèmes de reconnaissance complets produisant des phrases syntaxiquement correctes contenant des mots valides. Ces sorties permettent d'utiliser la programmation dyna-

mique. Cependant, si des sorties contenaient des erreurs de caractères, l'alignement par programmation dynamique [Bellman, 1956] au niveau mot serait plus difficile. En effet, l'alignement par programmation dynamique entre deux mots sans substitution ne peut se faire qu'avec une égalité stricte de leurs chaînes de caractères. On voit que l'utilisation du principe ROVER ne peut s'appliquer tel quel dans notre cas, où chaque réseau de neurones propose une séquence de caractères entachée d'erreurs. Du point de vue algorithmique, on sait que la recherche de l'alignement optimal de N séquences à l'aide de la programmation dynamique est un problème NP-complet [Wang and Jiang, 1994]. Comme écrit par J.G. Fiscus, en pratique, la recherche d'une solution ne peut pas être hyper-dimensionnelle (c'est-à-dire dépasser deux dimensions). Deux raisons empêchent une telle recherche : le coût exponentiel d'un tel algorithme ainsi que la malédiction de la dimension [Bellman, 1962]. C'est pour cette raison que l'algorithme proposé par J.G. Fiscus n'implémente qu'une approximation de cette recherche du meilleur alignement, en alignant successivement chaque séquence d'entrée sur une séquence de référence. La séquence de référence est elle même mise à jour à chaque fois. Les séquences sont alignées afin de minimiser le nombre de délétion, d'insertion et de substitution. La figure 4.2 présente l'alignement de trois séquences : les deux premières sont d'abord alignées, puis la première et la troisième ; enfin le treillis est mis à jour. Un symbole particulier '@' est ajouté aux séquences alignées pour signifier un mot manquant ou en trop lors de l'alignement. Deux séquences alignées ont donc le même nombre de mots. La mise à jour du treillis permet de conserver l'alignement des séquences par rapport à la séquence de référence en ajoutant si nécessaire l'insertion d'un mot manquant symbolisé ici par un '@'. La séquence de référence a un rôle prépondérant dans l'alignement des séquences. En effet, l'algorithme y est sensible et il est alors préférable de choisir la sortie du meilleur classifieur comme séquence de référence. L'algorithme d'alignement de ROVER est présenté de manière non détaillé par l'algorithme 2.

Malgré le coût réduit de cette recherche, la complexité d'un tel algorithme reste élevée. On peut évaluer sa complexité en trois temps : la segmentation en mot ($O(N \times l)$), la réalisation de N programmation dynamique avec Viterbi ($O(L \times L')$) et la mise à jour du treillis ($O(N \times N')$), où N est le nombre de reconnaisseurs, N' le nombre de séquences dans le treillis, L le nombre de mots dans la séquence de référence, L' le nombre de mots dans une séquence de sortie d'un autre reconnaisseur et l la longueur de la sortie (i.e. son nombre de caractères). La complexité totale du module d'alignement est ainsi en $O(N \times (l + L \times L' + N'))$. L'impact de la segmentation en mot et de la mise à jour du treillis est négligeable, donnant ainsi une complexité générale de $O(N \times L \times L')$.

Le module de vote ROVER consiste à parcourir le treillis de mots produit par le module d'alignement afin d'en extraire la meilleure solution. Pour ce faire, c'est un parcours d'un tableau à deux dimensions qui est réalisé afin de calculer la fréquence d'apparition de chaque mot à chaque position dans le treillis. Le mot de fréquence majoritaire est alors sélectionné. Si l'arobase est majoritaire, alors cela signifie qu'il n'y a pas de mot à cette position. Dans une variante de ce vote, on peut également prendre en considération un score de reconnaissance donné par le classifieur, en plus de la fréquence d'apparition du mot. Le chemin de segmentation contenant les mots de confiance la plus élevée est ainsi sélectionné par le module de vote. La complexité de cette recherche est celle d'un parcours

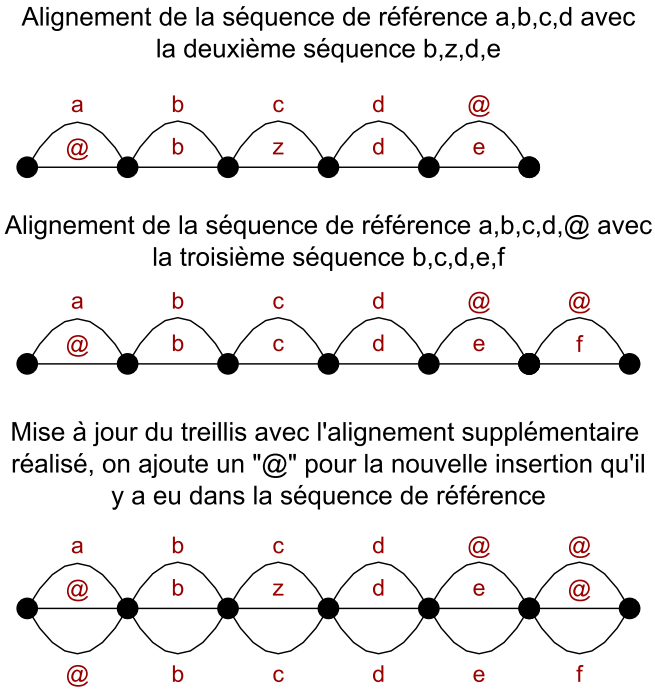


FIGURE 4.2 – Alignement du treillis de mots présenté dans [Fiscus, 1997]. Dans cet exemple, trois séquence de mots sont alignées : "a b c d", "b z d e" et "b c d e f". Les deux premières séquences sont alignées l'une par rapport à l'autre, puis les séquences 1 et 3 sont alignées, enfin le treillis de mots est mis à jour.

d'un tableau à deux dimensions $O(N \times L)$ où N est le nombre de séquences alignées et L le nombre de mots dans la séquence de référence après alignement.

La complexité totale de ROVER est donc en $O(N \times L \times (L' + 1))$, que l'on peut approximer en $O(n^3)$ bien qu'en général c'est le nombre N qui est principalement grand. En plus de la complexité de la méthode ROVER, il faut aussi considérer la complexité de chaque système de reconnaissance qui comprend généralement un décodage dirigé par le lexique à l'aide de modèle de langage de haut niveau. C'est probablement la raison pour laquelle la combinaison ROVER s'est toujours limitée dans la littérature à combiner un petit nombre de systèmes de reconnaissance, incluant systématiquement un modèle de reconnaissance optique et un modèle de langage. À notre connaissance, pas plus d'une vingtaine de reconnaisseurs sont généralement combinés en utilisant ROVER [Bluche et al., 2014b, Sánchez et al., 2016, Bertolami and Bunke, 2005].

Si l'implémentation d'une combinaison ROVER permet de fiabiliser la reconnaissance, cette amélioration est bien souvent pénalisée par le coût de calcul incluant la nécessité de réaliser un décodage dirigé par le lexique et la programmation dynamique pour l'alignement. Les méthodes à l'état de l'art ([Bluche et al., 2014b, Sánchez et al., 2016, Bertolami and Bunke, 2005]) ne décrivent pas de manière exhaustive le processus de génération des systèmes complémentaires ni leur intégration dans un système ROVER. Nous pensons que la combinaison ROVER n'est pas utilisée au maximum de son potentiel en raison des limitations présentées précédemment que sont la complexité algorithmique et le manque

Algorithme 2: Module d'alignement ROVER.

```
Input  : N = nbClassifier, inputStrings = [N]
Output: wordsLattice[N][]
// Séparation de la séquence référence en mots et stockage dans la première ligne
   du treillis.
1  wordsLattice[0] = splitIntoWordsSeq(inputStrings[0]);
// Itération sur le nombre de classifieurs.
2  for i = 1; i < N do
       // La fonction splitIntoWordsSeq permet de séparer une phrase (une chaîne de
          caractères) en mots (plusieurs chaînes de caractères) en itérant sur les
          caractères afin d'en trouver les espaces.
3      wordSequence = splitIntoWordsSeq(inputStrings[i]);
       // Alignement par programmation dynamique en utilisant un algorithme de type
          Viterbi de la séquence de référence par rapport à la séquence i. La
          fonction retourne les deux séquences de mots alignés avec des mots "@"
          insérés s'il y a des insertions dans l'alignement par programmation
          dynamique.
4      alignedRefSequence,alignedWordSequence =
       dynamicProgramming(wordsLattice[0],wordSequence);
       // Mise à jour du treillis de mots avec l'alignement supplémentaire réalisé,
          la séquence de référence conservant sa taille il n'est pas nécessaire de
          répercuter l'alignement sur les autres séquences.
5      if size(alignedRefSequence) == size(wordsLattice[0]) then
6          wordsLattice[0] = alignedRefSequence ;
7          wordsLattice[i] = alignedWordSequence ;
8      else
           // Mise à jour du treillis de mots avec l'alignement supplémentaire
              réalisé, la séquence de référence contient un ou plusieurs mots insérés
              "@". On impacte alors cette insertion dans toutes les séquences
              précédemment alignées en insérant les "@" aux mêmes positions que la
              nouvelle séquence de référence.
9          wordsLattice.update(alignedRefSequence,alignedWordSequence )
10
11 return wordsLattice
```

de méthode pour créer des réseaux complémentaires. C'est pourquoi nous proposons une nouvelle méthode de combinaison basée sur ROVER : LV-ROVER pour "Lexicon Verified" ROVER.

4.3 Combinaison LV-ROVER

L'objectif de LV-ROVER est de combiner les sorties au niveau ligne de texte de réseaux complémentaires à l'aide d'une vérification lexicale afin d'améliorer les performances de reconnaissance. LV-ROVER propose une variante des deux composants de ROVER : l'ali-

gnement et le vote. La difficulté majeure de l'algorithme ROVER réside dans l'utilisation d'une programmation dynamique pour l'alignement posant un problème de complexité et d'alignement sur des chaînes de caractères erronées. En effet, la combinaison ROVER s'effectue sur des sorties de systèmes de reconnaissance contraints par un décodage dirigé par le lexique utilisant un modèle de langage. Les sorties contiennent des mots valides et sont grammaticalement correctes. La programmation dynamique aura des difficultés à aligner des séquences provenant directement de systèmes de reconnaissance de caractères (comme les RNN LSTM) ayant des erreurs caractères. Nous proposons donc un module d'alignement s'appuyant sur un vote à la majorité et un module de vote utilisant la vérification lexicale. Le module d'alignement procède comme suit : il compte le nombre d'espaces dans la sortie de chaque reconnaisseur, puis seules les sorties de reconnaisseur ayant le nombre majoritaire d'espaces sont retenues et insérées dans un treillis de mots. Le module de vote quant à lui, utilise la vérification lexicale afin de favoriser les hypothèses appartenant au lexique et la fréquence des hypothèses lors du parcours du treillis. Un exemple de combinaison LV-ROVER est présentée figure 4.3, montrant le fonctionnement de chaque module.

4.3.1 Module d'alignement

Dans cette thèse, nous cherchons à combiner des centaines de reconnaisseurs, plus particulièrement des réseaux LSTM. Il s'agit donc de combiner des classifieurs de séquences de caractères. Or, nous avons vu que ROVER a été conçu pour combiner des séquences de mots. Il faut donc adapter les principes de l'algorithme ROVER afin de combiner efficacement des séquences de caractères et ainsi produire la meilleure séquence de mots possible.

Pour y parvenir, nous proposons de séparer le module d'alignement en deux étapes séquentielles. La première étape consiste à estimer la longueur la plus fréquente (exprimée en nombre de mots) des hypothèses de ligne. Cette étape est réalisée en calculant l'histogramme du nombre d'espaces dans les hypothèses de ligne, puis en notant la valeur la plus fréquente. La seconde étape aligne uniquement les N' séquences de longueur la plus fréquente W pour les combiner dans un treillis de mots de taille $N' \times W$. L'algorithme 3 décrit la procédure d'alignement de LV-ROVER.

La fiabilité d'un tel module d'alignement peut être obtenue grâce à la combinaison d'un grand nombre de reconnaisseurs, en supposant que la longueur la plus fréquente est probablement la bonne. Nous analysons cette fiabilité en fonction du nombre de réseaux dans les expérimentations de la sous-section 4.4.2. La méthode d'alignement proposée permet d'aligner des mots malgré des erreurs de caractères en se fiant à leurs positions dans les hypothèses. De plus, sa complexité est bien moindre puisqu'elle est de $O(N)$ où N est le nombre de sorties de reconnaisseur sur lesquelles on itère pour établir l'histogramme du nombre d'espaces. L'alignement se fait également sans avoir besoin de déterminer une séquence de référence ou un ordre de classifieurs.

Comme pour ROVER, l'alignement des hypothèses de ligne ne signifie pas pour autant le choix de la segmentation finale. C'est le module de vote qui s'en charge lors du parcours du treillis.

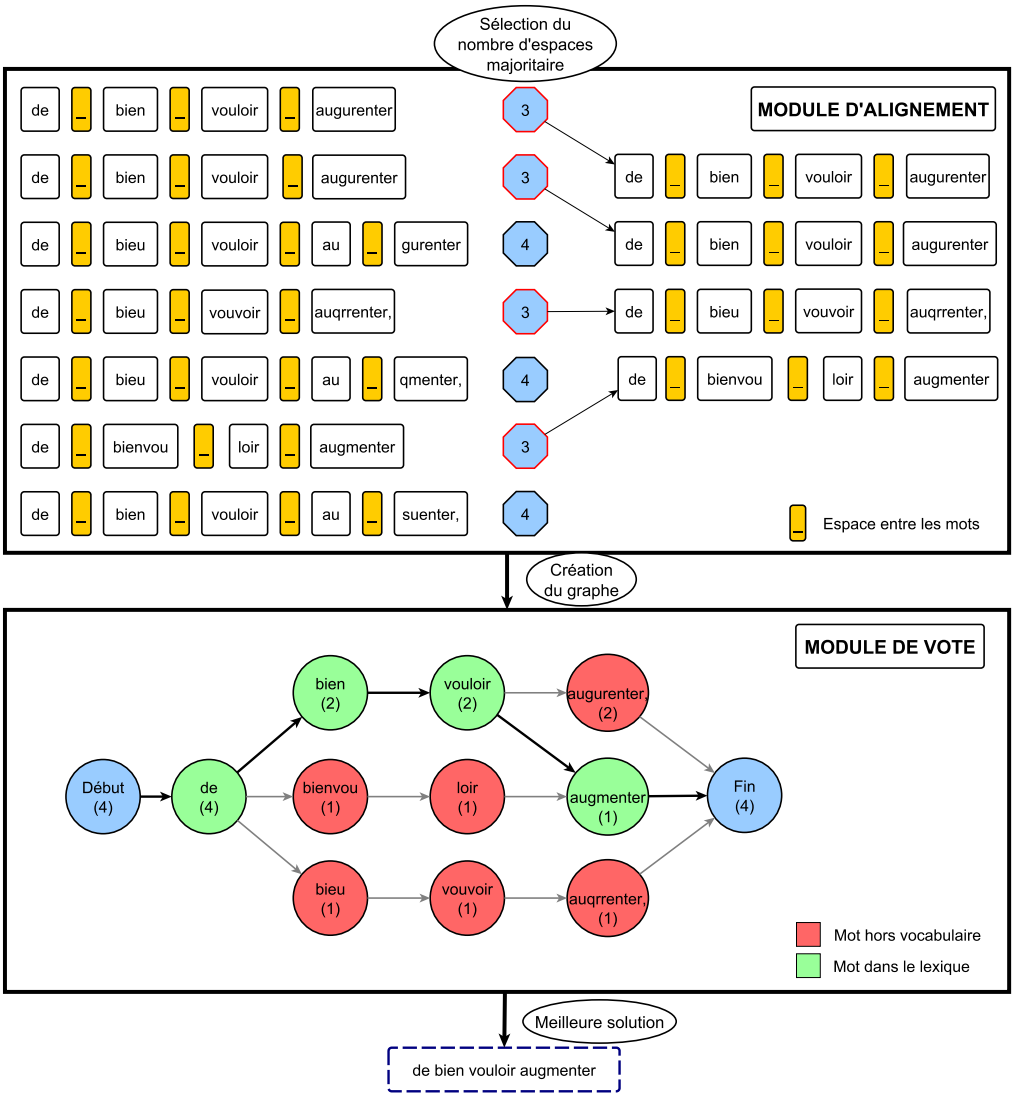


FIGURE 4.3 – Système de combinaison LV-ROVER.

4.3.2 Module de vote

Le module de vote se base sur la fréquence des mots, comme pour l'algorithme ROVER standard. Cependant, nous introduisons une règle de vérification lexicale supplémentaire lors du parcours du treillis de mots. Dans le processus de combinaison ROVER, les lignes sont alignées avec les mots ; la vérification lexicale semble naturellement s'y adapter. Comme vu dans le chapitre précédent, la vérification lexicale est un processus permettant de valider ou d'invalider une hypothèse de mot produite par un reconnaisseur. La vérification consiste simplement à accepter l'hypothèse de mot si elle appartient au lexique. Si elle n'appartient pas au lexique, l'hypothèse est alors écartée. Nous avons par ailleurs montré dans le chapitre précédent sa fiabilité pour la reconnaissance de mots isolés lorsqu'elle est associée à un grand nombre de classifieurs.

Le treillis de mots donné par le module d'alignement est assimilable à un tableau

Algorithme 3: Procédure d'alignement LV-ROVER.

```
Input  : N = nbReseaux, sequenceDeMotsDentree = [N]
Output: tableauDesSolutions[N][]
// Calcul de l'histogramme des espaces sur toutes les sorties de classifieurs.
histogrammeDesEspaces = calculDeLhistogrammeDesEspaces(sequenceDeMotsDentree) ;
nbMots = argMax(histogrammeDesEspaces) + 1;
for i = 0; i < N do
    // Les séquences de sorties de classifieurs n'ayant pas le bon nombre de mots
       sont retirés.
    if length(tableauDesSolutions[i]) != nbMots then
        remove(tableauDesSolutions[i]);
return tableauDesSolutions, nbMots
```

en deux dimensions que nous allons parcourir. Dans le treillis des séquences alignées, le nombre final de mots est déjà fixé. En revanche, la segmentation ne l'est pas encore. C'est pour cette raison qu'au moment du parcours du treillis, on contraint le choix du mot suivant un autre aux seuls mots ayant été trouvés dans les hypothèses données par les classifieurs. Dans l'exemple donné sur la figure 4.3, on peut voir sur la représentation du treillis en graphe que le mot "bieu" n'est relié qu'au mot "vouvoir". Cependant, des connexions peuvent s'établir entre deux mots non consécutifs dans une séquence de sortie et ne respectant pas le schéma de segmentation donné par les hypothèses. Cette situation est possible si et seulement si le mot à la position suivante appartient au lexique. Il faut également que la meilleure solution hors lexique à cette position dans le treillis soit un mot d'une longueur égale à celle du mot considéré à plus ou moins 1 caractère. Ce lien est schématisé sur la figure 4.3 entre les mots "vouloir" et "augmenter" dont l'enchaînement n'existe pas dans les hypothèses de séquences. À chaque position, lors du parcours du treillis, le mot appartenant au lexique de fréquence maximale est choisi. Ce choix est dicté par la stratégie suivante : privilégier les mots faisant partie du schéma de segmentation, ou rechercher dans ceux n'en faisant pas partie. Si aucun mot dans le lexique ne convient, on prend le mot hors lexique de fréquence maximale respectant la segmentation donnée par les hypothèses. Nous présentons l'algorithme de vote (algorithme 4) sur le treillis de mots. Nous utilisons, dans notre algorithme, des tables de hachage contenant les hypothèses associées à leur fréquence. Nous présentons une fonction de mise à jour détaillée de ces tables dans l'algorithme 5. Le treillis est parcouru deux fois : du début vers la fin et inversement. En effet, la meilleure solution peut différer suivant le sens de parcours. La solution du parcours ayant le plus de mots vérifiés dans le lexique est alors conservée.

La complexité de la recherche dans le treillis est en $O(2 \times N' \times W)$ où N' est le nombre de reconnaisseurs restants ($N' \leq N$) et W est la longueur de ligne la plus fréquente. Ces règles de vote permettent de trouver efficacement une solution approchée, avec une complexité équivalente au module de vote de ROVER qui est en $O(N \times L)$ avec ($W \leq L$). Ce module de vote permet également de gérer les mots hors vocabulaire car les mots hors lexique peuvent être aussi pris en compte. Ces considérations représentent un net avantage en comparaison des méthodes dirigées par le lexique.

Algorithme 4: Procédure de vote de LV-ROVER.

```
   Input  : lexique, tableauDesSolutions[N'][nbMots]
   Output: sequenceMotResultat
 1 solutionsCourantesPossibles = tableauDesSolutions[ :][0];
 2 for j ← 0 to nbMots do
       /* Création de trois tables de hachages : solutionsDsLex, solutionsHorsLex et
          solutionsHorsSeg. Ils prennent pour clé le mot et pour valeurs la fréquence
          du mot et la liste des mots suivants possibles. L'accès aux valeurs des
          tables de hachage se fait avec nomTableHachage{mot}.                    */
 3     solutionsDsLex ← new hashTable{}[[]] ; solutionsHorsLex ← new hashTable{}[[]] ;
       solutionsHorsSeg ← new hashTable{}[[]];
 4     for i ← 0 to N' do
 5         mot = tableauDesSolutions[i][j] ; motSuivant = tableauDesSolutions[i][j+1];
           /* On vérifie si le mot est dans les solutions suivantes du mot précédent
              pour ne pas faire d'erreur de segmentation.                         */
 6         if mot in solutionsCourantesPossibles then
 7             if mot in lexique then // Vérification lexicale
 8                 solutionsDsLex.updateHashTable(mot, motSuivant)
 9             else
10                 solutionsHorsLex.updateHashTable(mot, motSuivant)
11
12         else if mot in lexique then
               /* On ajoute dans solutionsHorsSeg tous les mots ∈ au lexique et ne
                  faisant pas partie des mots suivant le mot précédent, ne suivant
                  donc pas le schéma de segmentation trouvé par les classifieurs. */
13             solutionsHorsSeg.updateHashTable(mot, motSuivant)
14
       /* Cas où il y a au moins une solution dans le lexique.                    */
15     if solutionsDsLex is not empty then
           // Ajoute le mot de fréquence max au résultat.
16         sequenceMotResultat += argmaxFreq(solutionsDsLex);
17         solutionsCourantesPossibles ←
           solutionsDsLex{argmaxFreq(solutionsDsLex)}[suivants] ;
18     else
19         estHorsLexique = TRUE ; taille ← size(maxFreq(solutionsHorsLex));
           /* Cas où l'on cherche le mot ayant une taille similaire au mot hors
              lexique de fréquence max et étant dans le lexique mais n'étant pas dans
              les mots suivants du mot précédent.                                 */
20         for motHorsSeg in solutionsHorsSeg.sortedByDecreasingFreq() do
21             if taille-1 < size(motHorsSeg) < taille+1 then
22                 sequenceMotResultat += motHorsSeg;
23                 solutionsCourantesPossibles ← solutionsHorsSeg{motHorsSeg}[suivants];
24                 estHorsLexique = FALSE ; break;
25
26         if estHorsLexique then // Cas où la solution est hors lexique.
27             sequenceMotResultat += argmaxFreq(solutionsHorsLex);
28             solutionsCourantesPossibles ←
               solutionsHorsLex{argmaxFreq(solutionsHorsLex)}[suivants];
29
30
31 return sequenceMotResultat
```

Algorithme 5: Fonction updateHashTable.

```
Input  : hashTable, mot, motSuivant
Output:
if mot in hashTable then
    hashTable{mot}[frequence] += 1 ;// Incrémente la fréquence
    hashTable{mot}[suivants].add(motSuivant) ;// Ajout du mot aux solutions
        suivantes
else
    hashTable.add(mot);
    hashTable{mot}[frequence] ← 1 ;// Initialise la fréquence à 1
    hashTable{mot}[suivants].add(motSuivant) ;// Ajout du mot aux solutions
        suivantes
```

Notre méthode LV-ROVER permet de combiner des centaines de réseaux en gérant des mots hors lexique. La complexité globale de LV-ROVER est en $O(2 \times N' \times W + N)$ soit en l'approximant $O(n^2)$ contre $O(n^3)$ pour ROVER. Cette approximation ou simplification permet de comparer directement les deux complexités et d'en déduire le type complexité : quadratique pour LV-ROVER, cubique pour ROVER. Dans la prochaine section, nous présentons l'implémentation que nous avons réalisée ainsi que les expériences et résultats.

4.4 Implémentation et résultats

4.4.1 Génération de la cohorte

Pour parvenir à générer des centaines de réseaux complémentaires, nous avons proposé dans le chapitre précédent (sous-section 3.3.3) le principe de cohorte qui permet d'extraire des centaines de réseaux d'une seule procédure d'apprentissage. La génération de la cohorte est effectuée de la même manière que lors d'un apprentissage sur les mots isolés, c'est-à-dire en sélectionnant tous les réseaux sur une plage de minimum locaux comme présenté sur la figure 4.4. Nous avons montré précédemment l'efficacité de cette méthode en terme de rapidité et complémentarité des réseaux obtenus en sélectionnant certains paramètres de l'apprentissage qui sont :

- Le mélange des données entre chaque itération ;
- L'utilisation d'un moment à 0.9 pour s'échapper des minimums locaux ;
- La sélection d'un taux d'apprentissage adapté de 10^{-4} permettant de converger sans rester piégé dans un minimum local.

Nous procédons à la génération de multiples cohortes pour les différentes bases de lignes sur lesquelles nous travaillons : READ, RIMES et IAM.

Sur la base READ 2016 (sous-section 2.7.3), nous avons extrait 43 réseaux LSTM de deux apprentissages à partir d'initialisations aléatoires différentes ayant une architecture avec trois couches LSTM de taille 100, 70 et 120.

Afin de générer un maximum de réseaux complémentaires, nous apprenons 4 réseaux différents sur la base RIMES lignes :

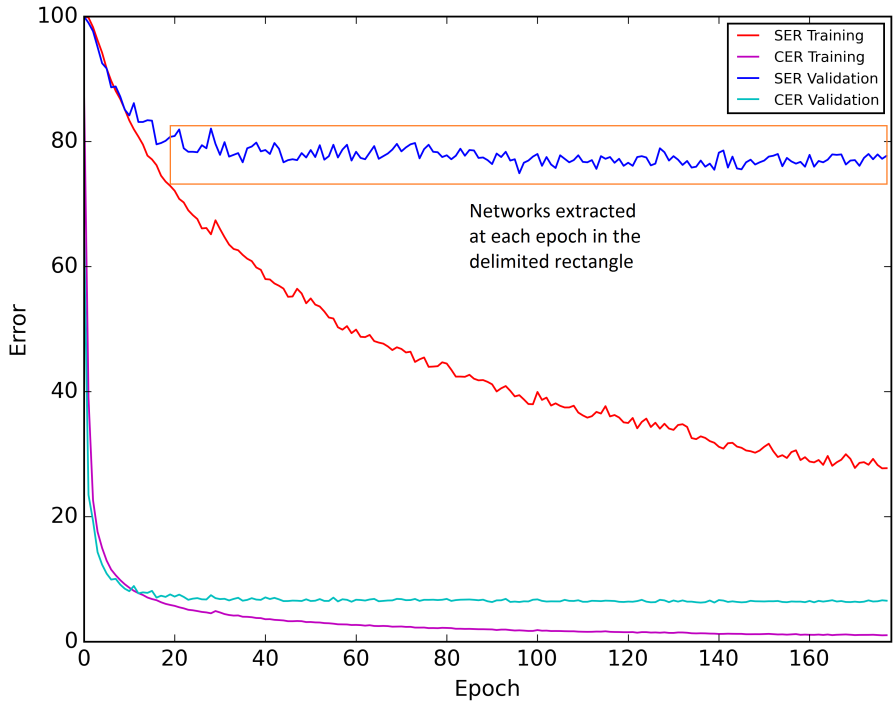


FIGURE 4.4 – Courbes d'apprentissage d'un réseau LSTM à partir duquel on extrait une cohorte présentant le taux d'erreur caractère et le taux d'erreur ligne.

- Un MDLSTM avec trois couches LSTM de taille 2, 10 et 50 respectivement ;
- Deux BLSTM avec trois couches LSTM de taille 60, 70 et 120 appris à partir d'initialisations aléatoires différentes ;
- Un BLSTM avec trois couches LSTM de taille 80, 100 et 120.

Ces 4 apprentissages ont donné une cohorte composée d'un total de 454 réseaux.

Sur la base IAM, nous avons extrait 377 réseaux de 4 apprentissages :

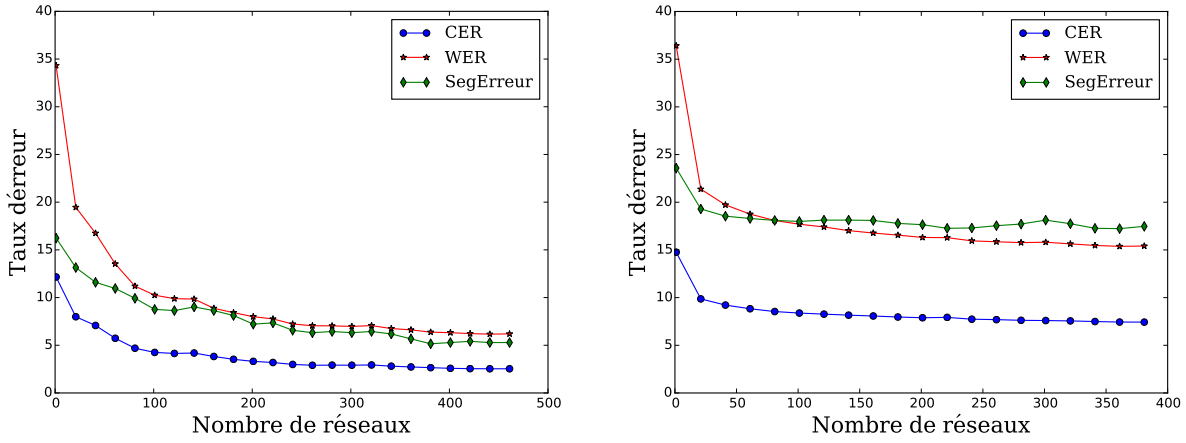
- Deux BLSTM avec trois couches LSTM de taille 60, 70 et 120 appris à partir d'initialisations aléatoires différentes ;
- Deux BLSTM avec trois couches LSTM de taille 100, 120 et 140 appris à partir d'initialisations aléatoires différentes ;

Les réseaux appris sur IAM sont des réseaux BLSTM car leur performance est supérieure au MDLSTM, de plus IAM étant une base plus difficile nous avons aussi augmenter le nombre de cellules LSTM dans les couches.

Comme précédemment, les apprentissages ont été réalisés avec la RNNLIB. Nous étudions d'abord la pertinence de la combinaison LV-ROVER en évaluant la progression de l'erreur en fonction du nombre de réseaux. Nous nous intéressons ensuite aux performances de notre méthode LV-ROVER avec les 454 réseaux de RIMES en les comparant à un ROVER classique. Enfin, nous montrons les résultats obtenus pour READ, RIMES et IAM en les comparant aux méthodes à l'état de l'art. Les résultats obtenus sont ainsi présentés dans les deux parties suivantes.

4.4.2 Évaluation de la combinaison LV-ROVER

Dans un premier temps, nous cherchons à savoir si l'ajout de réseaux complémentaires permettrait d'améliorer les performances de reconnaissance. En effet, nous avons montré précédemment (sous-section 3.4.4) la complémentarité des réseaux issus d'une cohorte. L'ajout de réseaux à une combinaison doit donc apporter une diminution de l'erreur. Nous mesurons la capacité de combinaison à l'aide du CER, du WER et du taux d'erreur de segmentation sur les lignes (SegErreur). Le taux d'erreur de segmentation des lignes est le pourcentage de lignes où le nombre de mots n'est pas celui de la vérité terrain. Cette métrique est intéressante car elle permet de mesurer l'amélioration de l'alignement en fonction du nombre de réseaux. Nous présentons, sur la figure 4.5, l'évolution des taux d'erreur en fonction du nombre de réseaux combinés par LV-ROVER sur la base RIMES (figure 4.5a) et IAM (figure 4.5b). Les lexiques utilisées sont celui de la base RIMES et celui de la base IAM ayant un taux de couverture de 95.23% et 85.75% respectivement. On peut voir sur les deux figures que tous les taux d'erreur évoluent à la baisse en fonction du nombre de réseaux. On peut noter que la tendance de baisse sur IAM est moins prononcée que sur RIMES et que le taux d'erreur de segmentation augmente parfois localement. De plus, un réseau seul sur IAM est moins performant qu'un réseau seul sur RIMES, tous taux d'erreur confondus. C'est particulièrement vrai pour l'erreur de segmentation pour laquelle l'écart est de plus de 5%.



(a) Taux d'erreur en fonction du nombre de réseaux combinés par LV-ROVER sur RIMES.

(b) Taux d'erreur en fonction du nombre de réseaux combinés par LV-ROVER sur IAM.

FIGURE 4.5 – Taux d'erreur en fonction du nombre de réseaux combinés par LV-ROVER.

Nous pouvons déduire de ces courbes de la figure 4.5 que la combinaison LV-ROVER permet de combiner efficacement des réseaux complémentaires. On remarque que le taux d'erreur de segmentation diminue, c'est-à-dire que l'alignement, en comptant le nombre de mots, se fiabilise avec le nombre de réseaux. On peut aussi, grâce à cette expérimentation, affirmer à nouveau la bonne complémentarité des réseaux issus d'une cohorte.

Dans un second temps, nous évaluons les performances de LV-ROVER en fonction de la taille du lexique en utilisant toujours les deux métriques principales que sont le taux

d'erreur caractère (CER) et le taux d'erreur mot (WER).

Nous avons pu voir dans le chapitre 3 que la taille d'un lexique a une influence significative sur les performances d'un système de reconnaissance [Koerich et al., 2003]. Avec un grand lexique, la couverture lexicale est meilleure. Toutefois, la reconnaissance est plus difficile car les confusions entre mots sont plus probables. Afin d'évaluer l'impact de très grands lexiques sur notre méthode, nous avons récupéré deux lexiques supplémentaires (en plus du lexique de la base RIMES) : le dictionnaire français Gutenberg (DFG) et un lexique gigantesque (GIG) composé en plus de tous les mots extraits des pages françaises de wikipédia et du wiktionnaire.

Lexique	Taille Lex.	**Couverture (%)**	**CER (%)**	**WER (%)**
RIMES	8.6K	95.23	2.53	7.84
RIMES + DFG	342K	97.59	2.74	8.18
RIMES + GIG	3.3M	98.65	3.16	10.42
GIG	3.3M	97.28	3.17	10.65
Sans lexique	0	0	4.06	15.76

TABLE 4.1 – Comparaison des résultats de LV-ROVER en utilisant divers lexiques sur la base de données ligne RIMES.

Nous présentons ainsi les résultats de notre approche LV-ROVER pour différents lexiques, avec les 454 réseaux appris sur RIMES présentés précédemment (sous-section 4.4.1), dans la table 4.1. Ces lexiques vont d'une taille de 8578 pour RIMES à plus de 3.3 millions pour le lexique gigantesque. De plus, notre méthode peut fonctionner sans lexique grâce à une gestion des mots hors vocabulaire. Nous présentons donc aussi des résultats sans lexique. On peut voir dans la table 4.1 que l'augmentation de la taille du lexique n'a qu'un faible impact sur les performances de LV-ROVER. Le CER n'augmente que de 0.21 points lors du passage du lexique RIMES au lexique RIMES avec le dictionnaire français et de 0.42 points lors du passage de ce dernier au lexique gigantesque. Ce résultat montre que notre méthode permet de travailler avec des lexiques de très grande taille sans que les performances soient pénalisées. De plus, on peut voir qu'en enlevant le lexique spécialisé de la base RIMES et en ne laissant que le lexique gigantesque, le CER reste stable et le WER n'est augmenté que de 0.23%. La réalisation d'une combinaison LV-ROVER des réseaux sans lexique permet d'obtenir de bons résultats dans un contexte où tous les mots sont hors lexique. Cependant, il est toujours préférable d'avoir un lexique couvrant même gigantesque ; dans la mesure où la vérification lexicale prend autant de temps avec un lexique de 1 mot ou de 3 millions de mots.

Nous avons vu que notre méthode de combinaison LV-ROVER permet de combiner efficacement des centaines de réseaux complémentaires. Elle est également peu sensible à la taille du lexique et peut être utilisée avec des lexiques gigantesques. Nous voulons maintenant comparer les résultats de LV-ROVER avec ceux que nous pouvons obtenir avec une combinaison ROVER.

4.4.3 Comparaison de performances ROVER / LV-ROVER

Notre méthode de combinaison LV-ROVER a une complexité moindre comparée à ROVER et elle permet de combiner aisément des centaines réseaux. Cependant, ROVER peut également combiner des centaines de réseaux au prix d'un temps de calcul plus élevé. Dans la table 4.2, nous présentons donc le résultat obtenu par ROVER pour la combinaison des 454 réseaux et rappelons les performances de notre méthode sur le jeu de données RIMES. On peut voir que LV-ROVER, avec le lexique RIMES, donne de meilleures performances que la combinaison classique ROVER. C'est également vrai pour les autres tailles de lexique et même sans lexique comme présenté dans la table 4.1. La méthode de combinaison ROVER classique ne donne pas de bons résultats car aucune modélisation de la langue n'est utilisée ici. L'alignement ROVER par programmation dynamique nécessite des sorties proches de la bonne solution afin de les aligner correctement. Or, les réseaux LSTM ont des sorties qui n'ont pas été contraintes par une méthode dirigée par le lexique. Cette observation suggère ainsi que la méthode ROVER convient mieux à la combinaison de sorties de systèmes complets de reconnaissance de l'écriture incluant à la fois un système de reconnaissance optique et un modèle de langage, alors que LV-ROVER permet bien de combiner les sorties brutes d'un reconnaisseur comme un réseau LSTM. De plus, dans la table 4.2, nous présentons le temps moyen de calcul de notre module d'alignement qui est plus de deux fois inférieur à celui du module ROVER de SRILM [Stolcke et al., 2002]. Ce temps de calcul porte sur un programme python n'ayant pas fait l'objet d'optimisation particulière. La table 4.2 présente également le taux d'erreur de segmentation mot (SegMot) et ligne (SegErreur) des deux méthodes. La différence en faveur de notre méthode est nette par rapport à la méthode ROVER sur les erreurs de segmentation. Grâce à cette évaluation de l'erreur de segmentation, on constate la fiabilité du module d'alignement de LV-ROVER permise par la combinaison de centaines de réseaux.

Méthode	**CER(%)**	**WER(%)**	**Temps(ms)**	**SegMot(%)**	**SegErreur(%)**
ROVER	4.73	16.74	39.08	2.00	7.34
LV-ROVER	**2.53**	**7.84**	17.28	0.88	5.27

TABLE 4.2 – Comparaison des méthodes ROVER et LV-ROVER pour une combinaison de 454 réseaux LSTM sur la base de test de RIMES.

LV-ROVER dépasse les performances que peut obtenir ROVER pour la combinaison de centaines de reconnaisseurs bruts, tout en écourtant les temps de calculs.

4.4.4 Performances de LV-ROVER

Nous évaluons maintenant les performances de LV-ROVER sur d'autres bases et comparons nos résultats à ceux à l'état de l'art.

4.4.4.1 Compétition READ 2016

La compétition READ 2016 [Sánchez et al., 2016] est une compétition de reconnaissance de lignes d'écritures manuscrites anciennes (voir sous-section 2.7.3), lors de laquelle nous avons ébauché une esquisse de la méthode LV-ROVER. La méthode que nous avons utilisée avait le même module d'alignement. Néanmoins, le module de vote ne créait pas de graphe et réalisait un vote à la majorité pour chaque position de mots dans le treillis avec vérification lexicale. Nous n'avions donc pas de remise en question de la segmentation. Nous gérions les mots hors vocabulaires de la même manière que pour LV-ROVER. Nous présentons dans la table 4.3 les résultats que nous avons obtenus avec d'autres membres du LITIS qui ont notamment travaillé sur la normalisation des lignes et participé aux discussions. Nous nous classons 4$^{\text{ème}}$ de cette compétition avec un système similaire mais moins abouti que LV-ROVER. Cependant, on peut noter que les CER et WER de notre système sont relativement proches de ceux des autres compétiteurs de tête. Pourtant, nous n'utilisons aucune modélisation de la langue dans notre système (à l'exception d'un lexique) contrairement aux autres participants.

Laboratoire	**WER**	**CER**
RWTH	20.9	4.8
BYU	21.1	5.1
A2IA	22.1	5.4
LITIS (ce travail)	26.1	7.3
ParisTech	46.6	18.5

TABLE 4.3 – Résultats de la compétition READ 2016.

4.4.4.2 Résultats sur RIMES

Nous nous intéressons maintenant aux performances de LV-ROVER sur le jeu de données RIMES ligne. Notre système est le même que présenté précédemment, et nous nous comparons aux deux meilleurs systèmes de la littérature dans la table 4.4. Le premier système de [Pham et al., 2014] est basé sur un réseau MDLSTM avec du dropout pour la reconnaissance optique et un modèle 4-grams de mots. Le second système [Voigtlaender et al., 2016] est très similaire et basé sur un réseau MDLSTM/CNN couplé à un modèle de langage 4-grams de mots. Les performances de LV-ROVER surpassent celles des précédentes méthodes en terme de CER et particulièrement en ce qui concerne le WER avec une réduction de 1.8 points soit 18.75%. En ajoutant le dictionnaire français, nous obtenons toujours des résultats à l'état de l'art (voir la deuxième ligne de la table 4.1) alors que le lexique atteint une taille de plus de 342k mots.

4.4.4.3 Résultats sur IAM

Nous comparons maintenant notre méthode aux deux mêmes articles de référence qui ont déjà servi de comparaison pour la base RIMES et qui utilisent le même modèle optique ; seuls leurs modèles de langage changent. Dans [Pham et al., 2014], un modèle de

Méthode	CER	WER
[Voigtlaender et al., 2016]	2.8	9.6
[Pham et al., 2014]	3.3	12.3
LV-ROVER	**2.5**	**7.8**

TABLE 4.4 – Comparaison des performances de notre méthode par rapport à d'autres méthodes sur la base de données RIMES.

langage 3-grams de mots est utilisé ; celui-ci a été appris sur les corpus LOB [Johansson, 1986], Brown [Francis, 1971] et Wellington [Bauer, 1993]. Le corpus LOB est celui dont sont extraites les données de la base IAM. Toutefois, les données utilisées excluent celles contenues dans la base de validation et de test de IAM. [Voigtlaender et al., 2016] utilisent également un modèle 3-grams de mots ainsi qu'un modèle 10-grams de caractères appris sur les mêmes données. Dans nos tests, nous utilisons la même division en bases d'apprentissage, validation et test que dans [Pham et al., 2014, Voigtlaender et al., 2016].

Nous comparons nos performances dans la table 4.5, on peut ainsi voir que nous sommes assez loin des performances de l'état de l'art. Si sur RIMES notre méthode a dépassé les autres méthodes, elle n'atteint pas les mêmes performances sur IAM. Nous pouvons fournir plusieurs explications. La première d'entre elles tient au fait que les réseaux LSTM appris sur IAM sont moins performants sur les données IAM que sur les données RIMES (CER environ 3% plus élevé). Aussi ces performances moindres affectent la combinaison LV-ROVER. En effet, cette conséquence est visible lorsqu'on compare les résultats sur la base de validation et de test. Les résultats obtenus sont meilleurs sur la base de validation ; LV-ROVER arrive au même niveau que la méthode utilisée par [Pham et al., 2014] alors qu'il y a une différence sur la base de test. L'explication vient du fait que la base de validation contient des exemples plus faciles à reconnaître, de ce fait les performances du réseaux sur ces exemples sont meilleurs.

La seconde explication se trouve dans le modèle de langage utilisé pour IAM. Il a été appris sur le corpus LOB qui correspond très exactement aux types de phrases et au langage utilisé pour IAM, ainsi que sur deux autres corpus. Le modèle de langage a pu apprendre sur un grand nombre de données et surtout sur des données spécialisées correspondant à la tâche. C'est particulièrement visible lorsqu'on s'intéresse au taux de mots hors vocabulaire. Les références de la littérature possèdent des taux autour de 4% alors que le notre est à 44.7% avec le seul lexique provenant de la base d'apprentissage et validation de IAM. On peut néanmoins noter que de telles performances à un niveau de OOV aussi élevé sont intéressantes, montrant ainsi la capacité de LV-ROVER à décider de bons mots hors du vocabulaire. En ajoutant le lexique du corpus LOB, nous avons réussi à améliorer significativement le CER et WER obtenus grâce à une diminution des OOV à 9.7%. De plus, le modèle de langage apprend une syntaxe permettant de mieux reconnaître les mots pouvant être fortement confondus comme par exemple "the", "there", "them" et "then". Ce sont ces deux raisons qui peuvent expliquer les performances moindres de notre méthode sur IAM où les mots souvent plus courts sont sensibles aux erreurs caractères et où la syntaxe grammaticale possède une forte importance.

Néanmoins, LV-ROVER reste intéressante par rapport à une méthode ROVER. Si on

Méthode	Lexique	OOV (%)		CER (%)		WER (%)	
		valid	test	valid	test	valid	test
[Voigtlaender et al., 2016]	LOB + Brown + Wellington	4	-	**2.4**	**3.5**	**7.1**	**9.3**
[Pham et al., 2014]	LOB + Brown + Wellington	2.6	3.7	3.7	5.1	11.2	13.6
LV-ROVER	IAM	29.2	44.7	4.4	7.4	11.9	18.2
LV-ROVER	IAM + LOB	8.9	9.7	3.9	6.7	10.7	16.1

TABLE 4.5 – Comparaison des performances de notre méthode par rapport à d'autres méthodes sur la base d'évaluation de IAM.

s'intéresse à la table 4.6, on constate que LV-ROVER obtient des meilleurs taux d'erreur caractère et mot que ROVER sur IAM.

Méthode	CER(%)	WER(%)	SegMot(%)	SegErreur(%)
ROVER	9.58	27.75	4.03	19.77
LV-ROVER	**7.43**	**18.26**	2.68	17.47

TABLE 4.6 – Comparaison des méthodes ROVER et LV-ROVER pour une combinaison de 377 réseaux LSTM sur la base de test de IAM.

Malgré des résultats en demi-teinte sur IAM, nous explorons dans la section suivante l'utilisation de la combinaison LV-ROVER dans un contexte multilingue sur RIMES et IAM.

4.4.4.4 Résultats multilingues RIMES/IAM

Avec la faible sensibilité à la taille du lexique et la gestion des mots hors vocabulaires de LV-ROVER, nous nous sommes demandés si notre méthode pouvait être utilisée dans un contexte multilingue. Nous avons alors réalisé 4 apprentissages sur les données RIMES ligne et IAM ligne mélangées et en avons extraits 253 réseaux. N'ayant pas la division en bases utilisée précédemment pour IAM ligne lors de ces apprentissages, ni le besoin de nous comparer à l'état de l'art pour ces expérimentations, nous avons utilisé la division officielle donnée avec la base IAM. Nous avons également fusionné les deux lexiques gigantesques que nous avions extraits pour RIMES et IAM afin de donner un lexique multilingue ("Giga Fr + En") de 5.246.002 mots.

Nous présentons les résultats obtenus pour une combinaison LV-ROVER de 253 réseaux appris sur RIMES et IAM dans la table 4.7. Sur la première ligne, on peut voir qu'avec moins de réseaux et le lexique de RIMES, on obtient toujours des résultats à l'état de l'art, avec une diminution du CER et du WER. Les résultats sont meilleurs pour RIMES car les réseaux appris avec l'ensemble des données sont meilleurs. Le résultat d'un réseaux sur IAM reste moins bon que sur RIMES. Sur la troisième ligne, le CER et le WER sont similaires à ceux obtenus pour l'autre séparation des données sur IAM. Toutefois, il est impossible d'en déduire davantage en raison du changement de séparation en

bases. Lorsque l'on utilise le lexique gigantesque multilingue (lignes 2 et 4 de la table 4.7), les CER et WER augmentent assez peu sur les bases de test RIMES et IAM. Les taux d'erreur CER et WER sont très satisfaisants dans la mesure où il faut rappeler que nous sommes dans un cadre multilingue, que nous n'avons recours à aucun lexique ou corpus spécialisé, que nous utilisons les mêmes réseaux ainsi qu'un lexique de plus de 5 millions de mots.

Lexique	**Base**	**Couverture** (%)	**CER** (%)	**WER** (%)
RIMES (app+valid)	RIMES	95.22	2.37	7.03
Giga Fr + En	RIMES	92.69	3.67	12.86
IAM (app+valid)	IAM	85.75	7.25	17.99
Giga Fr + En	IAM	98.56	7.53	22.02

TABLE 4.7 – Performances de LV-ROVER pour 253 réseaux appris sur RIMES et IAM.

4.5 Conclusion

Ce chapitre présente nos travaux sur la reconnaissance de lignes d'écriture manuscrite utilisant les deux principes introduits dans le chapitre précédent que sont la cohorte de réseaux LSTM et la vérification lexicale. Nous avons proposé une nouvelle méthode de combinaison LV-ROVER, basée sur la méthode ROVER, permettant de combiner les résultats d'une cohorte de réseaux au niveau ligne. La combinaison LV-ROVER se base comme ROVER sur deux modules : un module d'alignement et un module de vote. Le module d'alignement aligne les sorties des réseaux de même longueur égale au nombre de mots le plus fréquent dans les hypothèses de lignes. Le module de vote est basé sur une vérification lexicale et un vote à la majorité. Le module de vote permet d'établir une segmentation différente de celle donnée par les hypothèses de sorties des réseaux. Lorsqu'aucune solution dans le lexique n'a été trouvée, le module de vote sélectionne le mot hors lexique de fréquence maximale. Nous avons montré que ces deux modules permettent à la fois de segmenter efficacement et trouver une bonne solution tout en étant d'une complexité moindre que ROVER. Nous obtenons des résultats à l'état de l'art avec LV-ROVER sur la base RIMES ligne. Les résultats sur IAM nous montrent que la méthode rencontre des limitations lorsque les performances obtenues pour un réseau ne sont pas suffisantes. Une des perspectives de ce travail serait d'utiliser LV-ROVER pour combiner des réseaux de neurones CNN et LSTM plus performants, voire combiner des systèmes de reconnaissance complets.

La méthode LV-ROVER présente également une faible sensibilité à la taille du lexique utilisé notamment à travers les résultats obtenus avec un lexique gigantesque multilingue. Les résultats multilingues sont très intéressants et ouvrent des perspectives quant à l'utilisation de cette méthode pour des applications mélangeant de nombreuses langues sans avoir à détecter la langue à priori.

Chapitre 5

Auto-apprentissage de réseau LSTM

Contents

5.1 Introduction

L'apprentissage de réseaux profonds a été appliqué avec succès pour résoudre des problèmes complexes, comme nous l'avons vu dans le chapitre 1. Cependant, ces architectures profondes dépendent largement des jeux de données utilisés pour les apprendre. La taille des jeux de données se doit d'être élevée afin de permettre une bonne généralisation des problèmes complexes. Par exemple le jeu de données imagenet [Deng et al., 2009] contient 1 million d'images annotées. De plus, il a été montré par [Sun et al., 2017] que plus cette quantité de données est importante plus les performances des réseaux s'améliorent. Cependant, les jeux de données sont de plus en plus longs et complexes à annoter, rendant le coût d'annotation parfois bien trop important. Y.LeCun a d'ailleurs identifié l'entraînement de réseaux profonds à l'aide de données non étiquetées comme l'une des prochaines frontières de l'apprentissage profond [1].

Dans ce chapitre, nous abordons donc la thématique de l'apprentissage à l'aide de données non étiquetées à travers une méthode d'apprentissage semi-supervisé. Ce chapitre s'inscrit dans la continuité de la thèse car nous cherchons à apprendre des réseaux récurrents à mémoire LSTM pour la reconnaissance de l'écriture. Nous avons pu voir dans le chapitre 2 que l'utilisation de réseaux profonds avec des cellules LSTM était à l'état de l'art pour la reconnaissance hors-ligne de l'écriture manuscrite grâce à leur capacité à traiter les séquences et à reconnaître les caractères. C'est pour cette raison que nous souhaitons investiguer une nouvelle méthode d'apprentissage de ces réseaux à partir de données étiquetées et non étiquetées.

L'apprentissage semi-supervisé [Chapelle et al., 2009] est un compromis entre l'apprentissage supervisé (uniquement des données étiquetées) et l'apprentissage non supervisé (uniquement des données non étiquetées) utilisant une petite proportion de données étiquetées (coûteuses à produire) et une grande proportion de données non étiquetées (peu coûteuses à obtenir). Peu d'articles traitent du problème de l'apprentissage semi-supervisé pour la reconnaissance de l'écriture manuscrite [Ball and Srihari, 2009, Frinken et al., 2011, Frinken and Bunke, 2009]. Ces articles de la littérature présentent des méthodes d'apprentissage semi-supervisé de réseaux LSTM. Ces méthodes se basent sur l'apprentissage de plusieurs classifieurs sélectionnés pour étiqueter de nouvelles données en utilisant des règles de validation basées sur des scores de confiance. Ces approches ont néanmoins des inconvénients car elles requièrent à la fois l'entraînement de multiples classifieurs et un paramétrage (seuils) des règles de validation.

Ce chapitre introduit une alternative d'apprentissage semi-supervisé pour les réseaux LSTM. Cette stratégie s'appuie sur la vérification lexicale que nous avons déjà utilisée dans les chapitres précédents et sur l'utilisation d'une cohorte de réseaux LSTM issue d'un unique apprentissage. Le réseau est d'abord initialisé sur des exemples étiquetés. Ce réseau est ensuite utilisé pour étiqueter des données non étiquetées. Ces nouvelles données étiquetées sont ensuite soumises à une validation par vérification lexicale. Cette méthode exploite également le principe de cohorte afin de fiabiliser la validation d'une étiquette par un vote à la majorité de plusieurs réseaux. Les exemples dont l'étiquette a été validée sont dès lors utilisés pour l'apprentissage du réseau LSTM à l'itération suivante. Cette méthode

1. Séminaire à Carnegie Mellon Univ. le 18 Novembre 2016

nécessite moins de calculs que les méthodes citées précédemment de fait de l'apprentissage d'un seul réseau, et est efficace grâce à la complémentarité des réseaux de la cohorte et à la fiabilité de la vérification lexicale et à sa faible sensibilité à la taille du lexique.

Cette méthode améliore sensiblement les résultats dans des conditions où le nombre d'exemples étiquetés est faible et présente des résultats très intéressants comparé à un apprentissage supervisé. Le taux d'erreur caractère d'un réseau LSTM appris par ce biais s'en trouve fortement diminué pour se rapprocher des performances d'un système appris de façon supervisée.

Dans ce chapitre, nous faisons tout d'abord une revue des méthodes d'apprentissage semi-supervisé pour la reconnaissance de l'écriture. Ensuite, nous présentons progressivement nos propositions. Dans un premier temps, nous nous focalisons sur l'opérateur de vérification lexicale, puis dans un second temps nous introduisons la cohorte. Enfin, les résultats obtenus par ces méthodes sont présentés avant de conclure.

5.2 L'apprentissage semi-supervisé pour la reconnaissance de l'écriture

L'apprentissage semi-supervisé est étudié depuis de nombreuses années [Scudder, 1965]. De nombreux livres et revues traitant de ce sujet ont été publiés [Chapelle et al., 2009, Zhu, 2005]. L'attrait pour les apprentissages semi-supervisés ou non supervisés vient de l'intérêt pour l'utilisation de données non étiquetées. On peut y voir une envie de l'homme de créer une machine artificielle capable d'apprendre par elle même. Le but d'un apprentissage semi-supervisé consiste à améliorer l'apprentissage d'un algorithme à l'aide de données étiquetées et non étiquetées. Il existe une autre forme d'apprentissage semi-supervisé, l'apprentissage semi-supervisé avec contraintes [Abu-Mostafa, 1995] qui utilise des données partiellement étiquetées. Par exemple, deux données peuvent avoir comme information qu'elles n'ont pas la même classe. Les méthodes d'apprentissage semi-supervisé peuvent être transductives ou inductives. Les méthodes transductives cherchent à donner les prédictions pour un jeu de données de test non étiqueté sans nécessairement réaliser une phase préalable d'apprentissage d'un modèle. En opposition, les méthodes inductives cherchent une fonction de prédiction lors d'une phase d'apprentissage pouvant généraliser sur un jeu inconnu de données non étiquetées.

Il existe de nombreuses méthodes d'apprentissage semi-supervisé [Chapelle et al., 2009], on peut citer parmi ces méthodes : les modèles génératifs, l'auto-apprentissage(self-training), les méthodes basées sur des graphes, le co-apprentissage (co-training) et les machines à vecteurs de support transductives (TSVMs, attention les TSVMs sont des méthodes inductives). Les modèles génératifs sont parmi les plus anciennes méthodes [Hartley and Rao, 1968]. Ils se basent sur une distribution de mélange identifiable comme les modèles gaussiens de distribution de mélange (Gaussian Mixture Models : GMMs). Ces méthodes peuvent déterminer les composants de la distribution de mélange à partir d'un exemple étiqueté et de nombreux exemples non étiquetées du composant en utilisant par exemple l'algorithme d'espérance-maximisation (EM). L'auto-apprentissage est aussi une méthode ancienne [Scudder, 1965] qui consiste à entraîner un classifieur à partir de ses

propres prédictions sur des données non étiquetées. Les méthodes basées sur des graphes représentent les données étiquetées et non étiquetées par des nœuds dans un graphe où elles sont reliées suivant leur similarité par des arcs. L'objectif est de déterminer une fonction (la plus lisse possible) donnant des étiquettes aux données non étiquetées proches de celles qui le sont, en utilisant par exemple un algorithme mincut [Blum and Chawla, 2001]. Le co-apprentissage a été introduit pour la première fois en 1998 [Blum and Mitchell, 1998]. Il est basé sur l'apprentissage de deux classifieurs complémentaires. Chaque classifieur étiquette des données qui, pour celles ayant un taux de confiance dépassant un seuil fixé, sont par la suite utilisées par l'autre classifieur comme données d'apprentissage. Les TSVMs [Vapnik, 1998] sont des machines à vecteurs de support capables de gérer des données partiellement étiquetées en suivant les principes de la transduction introduit par Vapnik.

Les premières applications utilisant les techniques semi-supervisées ont été les applications dans le domaine de la langue naturelle [Yarowsky, 1995] et de classification de textes [Nigam et al., 1998]. Plus récemment, les méthodes semi-supervisées ont été utilisées pour des applications de la langue naturelle comme la reconnaissance de caractères dans des documents anciens [Richarz et al., 2014], la détection de mots manuscrits ("word-spotting") [Thomas et al., 2010, Frinken et al., 2014] ou la reconnaissance de l'écriture imprimée [Ait-Mohand et al., 2014]. Nous nous intéressons particulièrement aux trois seuls articles à notre connaissance traitant de l'apprentissage semi-supervisé pour la reconnaissance de l'écriture manuscrite. Un article se sert d'une méthode de co-apprentissage [Frinken et al., 2011], et deux articles d'une méthode d'auto-apprentissage [Ball and Srihari, 2009, Frinken and Bunke, 2009]. Plus spécifiquement deux méthodes s'intéressent à l'apprentissage semi-supervisé de réseaux LSTM ([Frinken and Bunke, 2009, Frinken et al., 2011]). Dans ce chapitre, nous nous concentrons sur l'auto-apprentissage car nous souhaitons une méthode inductive, une mise en œuvre qui ne soit pas complexe et qui soit peu gourmande en calculs. Pour que l'auto-apprentissage fonctionne, la règle de réapprentissage doit permettre à la fois d'étiqueter une quantité suffisante de données tout en introduisant le moins possible d'erreurs sur les étiquettes. Trouver une règle de réapprentissage efficace n'est pas trivial. Dans [Frinken and Bunke, 2009], les auteurs utilisent 10 réseaux BLSTM identiques (avec des initialisations différentes) pour estimer une confiance pour leur règle de réapprentissage. Cette règle s'appuie sur un seuil sur le nombre de réseaux donnant la même solution (la confiance) qui doit être déterminé sur une base de validation. De plus cette stratégie implique l'entraînement de multiples classifieurs, ce qui est coûteux en ressources de calcul et qui va à l'encontre du principe d'auto-apprentissage.

Nous proposons donc dans ce chapitre une méthode d'auto-apprentissage basée sur un unique apprentissage et une règle de réapprentissage utilisant la vérification lexicale.

5.3 Apprentissage semi-supervisé fondé sur la vérification

Nous avons pu voir à travers la littérature qu'un auto-apprentissage pouvait être compliqué à mener car les méthodes s'appuient sur plusieurs classifieurs ou différentes caracté-

ristiques avec des paramètres à optimiser. Ici notre approche est différente et ne nécessite qu'un lexique et qu'un seul réseau à apprendre.

5.3.1 Apprentissage semi-supervisé fondé uniquement sur la vérification lexicale

Dans le chapitre 3, la vérification lexicale a été proposée afin de remplacer des méthodes de décodage dirigées par le lexique. La vérification lexicale examine simplement la séquence de caractères prédite par le réseau LSTM et applique la règle suivante : "si la séquence de caractères appartient au lexique alors elle est acceptée, sinon elle est rejetée". Nous avons pu voir que cette vérification lexicale (voir la sous-section 3.3.2) permet d'avoir un faible taux d'erreur grâce à sa capacité de rejet et à la faible probabilité qu'une hypothèse erronée appartienne au lexique. De plus, la vérification lexicale est un processus très rapide, qui permet de gérer un lexique de taille gigantesque. C'est pour ses capacités que nous avons choisi de l'utiliser dans notre stratégie de réapprentissage.

La règle de réapprentissage est définie par l'ajout d'exemples auto-étiquetés à l'ensemble d'exemples d'apprentissage comprenant déjà des exemples étiquetés. L'objectif est d'augmenter la taille de l'ensemble d'apprentissage tout en maintenant la qualité de l'étiquetage au cours d'itérations successives. Il faut tout d'abord entraîner un réseau LSTM sur des données étiquetées, puis ce réseau sert d'initialisation pour prédire les séquences des données non étiquetées à la première itération d'auto-apprentissage. À chaque itération, toutes les images non étiquetées sont alors passées dans le réseau LSTM et leurs prédictions sont rejetées ou validées par la vérification lexicale pour être utilisées à l'itération suivante. Dans ce processus, toutes les données de l'ensemble non étiqueté initial sont reconsidérées à chaque itération afin que les progrès du reconnaisseur (dans notre cas un réseau LSTM) soient pris en compte. Cette reconsidération permet ainsi de corriger des erreurs et de ne pas avoir un taux d'erreur introduit qui augmenterait systématiquement avec les itérations. L'auto-apprentissage se termine lorsque le taux d'erreur du réseau sur la base de validation n'évolue plus depuis un certain nombre d'itérations. Le fonctionnement de notre méthode d'auto-apprentissage est décrit par la figure 5.1 et l'algorithme 6.

Cette méthode d'auto-apprentissage ne nécessite qu'un seul réseau, n'introduit aucun paramètre supplémentaire, et est simple à implémenter. Les seuls prérequis sont un lexique, une base de données étiquetées (début d'apprentissage et validation) et une base non étiquetée. On peut souligner le fait que n'importe quel lexique peut être utilisé : un lexique relativement petit validera moins de données à chaque itération mais avec un faible taux d'erreur, alors qu'un lexique beaucoup plus grand produira plus de données auto-étiquetées au prix d'un taux d'erreur plus élevé.

Afin de comparer cette nouvelle stratégie, nous expérimentons également une méthode qui s'apparente au co-apprentissage. Deux réseaux LSTM différents sont appris en parallèle sur les données étiquetées. Les exemples non étiquetés pour lesquels les deux réseaux donneront la même hypothèse de séquence de caractères seront utilisés pour réapprendre itérativement ces réseaux. Nous appelons cette règle de réapprentissage l'accord.

Cet accord pouvant être un moyen de fiabiliser la décision, nous proposons dans la section suivante une autre stratégie de réapprentissage utilisant le principe de cohorte et

Algorithme 6: Algorithme d'auto-apprentissage.

Input : networkArchitecture, labeledDataset, unlabedDataset, lexicon
Output: network

```
// selfLabeledDataset créé vide afin d'initialiser le réseau à l'itération 0.
selfLabeledDataset = [] ;
// Itération jusqu'à convergence de l'approche.
while network has not converged do
    network ← train(networkArchitecture, [labeledDataset + selfLabeledDataset]) ;
    selfLabeledDataset = [] ;
    foreach example in unlabeledDataset do
        label ← predict(network, example) ;
        if label in lexicon then
            selfLabeledDataset.add([example,label]);
    testNetworkConvergence();
return network
```

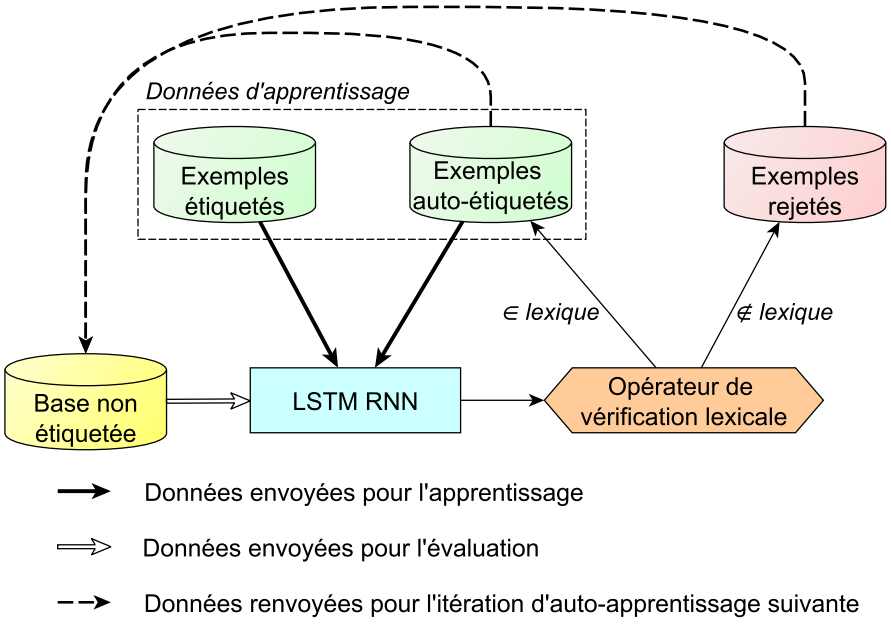


FIGURE 5.1 – Notre processus d'apprentissage semi-supervisé.

restant fidèle au principe d'auto-apprentissage (un réseau appris à partir de lui même).

5.3.2 Apprentissage semi-supervisé fondé sur la vérification lexicale et le principe de cohorte

Nous avons proposé une méthode d'auto-apprentissage basée sur la vérification lexicale pour laquelle nous présentons dans la section suivante des résultats satisfaisants.

Néanmoins, il est toujours possible d'améliorer cette stratégie, notamment en augmentant la proportion d'images auto-étiquetées ou en diminuant la proportion d'erreur voire si possible les deux. C'est pour cette raison que nous proposons d'appliquer le principe de cohorte dans la règle de réapprentissage. Le principe de la cohorte permet d'obtenir des réseaux complémentaires en un seul apprentissage, qui peuvent être utilisés afin de fiabiliser la décision et d'amener de nouvelles solutions comme nous avons pu le voir dans les chapitres 3 et 4.

Pour ce faire, au lieu de ne considérer que les résultats du meilleur réseau sur la base de validation, les résultats de n réseaux issus de la cohorte sélectionnés autour de la meilleure itération de l'apprentissage sont utilisés. La règle de réapprentissage consiste alors à choisir la solution validée par la vérification lexicale majoritaire dans la cohorte. En cas d'égalité la première hypothèse de la liste d'hypothèses est validée. Il est possible d'ajouter également dans cette règle de réapprentissage un seuil sur le nombre minimum de réseau en accord, de la même manière que dans le chapitre 3. Seules les prédictions suffisamment fréquentes au sein de la cohorte seront auto-étiquetés. L'algorithme 7 présente l'algorithme avec l'ajout de la cohorte. Pour plus de clarté l'étape d'initialisation n'a pas été développée car identique à celle de l'algorithme sans cohorte.

Cet ajout du principe de cohorte permet à la règle de réapprentissage d'être fidèle au principe d'auto-apprentissage en apprenant un unique réseau par itération. Il permet également un éventuel paramétrage de la règle avec le seuil sur le nombre minimum de réseau en accord, afin de traiter des cas plus difficiles. Ce paramétrage du seuil n'est pas critique pour l'auto-apprentissage et ne s'avère utile que dans des cas particuliers comme nous le verrons en sous-sous-section 5.4.2.2.

5.4 Expérimentations et Résultats

Dans cette section, les résultats de nos propositions sont présentés sur les bases de mots isolés RIMES et IAM (voir section 2.7). Dans un premier temps nous présentons l'architecture du réseau utilisé pour les expérimentations. Dans un second temps, les résultats de la méthode sans cohorte sont présentés sur RIMES. Nous présentons ensuite les résultats obtenus avec la cohorte sur RIMES et IAM. Enfin, nous étudions l'impact du nombre d'images étiquetées sur l'auto-apprentissage et procédons à une analyse des images auto-identifiées par les différentes méthodes.

5.4.1 Architecture du réseau

Le réseau que nous utilisons pour ces expérimentations est un BLSTM. Il est composé de deux couches cachées de 70 et 100 cellules LSTM. Les images d'entrée sont normalisées à une hauteur de 64 pixels. Les données d'entrée sont les caractéristiques d'histogrammes de gradients orientés (HoG) [Dalal and Triggs, 2005] extraits sur une fenêtre glissante de taille 8 avec un pas de 1 pixel. Les HoG ont montré leur efficacité comme entrée d'un réseau LSTM [Mioulet et al., 2015].

Pour des besoins de comparaison de notre méthode nous utilisons également un second réseau pour la méthode d'accord. Ce réseau est un MDLSTM composé de 3 couches

Algorithme 7: Algorithme d'auto-apprentissage avec cohorte.

```
Input  : networkArchitecture, labeledDataset, unlabedDataset, lexicon
Output: network
// Initialisation du réseau identique à celle de l'algorithme précédent.
network, selfLabeledDataset = initialize(networkArchitecture, labeledDataset,
unlabeledDataset);
// Itération jusqu'à convergence de l'approche.
while network has not converged do
    networks ← train(networkArchitecture, [labeledDataset + selfLabeledDataset]) ;
    selfLabeledDataset = [] ;
    foreach example in unlabeledDataset do
        labels = [];
        foreach network in networks do
            label ← predict(network, example) ;
            if label in labels then
                labels[label] = labels[label] + 1;
            if label not in labels then
                labels[label] = 1;
        // Itération sur les étiquettes triées par valeur décroissante.
        foreach label in sortedByValue(labels) do
            if label in lexicon then
                selfLabeledDataset.add([example,label]);
                break;
    testingNetworkConvergence();
network ← getBestNetworkOnValid(networks) return network
```

contenant respectivement 2, 10 et 40 neurones travaillant directement sur l'image. Cette architecture a été présentée dans [Graves and Schmidhuber, 2009] et atteignait à l'époque de la publication, l'état de l'art sur de nombreux jeux de données.

Nous utilisons toujours la RNNLIB [Graves, 2013b] pour les apprentissages et réalisons une descente de gradient stochastique avec un moment de 0.9 et un taux d'apprentissage fixe de 10^{-4}. Les apprentissages sont stoppés dès que le taux d'erreur du réseau n'est plus significativement amélioré sur la base de validation.

5.4.2 Expérimentations sur l'auto-apprentissage

Dans le cadre de nos expérimentations nous utilisons plusieurs métriques :
- Le taux d'erreur caractère (CER) ;
- La précision qui est le taux de reconnaissance caractère ;
- Le taux d'erreur mot.

Nous définissons également deux nouvelles métriques :
- L'amélioration : c'est la progression relative de la précision (en %) entre un réseau

appris sur les données étiquetées (itération 0) et un réseau appris sur les données étiquetées et non étiquetées (calcul : $\frac{CER_{iteN}-CER_{ite0}}{CER_{ite0}}$) ;
– % vers supervisé : c'est le pourcentage relatif qui sépare un réseau appris de manière semi-supervisé d'un réseau appris sur toutes les données de manière supervisée (calcul : $\frac{CER_{supervise}-CER_{iteN}}{CER_{supervise}}$).

Pour ces expérimentations nous avons extrait au hasard dix mille images de la base d'apprentissage RIMES afin de servir d'initialisation. Les autres images sont alors considérées comme des images non étiquetées. Avec le jeu de données défini, nous pouvons procéder aux itérations d'auto-apprentissage pour lesquels nous présentons les résultats ci après.

5.4.2.1 Expérimentations avec la vérification lexicale seule

Nous reportons, dans la table 5.1, les résultats de notre méthode semi-supervisée avec uniquement la vérification lexicale sur le jeu de test RIMES pour la meilleure itération sélectionnée sur la validation. "Entièrement supervisé" désigne les performances du réseau appris sur l'ensemble des données étiquetées de la base RIMES d'apprentissage soit 51k mots. On peut observer que notre méthode semi-supervisée réalise de bonnes performances ; l'amélioration est significative quelle que soit la taille du lexique en partant d'un apprentissage appris sur uniquement dix milles images étiquetées. Sur la figure 5.2, on peut observer que notre méthode est efficace pour les 3 lexiques, même si la précision pour le lexique gigantesque est moindre. La méthode dépasse largement les résultats obtenus par un accord entre deux réseaux aux architectures différentes et exploitant des caractéristiques différentes.

Stratégie	CER	Amélioration(%)	% vers supervisé
Entièrement supervisé	11.45	-	-
Supervisé 10k	20.34	-	43.7
Vérification 5k (ite13)	13.51	33.5	17.9
Vérification 340k (ite15)	13.70	32.6	19.6
Vérification 3M (ite13)	14.97	26.4	29.8
Accord (ite10)	16.36	19.5	42.8

TABLE 5.1 – Comparaison des taux d'erreur caractère (CER) obtenus pour : un apprentissage supervisé sur toute la base, un apprentissage supervisé sur 10k mots et notre stratégie d'apprentissage semi-supervisé avec 3 lexiques différents.

Dans la table 5.2, nous présentons les résultats de notre méthode en utilisant un décodage de type Viterbi [Viterbi, 1967] afin de comparer notre méthode avec le lexique de la base RIMES (5k mots). Le taux d'erreur mot s'améliore de 7.07 points grâce à l'apprentissage semi-supervisé, il n'est pas très éloigné des valeurs que l'on obtient pour l'apprentissage entièrement supervisé (3.73 points). Pour étendre la comparaison nous regardons les résultats de la compétition RIMES à ICDAR 2011 [Grosicki and El-Abed, 2011] et plus particulièrement le système proposé par Jouve qui combine un modèle HMM classique et l'apprentissage d'un MDLSTM avec décodage dirigé par le lexique qui réalise un score de

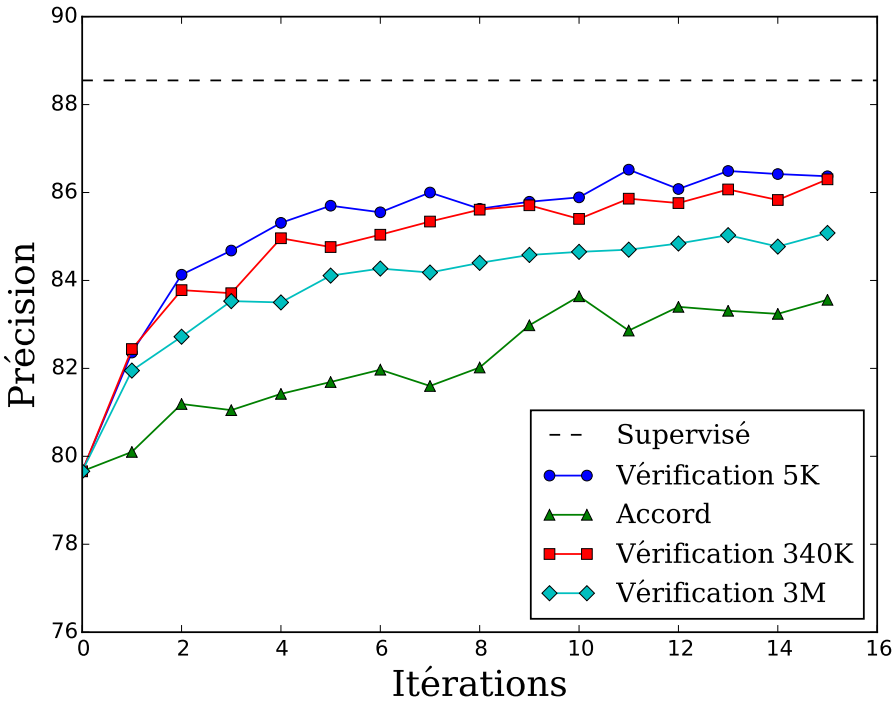


FIGURE 5.2 – Précision en fonction du nombre d'itération. La ligne en pointillé représente le CER obtenu par l'apprentissage supervisé sur toute la base. Vérification 5k, 340k, 3M sont les résultats de notre méthode pour 3 tailles de lexique.

reconnaissance de 77.45% sur la base de validation. Nous nous positionnons juste derrière la deuxième place qui a un système similaire au notre (réseau LSTM suivi d'un décodage dirigé par le lexique) avec une performance nettement supérieure au troisième participant en n'utilisant pourtant qu'un cinquième de la base d'apprentissage étiquetée.

Stratégie	WER
Supervisé full	11.10
Supervisé 10k	21.90
Vérification 5k	14.83
ICDAR 2011 second place [Grosicki and El-Abed, 2011]	12.53

TABLE 5.2 – Comparaison de notre taux d'erreur mot (WER) avec décodage de Viterbi entre l'itération 0 et la borne maximale représentée par l'apprentissage supervisé sur toute la base.

Nous nous intéressons maintenant aux apports de la cohorte à la règle de réapprentissage pour la même expérimentation.

5.4.2.2 Expérimentations avec la vérification et la cohorte

Nous reportons, dans la table 5.3, les résultats de notre méthode semi-supervisée avec la vérification lexicale et la cohorte sur le jeu de test RIMES, pour la meilleure itération sélectionnée sur la validation. Nous utilisons une cohorte de 11 réseaux composée des 5 réseaux précédents et des 5 suivants le meilleur réseau sur la base de validation lors de l'apprentissage. Nous rappelons également les résultats pour la méthode sans cohorte. On peut voir que l'ajout de la cohorte sans paramètre en sélectionnant l'hypothèse majoritaire améliore significativement les performances passant ainsi d'un CER de 13.51% à 12.73%. On peut voir qu'il en est de même dans la table 5.4 pour un lexique de taille gigantesque, le CER est diminué de 1.11 points grâce à l'ajout de la cohorte.

Stratégie	Nb réseaux	Seuil	CER	Amélioration(%)	% vers supervisé
Entièrement supervisé	-	-	11.45	-	-
Vérification	1	0	13.51	33.5	17.9
Vérif + cohorte	11	0	12.73	37.4	11.1
Vérif + cohorte	11	1	12.72	37.4	11.1
Vérif + cohorte	21	3	12.58	38.1	9.8
Vérif + cohorte	21	7	13.17	35.2	15.0

TABLE 5.3 – Comparaison des taux d'erreur caractère (CER) obtenus pour notre stratégie d'apprentissage semi-supervisé avec vérification lexicale (lexique de 5k mots) et cohorte pour différents paramètres : seuil d'acceptation de la cohorte, nombre de réseaux dans la cohorte.

Stratégie	CER	Amélioration(%)	% vers supervisé
Entièrement supervisé	11.45	-	-
Vérification (ite13)	14.97	26.4	30.7
Vérif + cohorte (ite13)	**13.86**	31.8	21.0

TABLE 5.4 – Comparaison des taux d'erreur caractère (CER) obtenus pour notre stratégie d'apprentissage semi-supervisé avec vérification lexicale et cohorte avec un lexique gigantesque de 3M mots.

Nous avons vu qu'il est possible d'ajouter un paramètre à la décision de cohorte qui est le seuil sur le nombre minimum d'accord nécessaire entre les hypothèses de la cohorte. Dans la table 5.3, nous testons trois valeurs : 1, 3 et 7. On peut voir que la paramétrisation de la règle d'accord et l'augmentation du nombre de réseaux à 21 (10 réseaux précédents et 10 suivant) n'influent quasiment pas sur les performances (écart type de 0.22 sur le CER). De plus sur la figure 5.3, on peut observer que toutes les courbes avec ou sans paramètre sont presque confondues. Le fait de prendre l'hypothèse vérifiée majoritaire est une règle de réapprentissage suffisante et efficace qui se passe de paramètres. Néanmoins pour des jeux de données plus complexes ces paramètres pourraient s'avérer utiles.

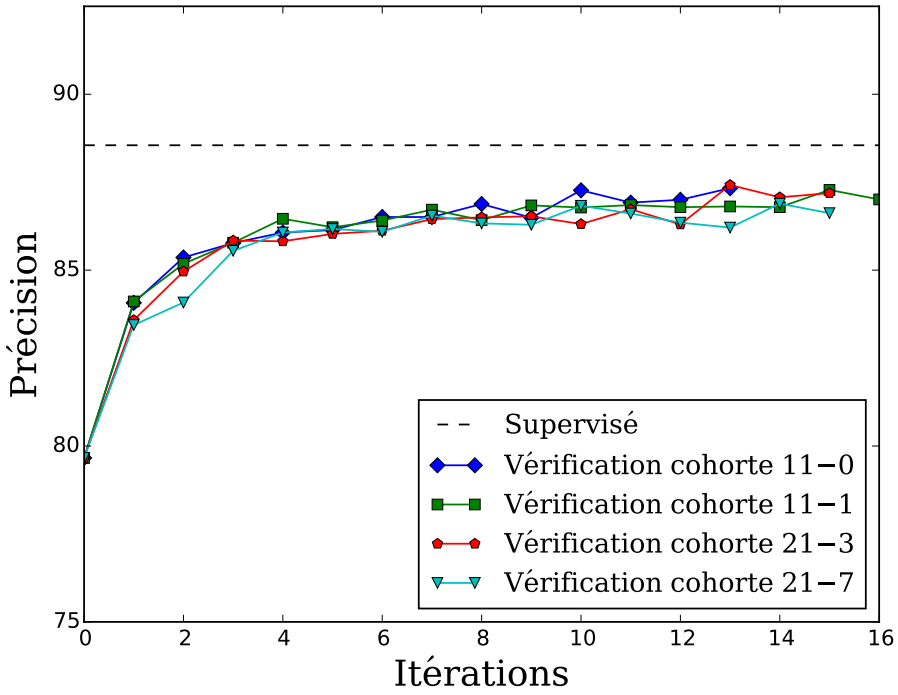


FIGURE 5.3 – Précision en fonction du nombre d'itération pour différents paramétrages de la règle de réapprentissage avec cohorte. Le premier chiffre indique le nombre utilisé de réseaux de la cohorte, le deuxième le seuil de validation. La ligne en pointillé représente le CER obtenu par l'apprentissage supervisé sur toute la base.

Nous calculons à nouveau le WER à l'aide du même algorithme de Viterbi pour le meilleur réseau de la meilleure itération avec la cohorte. Dans la table 5.5, nous comparons le taux d'erreur mot avec et sans la cohorte, et nous observons le même gain qu'avec le CER. En allant plus loin et en moyennant toutes les probabilités de toutes les trames des 11 réseaux de la cohorte et en y appliquant un décodage de Viterbi, nous obtenons un WER de 11.34%, dépassant ainsi les performances obtenus par le deuxième participant dans le cadre de la compétition ICDAR 2011 avec un cinquième des données. Le fait de moyenner les matrices de probabilités permet de combiner les sorties des BLSTM avant l'application de l'algorithme de Viterbi.

Stratégie	WER
Vérification 5k + Viterbi	14.83
Vérification + cohorte 5k + Viterbi	13.96
Vérification + cohorte 5k + proba moyenne + Viterbi	**11.34**

TABLE 5.5 – Comparaison du taux d'erreur mot (WER) avec décodage de Viterbi entre nos deux méthodes avec ou sans cohorte et avec ou sans moyenne des probabilités de la cohorte.

Notre méthode d'auto-apprentissage avec cohorte présentant d'excellents résultats sans paramétrages, nous réalisons un auto-apprentissage avec cohorte sur le jeu de données IAM de mots isolés afin de confirmer ces résultats. Comme pour les expériences précédentes nous tirons au hasard 10000 exemples étiquetés et utilisons le reste des exemples de la base d'apprentissage pour l'auto étiquetage. Les résultats sur la base IAM sont présentés sur la figure 5.4. Sur le critère de la base de validation nous obtenons une précision de 81.76%, très proche de la limite d'un entraînement entièrement supervisé qui atteint 84.15% de précision.

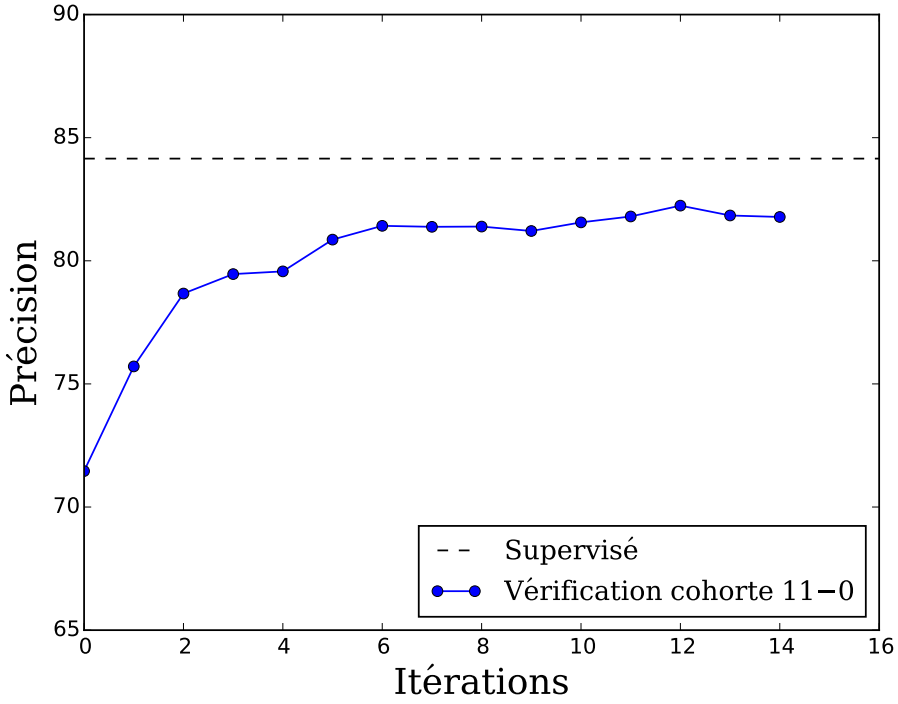


FIGURE 5.4 – Précision en fonction du nombre d'itération sur le jeu de données IAM.

Notre méthode avec ou sans cohorte permet d'obtenir de très bons résultats sur des bases différentes RIMES et IAM. Pour valider nos résultats, nous comparons notre méthode dans la table 5.6 à d'autres approches de la littérature [Frinken and Bunke, 2009, Frinken et al., 2011]. Cependant, ni l'architecture du réseau LSTM, ni le découpage de la base de donnée n'aient été indiqués dans ces deux articles ; la comparaison ne peut donc pas se faire sur les mêmes données. Nous pouvons quand même nous comparer en utilisant les métriques du % vers supervisé et de l'amélioration. Nous reportons les résultats des différentes méthodes, pour ces deux métriques, dans la table 5.6. Nous calculons les deux critères à partir des résultats donnés sur les courbes des articles. Pour les deux critères notre méthode avec ou sans cohorte sur RIMES ou IAM dépasse les méthodes à l'état de l'art et se montre donc plus efficace. De plus on peut observer que sur IAM notre méthode a permis une nette amélioration de 38.2% par rapport au point d'initialisation tout en étant proche de la limite d'un entraînement entièrement supervisé.

La cohorte est un atout indispensable à utiliser afin de faciliter l'auto-apprentissage

Système	Base	Amélioration(%)	% vers supervisé
Notre méthode	RIMES	33.5	17.9
Notre méthode avec cohorte	RIMES	37.4	11.1
Notre méthode avec cohorte	IAM	38.2	15.0
[Frinken and Bunke, 2009]	IAM	16.1	15.1
[Frinken et al., 2011]	IAM	19.1	-

TABLE 5.6 – Comparaison de l'amélioration de la précision par notre méthode semi-supervisé par rapport à d'autres méthodes à l'état de l'art.

et elle ne nécessite aucun paramètre. Le seul coût d'utilisation de la cohorte se fait lors du décodage des exemples que l'on souhaite auto-étiqueter. L'utilisation de la cohorte présente d'autres avantages, notamment, lorsque le nombre de données étiquetées en entrée est moindre. Nous étudions dans la prochaine sous section l'impact d'une diminution du nombre de données étiquetées.

5.4.3 Impact du nombre d'exemples étiquetés sur l'auto-apprentissage

Afin d'étudier plus en détails notre méthode d'auto-apprentissage, nous expérimentons un auto-apprentissage avec trois tailles de jeu de données étiquetées différentes : 10000 (résultats précédents), 6000 et 2000.

Le premier objectif est d'observer le comportement de notre méthode dans un contexte plus difficile pour lequel le réseau aura des difficultés à proposer une séquence de données correctes. Le second objectif est d'étudier l'apport du principe de cohorte par rapport à la vérification seule sans accord. En effet l'utilisation d'un critère de majorité avec une dizaine de réseaux issus d'une cohorte a permis de fiabiliser le taux d'erreur et/ou d'augmenter le nombre d'images auto-étiquetées.

Nous mesurons la difficulté de la tâche en calculant les performances CER et WER du réseau à l'issue du premier apprentissage avec uniquement les données étiquetées. Dans la table 5.7, on observe qu'avec 10000 exemples étiquetés la moitié des mots sont en erreurs sur la base de test. Le taux d'erreur mot augmente un peu avec 6000 exemples étiquetés et devient très élevé, atteignant plus de 75% avec 2000 exemples étiquetés. La difficulté est donc particulièrement présente car au moins un mot sur deux est faux à l'initialisation, se dégradant jusqu'à 3 mots sur 4 pour une initialisation avec 2000 exemples.

Nb étiquetés	CER	WER
10k	20.34	51.31
6k	24.15	56.97
2k	39.34	76.37

TABLE 5.7 – Taux d'erreur caractère (CER) et mots (WER) obtenus par le réseau BLSTM à l'initialisation en fonction du nombre d'exemples étiquetés.

Dans la table 5.8, les différents CER obtenus par notre méthode d'auto-apprentissage par vérification lexicale avec ou sans cohorte pour différentes initialisations sont présen-

tés. La figure 5.5 présente les courbes de ces CER en fonctions du nombre d'itération. On peut tout d'abord déduire de la table 5.8 et de la figure 5.5 que notre méthode avec ou sans cohorte fonctionne quel que soit le niveau d'information de départ. En effet le CER progresse positivement qu'il y ait 10000, 6000 ou 2000 exemples étiquetés. Cependant, une plus grande quantité d'exemples étiquetés au départ produit en toute logique de meilleurs résultats. On peut également observer que l'utilisation du principe de cohorte permet d'améliorer les performances de notre méthode d'auto-apprentissage avec vérification lexicale. L'ajout de cette règle est d'autant plus important que le nombre d'exemples étiquetés est faible, améliorant ainsi de 0.79 points pour 10k exemples et jusqu'à 1.47 points pour 2k. La figure 5.5 montre aussi que l'ajout de la cohorte permet une convergence plus rapide de notre méthode. C'est particulièrement visible pour 2k exemples entre la courbe violette (avec cohorte) et la jaune (sans cohorte).

Stratégie	Nb étiquetées	CER
Entièrement supervisé	51k	11.45
Vérification	10k	13.51
Vérification + cohorte	10k	**12.72**
Vérification	6k	14.68
Vérification + cohorte	6k	**13.72**
Vérification	2k	17.11
Vérification + cohorte	2k	**15.64**

TABLE 5.8 – Comparaison des taux d'erreur caractère (CER) obtenus pour notre stratégie d'apprentissage semi-supervisé avec vérification lexicale et avec ou sans cohorte en fonction du nombre d'exemples étiquetés.

5.4.4 Analyse des images auto-étiquetées

La qualité et la quantité des données auto-étiquetées sont importantes car elles déterminent l'efficacité de notre règle de réapprentissage, nous analysons donc ces deux éléments dans cette sous section. En effet, il doit y avoir une quantité suffisante de données injectées dans le processus itératif d'auto-apprentissage, tout en garantissant un taux d'erreur le plus faible possible. La vérification lexicale ainsi que la cohorte nous permettent d'obtenir ces deux propriétés.

Dans la figure 5.6, nous étudions la qualité des données étiquetées en nous intéressant à la proportion des longueurs de mots dans différentes bases. La figure 5.6a présente les proportions de mots de diverses longueurs pour la base de mots non-étiquetés contenant 41k mots. On peut voir que les mots de longueur 4 ou inférieure représentent un peu plus de 50% de la base alors qu'ils représentent quasiment 75% de la base auto-étiquetée à l'issue de l'itération 0 (figure 5.6b). Cette observation montre bien que le réseau LSTM avec notre règle de réapprentissage basée sur la vérification lexicale valide d'abord les mots courts qui sont les plus faciles à apprendre. La forte proportion de mots courts est aussi due au fait que même en erreur ils ont une probabilité plus élevée d'appartenir au lexique. Grâce à l'application du principe de cohorte, la proportion de mots courts est

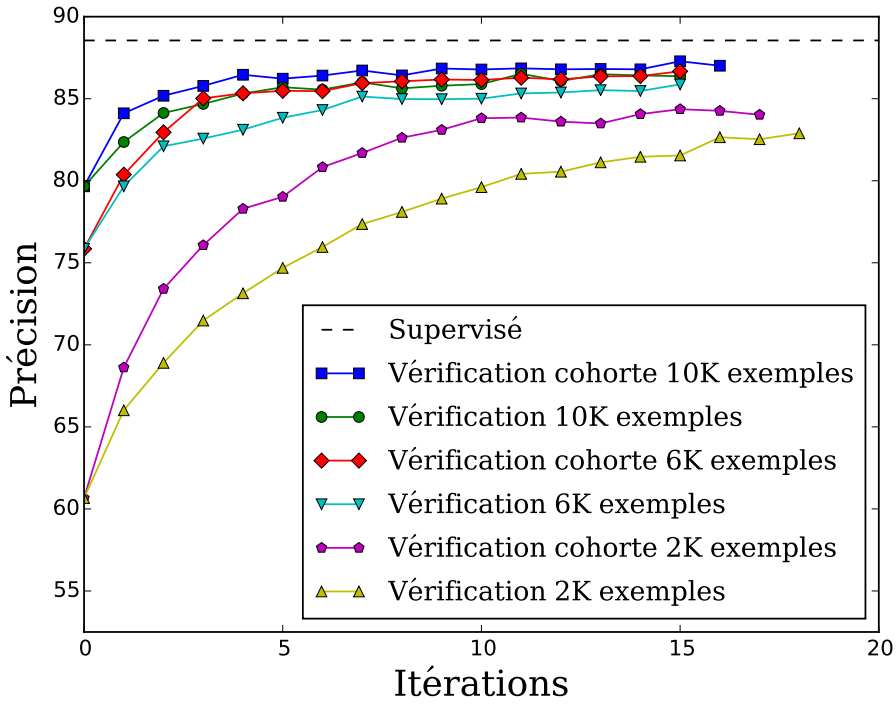
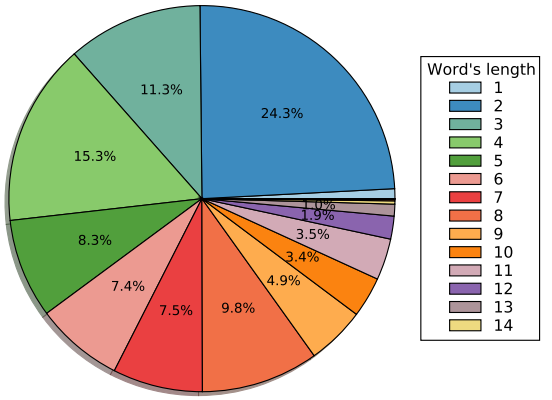


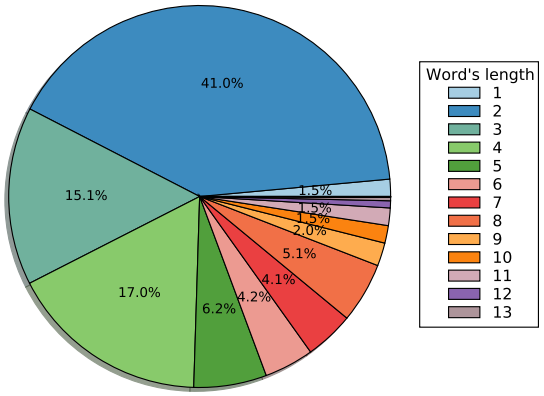
FIGURE 5.5 – Taux d'erreur caractère (CER) obtenus pour notre stratégie d'apprentissage semi-supervisé avec vérification lexicale et avec ou sans cohorte en fonction des itérations pour différents nombres d'exemples étiquetés.

abaissée à 68% à l'itération 0 (cf figure 5.6c) montrant une plus grande capacité à identifier des mots longs dès le début tout en diminuant les erreurs sur les mots courts. Notre stratégie est efficace car au fil des itérations, de plus en plus de mots longs, supposés plus difficiles à reconnaître, sont auto-étiquetés avec succès. A l'itération 15, comme montré sur la figure 5.6d, on peut voir que la proportion de mots courts a été significativement réduite représentant environ 57% des mots. La distribution en fin d'auto-apprentissage avec vérification et cohorte (figure 5.6d) est proche de la distribution initiale (figure 5.6a). Ces diagrammes montrent bien que notre règle de réapprentissage permet d'introduire de plus en plus de mots longs perçus comme plus difficiles à reconnaître, permettant ainsi l'amélioration des performances itérativement.

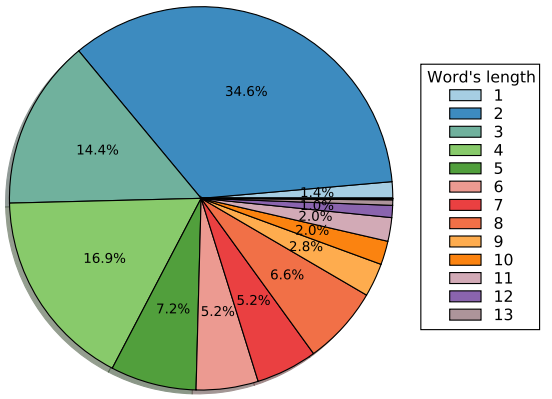
Un autre indicateur de la qualité de la base d'images auto-étiquetées est le taux d'erreur caractère que l'on peut calculer grâce à la vérité terrain de la base RIMES. On observe sur la figure 5.7a que le taux d'erreur caractère est globalement stable au cours de l'apprentissage, sauf pour le lexique gigantesque pour lequel le CER décroît au début quelle que soit la stratégie avec ou sans le principe de cohorte. L'augmentation de la taille du lexique augmente ce taux d'erreur faisant décroître les performances comme nous avons pu le voir précédemment. Sur la figure 5.7b, l'augmentation du seuil du nombre d'accord avec la cohorte permet de réduire sensiblement le taux d'erreur caractère. On peut aussi noter que la cohorte donne un taux d'erreur légèrement supérieur à celui de la vérification seule. Sur la figure 5.7a, on voit que l'accord produit un taux d'erreur assez faible, alors



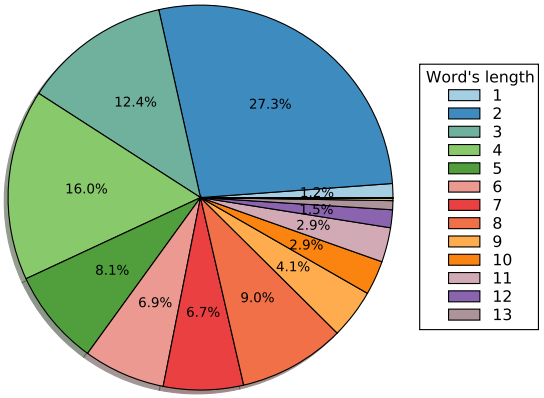
(a) Pourcentages de mots dans la base sans étiquettes.



(b) Pourcentages de mots dans la base auto-étiquetée à l'issue de la première itération avec juste la vérification.



(c) Pourcentages de mots dans la base auto-étiquetée à l'issue de la première itération avec la vérification et la cohorte.

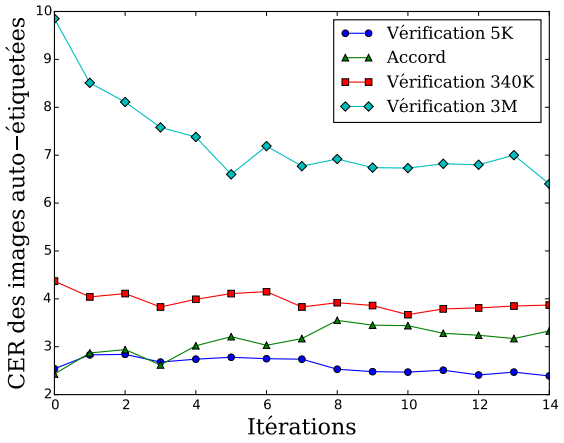


(d) Pourcentages de mots dans la base auto-étiquetée à l'issue de la dernière itération avec la vérification et la cohorte.

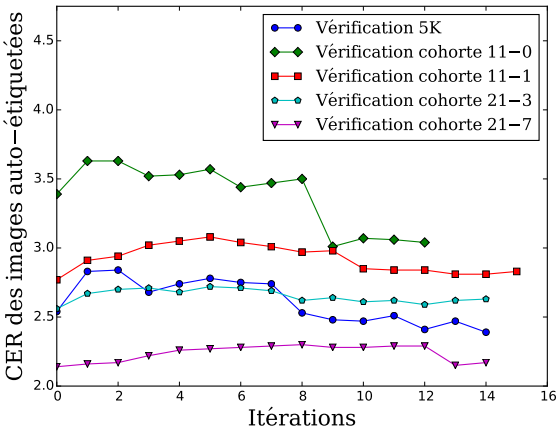
FIGURE 5.6 – Comparaison des pourcentages de nombre de mots pour différentes parties de la base.

que nous avons pu voir que les performances étaient en dessous des autres méthodes. Nous en déduisons donc que la quantité de données joue également un rôle prépondérant.

En effet, la quantité de données a aussi son importance, en regardant les courbes figure 5.8 on s'aperçoit que la proportion d'images augmente avec les itérations apportant toujours de nouvelles données au réseau. En regardant la figure 5.8a, on peut voir que la règle d'accord étiquette beaucoup moins de données que notre stratégie avec vérification seule d'où ses faibles performances. Sur la figure 5.8b, on voit que l'ajout du principe de cohorte permet d'augmenter sensiblement la quantité de données étiquetées. Bien que l'erreur apportée soit plus grande comme observé précédemment, l'ajout du principe de cohorte compense l'augmentation du CER par un plus grand nombre d'exemples auto-étiquetés. Cette affirmation se vérifie avec les différents seuils testés pour lesquels les performances sont similaires, par exemple sans seuil le taux d'erreur est plus élevé. Cepen-



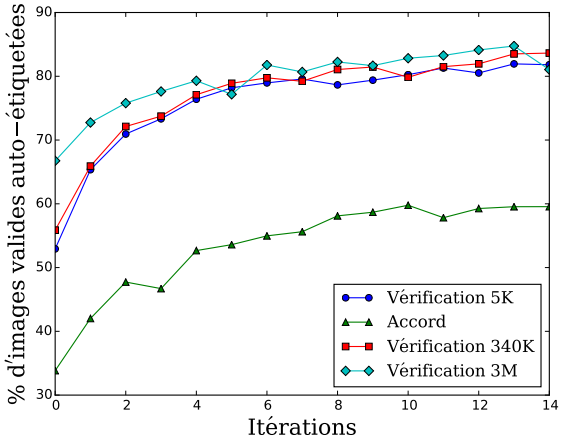
(a) Vérification 5k, 340k, 3M sont les résultats de notre méthode pour 3 tailles de lexique différentes.



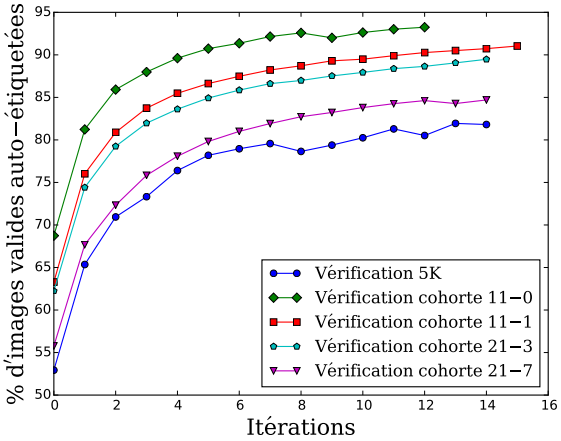
(b) Comparaison entre notre méthode sans cohorte et avec cohorte et différents paramètres pour un lexique de 5k.

FIGURE 5.7 – Taux d'erreur caractère des images auto-étiquetées par itération.

dant, cette augmentation des erreurs est compensée par un plus grand nombre d'images auto-étiquetées.



(a) Vérification 5k, 340k, 3M sont les résultats de notre méthode pour 3 tailles de lexique différentes.



(b) Comparaison entre notre méthode sans cohorte et avec cohorte et différents seuils pour un lexique de 5k.

FIGURE 5.8 – Pourcentage d'images auto-étiquetées par itération.

Étant donné le pourcentage d'images auto-étiquetées et le taux d'erreur caractères obtenu sur celles-ci, il est naturel de se demander s'il n'existe pas un seuil meilleur que les autres. Sur la figure 5.9, nous représentons le pourcentage d'images auto-étiquetées en fonction du taux d'erreur caractères sur celles-ci pour différents seuils à chaque itération. On peut voir qu'il n'y a aucun point de fonctionnement dominant sur les différentes courbes (plus petit CER et plus grand % à la fois), corroborant ainsi le fait que les

réseaux appris avec différents seuils ont des performances très proches (écart type de 0.22). On peut néanmoins conclure que le point de fonctionnement optimum (meilleur combinaison de CER et %) de notre méthode sans cohorte se situe dans la concavité de la courbe ROC formée par les points optimum de notre méthode avec cohorte pour les différents seuils. Cette analyse montre bien l'intérêt d'utiliser le principe de cohorte.

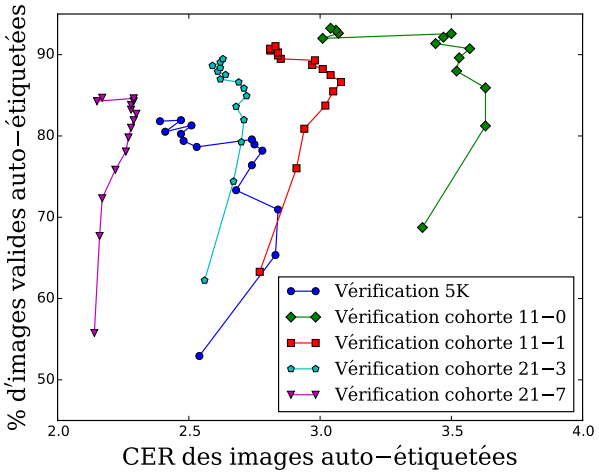
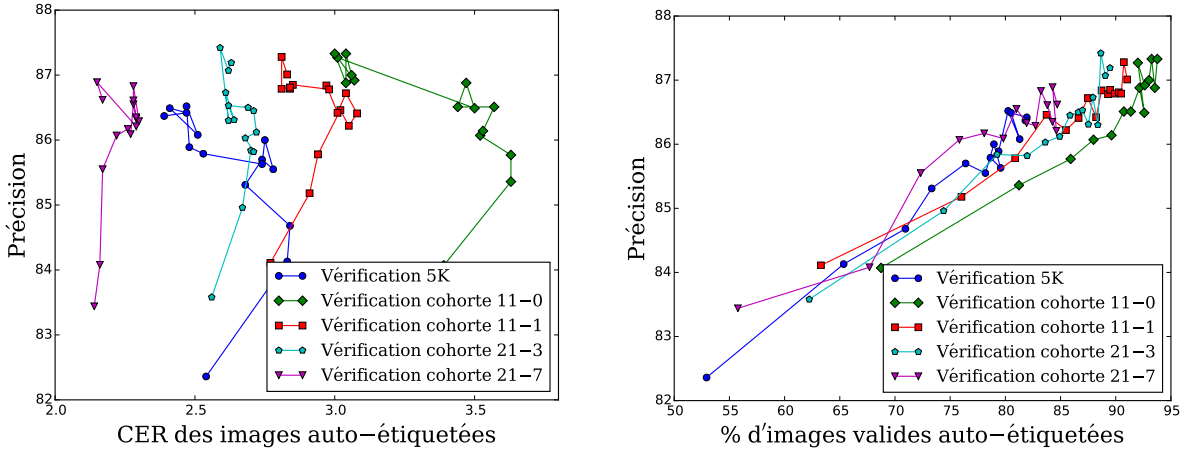


FIGURE 5.9 – Pourcentage d'images auto-étiquetées en fonction du CER.

En se demandant si un point de fonctionnement serait meilleur que les autres, une des questions sous-jacentes était de savoir s'il était plus important d'avoir un CER bas au détriment du pourcentage auto-étiqueté ou inversement. La figure 5.9 ne permettant pas d'y répondre, nous avons donc tracé sur la figure 5.10 la précision en fonction du CER d'un côté et du pourcentage de l'autre. On peut voir sur la figure 5.10a que la



(a) Précision en fonction du CER.

(b) Précision en fonction du pourcentage d'images auto-étiquetées.

FIGURE 5.10 – Effet du CER et du pourcentages des images auto-étiquetées sur la précision.

précision augmente même si le CER sur les images auto-étiquetées augmente ou stagne. Par contre, sur la figure 5.10b, on peut voir que l'augmentation du pourcentage d'images auto-étiquetées se traduit généralement par une augmentation de la précision. On ne peut pas dire qu'il faut privilégier le CER des images auto-étiquetées au détriment du pourcentage d'images auto-étiquetées et inversement. Cependant, le pourcentage d'images auto-étiquetés semble plus important.

Pour connaître quel critère favoriser, une solution serait de réaliser de très nombreux apprentissages en fonction du nombre d'exemples et du taux d'erreur sur la base.

5.5 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthode d'auto-apprentissage basée sur la vérification pour l'apprentissage de réseau LSTM dédié à la reconnaissance de l'écriture. Cette méthode est fondée sur un seul réseau LSTM et ne nécessite pas de paramètres supplémentaires contrairement aux méthodes actuellement présentées dans l'état de l'art. La vérification lexicale permet de réaliser l'auto-apprentissage à partir de n'importe quelle taille de lexique. Le seul prérequis étant d'obtenir un lexique de la langue à reconnaître qui couvre suffisamment le lexique de la base à étiqueter.

Nous améliorons cette méthode d'auto-apprentissage en appliquant le principe de cohorte qui permet d'augmenter le nombre d'images auto-étiquetées tout en maintenant un faible taux d'erreur. La passe en avant des exemples non étiquetés est réalisée pour cet ensemble de réseaux afin de procéder à un vote à la majorité en plus de la vérification pour déterminer la bonne étiquette. Ce moyen a permis de diminuer le taux d'erreur ou d'augmenter le nombre d'exemples auto-étiquetés. Cet ajout permet d'améliorer significativement les performances de notre méthode, notamment lorsque le nombre de données étiquetées est faible.

L'auto-apprentissage proposé dans ce chapitre permet d'obtenir des résultats très proches d'un apprentissage entièrement supervisé sur la base RIMES et IAM. En comparant notre méthode aux autres méthodes à l'état de l'art en s'intéressant à l'amélioration en % et le % vers supervisé, on observe que les réseaux appris par notre méthode obtiennent de meilleurs résultats.

La perspective principale de ce travail serait de réaliser un auto-apprentissage basé sur des lignes de textes et non plus des mots isolés à l'image du passage de la cascade à notre combinaison LV-ROVER.

Conclusions et perspectives

Le sujet d'études de cette thèse réside dans l'apprentissage profond de réseaux de neurones récurrents pour la reconnaissance de l'écriture manuscrite.

Comme nous l'avons vu à travers le chapitre 1, les réseaux de neurones sont une méthode d'apprentissage proposant de nombreuses architectures permettant de résoudre de multiples tâches comme la classification ou la régression. Aujourd'hui l'apprentissage profond de réseaux de neurones est une discipline en plein essor qui a permis d'attaquer des problèmes complexes que nous ne pouvions jusqu'à présent pas résoudre avec autant d'efficacité. Les réseaux de neurones profonds ainsi que le développement de librairies permettant de les apprendre en temps raisonnable ont facilité l'application des méthodes d'apprentissage neuronales. Parmi les nombreux types de réseaux de neurones, nous avons particulièrement analysé le fonctionnement des réseaux de neurones récurrents à cellules LSTM, car leurs propriétés et notamment leurs mémoires sont très intéressantes pour la reconnaissance de l'écriture.

En effet, la reconnaissance de l'écriture est l'intérêt applicatif de cette thèse. Nous cherchons à améliorer cette reconnaissance à travers de nouvelles méthodes. Nous avons étudié l'historique des méthodes de reconnaissance de l'écriture sur les 20 dernières années. Nous avons vu à travers le chapitre 2 que la grande majorité des méthodes suivent le même paradigme de reconnaissance. La reconnaissance de l'écriture est composée d'une reconnaissance de caractères suivie d'un décodage dirigé par le lexique intégrant une modélisation de la langue. Dans ce paradigme, il est difficile de se passer de l'incontournable reconnaissance de caractères. Il est en effet impossible d'apprendre à reconnaître des formes de lignes de texte et de mots tant la combinatoire est grande. Pour fiabiliser la reconnaissance de caractères, celle-ci est, dans la majorité des méthodes et notamment celles à l'état de l'art [Voigtlaender et al., 2016], accompagnée d'un décodage dirigé par le lexique. Les décodages dirigés par le lexique ont deux inconvénients : leur sensibilité à la taille du lexique et ainsi que des problèmes liés aux mots hors vocabulaire, ce même avec une modélisation de la langue. D'une part, des progrès significatifs ont été réalisés sur les systèmes de reconnaissance de caractères qui utilisent des réseaux LSTM ayant des performances élevées. D'une autre part, les méthodes à l'état de l'art suivent toujours le même paradigme de reconnaissance basé sur un décodage dirigé par le lexique.

À partir de ces analyses, nous nous sommes interrogés sur la façon d'améliorer la reconnaissance de l'écriture avec des réseaux de neurones récurrents. De cette réflexion sont nées, dans l'introduction, des questions liées au paradigme de reconnaissance de l'écriture, aux réseaux de neurones récurrents, à leurs propriétés et leur apprentissage. À travers les chapitres 3, 4 et 5, nous avons tenté d'apporter des réponses à ces questionnements dans le but d'améliorer la reconnaissance de l'écriture manuscrite.

Dans un premier temps, notre réflexion s'est portée sur la façon d'*utiliser un réseau LSTM sans décodage dirigé par le lexique afin de contourner les problèmes liés aux*

lexiques.

Ayant constaté les bonnes performances de reconnaissance d'un réseau LSTM seul, nous avons proposé de remplacer le décodage dirigé par le lexique par une **vérification lexicale** (sous-section 3.3.2). Elle consiste à accepter le résultat de reconnaissance d'un classifieur si celui-ci appartient au lexique. La vérification lexicale permet d'utiliser le lexique sans contraindre le résultat de la reconnaissance à celui-ci. Nous avons montré la fiabilité de la vérification lexicale et apporté des éléments pour la renforcer comme le nombre minimum d'accord de décisions. Nos expérimentations ont montré que nos méthodes étaient très peu sensibles à la taille du lexique. La vérification lexicale permet notamment d'utiliser des lexiques gigantesques contenant des millions de mots sans impacter le temps de calcul ni trop dégrader les performances. La vérification lexicale est un élément important du nouveau paradigme de reconnaissance que nous proposons dans cette thèse.

Cependant, la vérification lexicale seule ne permet pas d'obtenir des résultats à l'état de l'art et présente notamment un rejet des mots non vérifiés. C'est la raison pour laquelle nous avons introduit une nouvelle méthode pour obtenir un grand nombre de réseaux LSTM complémentaires (appelé cohorte) afin de fiabiliser la reconnaissance par leur combinaison.

Ces réseaux à combiner doivent produire des résultats différents et être complémentaires, sans quoi la combinaison ne produira pas de meilleurs résultats. Nous avons alors logiquement soulevé l'interrogation suivante : "*Comment obtenir facilement et rapidement des reconnaisseurs (réseaux LSTM) ayant une bonne complémentarité afin de les combiner ?*".

Nous avons présenté dans la sous-section 3.3.3 une méthode permettant de générer des centaines de réseaux de neurones récurrents LSTM complémentaires au cours d'un seul apprentissage appelés cohorte. Cette méthode a pour principe de récupérer un réseau à chaque itération et se base sur les travaux de [Choromanska et al., 2015]. Afin de générer une cohorte, il suffit de prendre des paramètres de taux d'apprentissage et de momentum permettant de ne pas rester dans un minimum local. Nous observons et montrons la complémentarité des réseaux issus d'une cohorte à travers les expérimentations de la partie II. Nous montrons particulièrement cette complémentarité grâce à l'analyse réalisée dans la sous-section 3.4.4. Cette complémentarité transparaît dans les résultats de nos méthodes de combinaisons où l'augmentation du nombre de réseaux d'une cohorte se traduit par un gain de performances.

À partir de nos deux propositions que sont la **cohorte** et la **Vérification lexicale**, la question "*Comment combiner efficacement un grand nombre (des centaines voir des milliers) de réseaux LSTM afin d'améliorer les performances ?*" s'est posée.

Dans le chapitre 3, nous proposons une combinaison en **cascade** pour la reconnaissance de mots isolés qui utilise le principe de vérification lexicale comme mécanisme de rejet. Cette cascade présente un schéma facile à mettre en œuvre où les rejets d'un réseau LSTM après vérification lexicale sont donnés en entrée d'un nouveau réseau LSTM. En utilisant le principe de cohorte, nous avons réalisé une combinaison en cascade de milliers de réseaux

sur deux bases publiques RIMES et IAM. Sur ces deux bases, nous obtenons des résultats à l'état de l'art grâce à l'utilisation du principe de cohorte. La cascade montre également son efficacité lorsqu'il s'agit de traiter des lexiques gigantesques ; ce qui n'a jamais été réalisé dans la littérature auparavant.

Ces résultats nous ont confortés quant à l'efficacité de l'utilisation de la cohorte et de la vérification lexicale. Nous avons donc étendu ce principe à la reconnaissance de lignes de texte qui est plus difficile mais aussi plus intéressante à réaliser. Pour y parvenir, nous avons proposé une version modifiée de l'algorithme de combinaison à l'état de l'art ROVER. Nous proposons un algorithme **LV-ROVER** pour Lexicon Verified ROVER qui permet de combiner rapidement des centaines de réseaux LSTM sans décodage dirigé par le lexique avec une complexité réduite par rapport à ROVER. Nous obtenons des résultats à l'état de l'art pour RIMES et des résultats encourageants pour IAM compte tenu du contexte de reconnaissance plus complexe.

Notre dernier chapitre s'articule autour du questionnement suivant : "*Comment apprendre des réseaux LSTM pour la reconnaissance de l'écriture avec un nombre restreint de données en entrée ?*".

Nous avons constaté dans le chapitre 1 que l'apprentissage d'un réseau de neurones nécessite une grande quantité de données étiquetées souvent coûteuses à obtenir. Nous proposons dans le chapitre 5 une méthode d'**auto-apprentissage** permettant d'apprendre un réseau LSTM par lui même pour la reconnaissance de mots isolés. Notre méthode est quasiment insensible aux paramètres. Elle se sert de la vérification lexicale mais aussi du principe de cohorte. Le principe est d'auto-étiqueter les exemples dont les prédictions d'un ou plusieurs BLSTM d'une cohorte appartiennent au lexique. Nous montrons, dans nos expérimentations sur RIMES et IAM, l'efficacité de cette méthode par rapport à d'autres méthodes de l'état de l'art.

Dans cette thèse, nous proposons un nouveau paradigme de reconnaissance où nous considérons autrement l'information donnée par la reconnaissance de caractères. Les méthodes traditionnelles considèrent la reconnaissance de caractère comme incertaine. Elles utilisent des techniques de décodage dirigé par le lexique coûteuses en temps de calcul et en pratique inutilisables pour de très grands lexiques pour interpréter les probabilités données par le reconnaisseur. Dans cette thèse, nous proposons de réaliser un décodage très rapide donnant le caractère de probabilité maximale pour chaque trame. Nous ne travaillons plus à partir des probabilités mais à partir d'une chaîne de caractères afin de réaliser une vérification lexicale quasi instantanée et ce quelle que soit la taille du lexique. Cette méthode nous permet donc de combiner, grâce à notre principe clé de cohorte, un très grand nombre de reconnaisseurs ayant une fiabilité élevée dans des temps de calculs raisonnables tout en atteignant l'état de l'art.

Durant cette thèse, de nombreuses expérimentations et travaux complémentaires ont été réalisés. Ils ne sont pas présentés ici car conduits dans le cadre industriel de la thèse CIFRE ou ne présentant qu'un intérêt scientifique limité. Parmi ces travaux, il y a notamment : le développement d'une librairie de décodage de réseaux de neurones (récur-

rents notamment) plus performante que certaines librairies de réseaux de neurones comme RNNLIB et tensorflow ; des expérimentations sur les architectures des réseaux LSTM et les paramètres d'apprentissage ; des expérimentations sur la pénalisation de l'erreur CTC en fonction du type de l'erreur (insertion, délétion, substitution) ; des expérimentations utilisant la cohorte, la vérification lexicale, nos combinaisons ou notre auto-apprentissage sur des bases privées. Tous ces éléments non présentés ont permis soit de monter en compétence sur les réseaux de neurones et en particulier les RNN LSTM, soit d'initier des idées qui ont donné lieu à nos contributions, soit afin de valider nos méthodes en condition réelle.

Il y a de nombreuses perspectives pour ces travaux. Une première perspective serait de porter le principe d'auto-apprentissage que nous avons développé pour la reconnaissance de lignes de texte. En effet, la reconnaissance de lignes est bien plus intéressante que celle de mots isolés. Il faudrait alors trouver une méthode permettant de les auto-étiqueter sans segmentation en mots. Il serait également intéressant d'étudier de manière exhaustive les performances d'une architecture de réseau LSTM en fonction du nombre d'exemples et du taux d'erreur sur un jeu de données que ce soit dans le cadre d'un apprentissage semi-supervisé, d'une campagne d'annotation réalisée automatiquement (autres experts) ou humainement (rigueur demandée à l'annotateur et nombre de vérifications).

D'un point de vue méthodologique, il serait aussi très enrichissant d'étudier la généricité de la génération de cohorte et d'analyser son efficacité en utilisant d'autres architectures de réseaux de neurones tel que les réseaux convolutifs. Il faudrait aussi étudier l'utilité de la cohorte pour d'autres applications comme des applications de vision par ordinateur. Cependant le temps et les moyens nécessaires n'ont pas permis de réaliser cette étude au cours de cette thèse.

Une autre perspective résiderait dans l'étude de la réduction de la taille d'une cohorte quel que soit le réseau et l'application. En effet, le grand nombre de réseaux peut être un frein à certaines applications ; il faudrait être capable de compresser toute la complémentarité de cette centaine de réseaux en un seul. C'est un sujet de recherche qui peut être connexe à celui de la réduction de taille d'un réseau de neurones en un plus petit et plus rapide tout en conservant ses performances.

Dans le domaine la reconnaissance de l'écriture, des progrès sont encore possibles, bien que des avancements récents aient déjà été réalisés sur le modèle optique avec des réseaux combinant convolutions et récurrences [Voigtlaender et al., 2016]. Ces progrès se tournent désormais vers les modélisations de la langue. Cependant, nous avons montré qu'avec les performances des réseaux récurrents LSTM et une vérification lexicale, il était possible d'atteindre des résultats à l'état de l'art. Les progrès que nous avons opérés sur la reconnaissance de l'écriture amenuisent l'intérêt d'utiliser des méthodes contraignant le résultat par un décodage dirigé par un lexique ou utilisant une modélisation de la langue. Un idéal de système de reconnaissance serait de réussir à apprendre de manière non-supervisée un réseau de neurones sur une très grande quantité de données lui permettant de reconnaître et modéliser directement l'écriture dans plusieurs langues.

Liste des publications

Articles acceptés :

Stuner, B., Chatelain, C., and Paquet, T. (2016a). Cascade de réseaux blstm vérifiée par le lexique pour la reconnaissance d'écriture. In *Reconnaissance de Formes et Intelligence Artificielle.*

Stuner, B., Chatelain, C., and Paquet, T. (2016d). A lexicon verification strategy in a blstm cascade framework. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 234–239. IEEE

Stuner, B., Chatelain, C., and Paquet, T. (2016b). Cascading blstm networks for handwritten word recognition. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 3416–3421. IEEE

Stuner, B., Chatelain, C., and Paquet, T. (2017b). Self-training of blstm with lexicon verification for handwriting recognition. In *Document Analysis and Recognition (ICDAR), 2017 14th International Conference on*. IEEE

Articles soumis :

Stuner, B., Chatelain, C., and Paquet, T. (2016c). Cohort of lstm and lexicon verification for handwriting recognition with gigantic lexicon. *arXiv preprint arXiv:1612.07528*

Stuner, B., Chatelain, C., and Paquet, T. (2017a). Lv-rover: Lexicon verified recognizer output voting error reduction. *arXiv preprint arXiv:1707.07432*

Bibliographie

[Abu-Mostafa, 1995] Abu-Mostafa, Y. S. (1995). Machines that learn from hints. *Scientific American*, 272(4) :64–69.

[Ackley et al., 1985] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines*. *Cognitive science*, 9(1) :147–169.

[Ait-Mohand et al., 2014] Ait-Mohand, K., Paquet, T., and Ragot, N. (2014). Combining structure and parameter adaptation of hmms for printed text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 36(9) :1716–1732.

[Anderson et al., 2017] Anderson, P., He, X., Buehler, C., Teney, D., Johnson, M., Gould, S., and Zhang, L. (2017). Bottom-up and top-down attention for image captioning and vqa. *arXiv preprint arXiv :1707.07998*.

[Antonacopoulos et al., 2011] Antonacopoulos, A., Clausner, C., Papadopoulos, C., and Pletschacher, S. (2011). Historical document layout analysis competition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1516–1520. IEEE.

[Augustin et al., 2006] Augustin, E., Carré, M., Grosicki, E., Brodin, J.-M., Geoffrois, E., and Prêteux, F. (2006). Rimes evaluation campaign for handwritten mail processing. In *International Workshop on Frontiers in Handwriting Recognition (IWFHR'06),*, pages 231–235.

[Ball and Srihari, 2009] Ball, G. R. and Srihari, S. N. (2009). Semi-supervised learning for handwriting recognition. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 26–30. IEEE.

[Barlas et al., 2014] Barlas, P., Adam, S., Chatelain, C., and Paquet, T. (2014). A typed and handwritten text block segmentation system for heterogeneous and complex documents. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 46–50. IEEE.

[Bauer, 1993] Bauer, L. (1993). *Manual of information to accompany the Wellington corpus of written New Zealand English*. Department of Linguistics, Victoria University of Wellington Wellington.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf : Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417.

[Belharbi et al., 2017] Belharbi, S., Chatelain, C., Hérault, R., Adam, S., Thureau, S., Chastan, M., and Modzelewski, R. (2017). Spotting l3 slice in ct scans using deep convolutional network and transfer learning. *Computers in Biology and Medicine*.

[Bellman, 1956] Bellman, R. (1956). Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10) :767–769.

[Bellman, 1962] Bellman, R. E. (1962). *Adaptive control processes : a guided tour*. Princeton university press.

[Bengio et al., 1995] Bengio, Y., LeCun, Y., Nohl, C., and Burges, C. (1995). Lerec : A nn/hmm hybrid for on-line handwriting recognition. *Neural Computation*, 7(6) :1289–1303.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2) :157–166.

[Berkson, 1944] Berkson, J. (1944). Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227) :357–365.

[Bertolami and Bunke, 2005] Bertolami, R. and Bunke, H. (2005). Ensemble methods for handwritten text line recognition systems. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 3, pages 2334–2339. IEEE.

[Bharath and Madhvanath, 2012] Bharath, A. and Madhvanath, S. (2012). Hmm-based lexicon-driven and lexicon-free word recognition for online handwritten indic scripts. *IEEE PAMI*, 34(4) :670–682.

[Bideault et al., 2015] Bideault, G., Mioulet, L., Chatelain, C., and Paquet, T. (2015). Spotting handwritten words and regex using a two stage blstm-hmm architecture. In *Document Recognition and Retrieval*.

[Bluche, 2015] Bluche, T. (2015). *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition*. PhD thesis, Université Paris Sud-Paris XI.

[Bluche et al., 2014a] Bluche, T., Louradour, J., Knibbe, M., Moysset, B., Benzeghiba, M. F., and Kermorvant, C. (2014a). The a2ia arabic handwritten text recognition system at the open hart2013 evaluation. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 161–165. IEEE.

[Bluche et al., 2016] Bluche, T., Louradour, J., and Messina, R. (2016). Scan, attend and read : End-to-end handwritten paragraph recognition with mdlstm attention. *arXiv preprint arXiv :1604.03286*.

[Bluche et al., 2014b] Bluche, T., Ney, H., and Kermorvant, C. (2014b). A comparison of sequence-trained deep neural networks and recurrent neural networks optical modeling for handwriting recognition. In *International Conference on Statistical Language and Speech Processing*, pages 199–210. Springer.

[Blum and Chawla, 2001] Blum, A. and Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts.

[Blum and Mitchell, 1998] Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM.

[Bottou, 1998] Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9) :142.

[Bottou, 2012] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks : Tricks of the trade*, pages 421–436. Springer.

[Bozinovic and Srihari, 1989] Bozinovic, R. M. and Srihari, S. N. (1989). Off-line cursive script word recognition. *IEEE Transactions on pattern analysis and machine intelligence*, 11(1) :68–83.

[Brakensiek et al., 2002] Brakensiek, A., Rottland, J., and Rigoll, G. (2002). Handwritten address recognition with open vocabulary using character n-grams. In *International Workshop on Frontiers in Handwriting Recognition*, pages 357–362.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1) :5–32.

[Breuel, 2002] Breuel, T. M. (2002). Two geometric algorithms for layout analysis. In *Document analysis systems v*, pages 188–199. Springer.

[Brown et al., 1992] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4) :467–479.

[Brunessaux et al., 2014] Brunessaux, S., Giroux, P., Grilheres, B., Manta, M., Bodin, M., Choukri, K., Galibert, O., and Kahn, J. (2014). The maurdor project : Improving automatic processing of digital documents. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 349–354. IEEE.

[Bunke et al., 2004] Bunke, H., Bengio, S., and Vinciarelli, A. (2004). Offline recognition of unconstrained handwritten texts using hmms and statistical language models. *IEEE PAMI*, 26(6) :709–720.

[Casey and Lecolinet, 1996] Casey, R. G. and Lecolinet, E. (1996). A survey of methods and strategies in character segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 18(7) :690–706.

[Chapelle et al., 2009] Chapelle, O., Scholkopf, B., and Zien, A. (2009). Semi-supervised learning (chapelle, o. et al., eds. ; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3) :542–542.

[Chatelain et al., 2006a] Chatelain, C., Heutte, L., and Paquet, T. (2006a). Segmentation-driven recognition applied to numerical field extraction from handwritten incoming mail documents. In *Document Analysis System, LNCS 3872, Springer*, pages 564–575.

[Chatelain et al., 2006b] Chatelain, C., Heutte, L., and Paquet, T. (2006b). A two-stage outlier rejection strategy for numerical field extraction in handwritten documents. In *ICPR*, volume 3, pages 224–227.

[Chelba et al., 2013] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv :1312.3005*.

[Chen et al., 2017] Chen, K., Seuret, M., Hennebert, J., and Ingold, R. (2017). Convolutional neural networks for page segmentation of historical document images. In *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, volume 1, pages 965–970. IEEE.

[Chen et al., 1995] Chen, M.-Y., Kundu, A., and Srihari, S. N. (1995). Variable duration hidden markov model and morphological segmentation for handwritten word recognition. *IEEE transactions on image processing*, 4(12) :1675–1688.

[Chen and Goodman, 1996] Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation : Encoder-decoder approaches. *arXiv preprint arXiv :1409.1259*.

[Choromanska et al., 2015] Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks. In *AISTATS*.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv :1412.3555*.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3) :273–297.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1) :21–27.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4) :303–314.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *CVPR*, volume 1, pages 886–893. IEEE.

[Davis, 1991] Davis, L. (1991). Handbook of genetic algorithms.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet : A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE.

[Diem et al., 2017] Diem, M., Kleber, F., Fiel, S., Grüning, T., and Gatos, B. (2017). cbad : Icdar2017 competition on baseline detection. In *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, volume 1, pages 1355–1360. IEEE.

[Donahue et al., 2015] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul) :2121–2159.

[Earnest, 1960] Earnest, L. D. (1960). *A Boolean matrix processor with applications to handwriting recognition*. PhD thesis, Massachusetts Institute of Technology.

[Edelman et al., 1990] Edelman, S., Flash, T., and Ullman, S. (1990). Reading cursive handwriting by alignment of letter prototypes. *International Journal of Computer Vision*, 5(3) :303–331.

[El-Yacoubi et al., 1999] El-Yacoubi, A., Gilloux, M., Sabourin, R., and Suen, C. Y. (1999). An hmm-based approach for off-line unconstrained handwritten word modeling and recognition. *IEEE PAMI*, 21(8) :752–760.

[El-Yacoubi et al., 2002] El-Yacoubi, M. A., Gilloux, M., and Bertille, J.-M. (2002). A statistical approach for phrase location and recognition within a text line : An application to street name recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2) :172–188.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2) :179–211.

[Favata and Srikantan, 1996] Favata, J. T. and Srikantan, G. (1996). A multiple feature/resolution approach to handprinted digit and character recognition. *International journal of imaging systems and technology*, 7(4) :304–311.

[Fischer et al., 2010] Fischer, A., Keller, A., Frinken, V., and Bunke, H. (2010). Hmm-based word spotting in handwritten documents using subword models. In *Pattern recognition (icpr), 2010 20th international conference on*, pages 3416–3419. IEEE.

[Fiscus, 1997] Fiscus, J. G. (1997). A post-processing system to yield reduced word error rates : Recognizer output voting error reduction (rover). In *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, pages 347–354. IEEE.

[Fissore et al., 1988] Fissore, L., Micca, G., Pieraccini, R., and Palace, P. (1988). Strategies for lexical access to very large vocabularies. *Speech Communication*, 7(4) :355–366.

[Francis, 1971] Francis, W. N. (1971). *A manual of information to accompany A standard sample of present-day edited American English, for use with digital computers.* Department of Linguistics, Brown University.

[Frinken and Bunke, 2009] Frinken, V. and Bunke, H. (2009). Evaluating retraining rules for semi-supervised learning in neural network based cursive word recognition. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 31–35. IEEE.

[Frinken et al., 2014] Frinken, V., Fischer, A., Baumgartner, M., and Bunke, H. (2014). Keyword spotting for self-training of blstm nn based handwriting recognition systems. *Pattern Recognition*, 47(3) :1073–1082.

[Frinken et al., 2011] Frinken, V., Fischer, A., Bunke, H., and Foornes, A. (2011). Co-training for handwritten word recognition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 314–318. IEEE.

[Frinken et al., 2012] Frinken, V., Fischer, A., Manmatha, R., and Bunke, H. (2012). A novel word spotting method based on recurrent neural networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2) :211–224.

[Fukushima and Miyake, 1982] Fukushima, K. and Miyake, S. (1982). Neocognitron : A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer.

[Gatos et al., 2006] Gatos, B., Pratikakis, I., and Perantonis, S. J. (2006). Adaptive degraded document image binarization. *Pattern recognition*, 39(3) :317–327.

[Gers et al., 2000] Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget : Continual prediction with lstm. *Neural computation*, 12(10) :2451–2471.

[Gers et al., 2003] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2003). Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research*, 3 :115–143.

[Gilloux, 1993] Gilloux, M. (1993). Research into the new generation of character and mailing address recognition systems at the french post office research center. *Pattern Recognition Letters*, 14(4) :267–276.

[Gilloux, 1998] Gilloux, M. (1998). Réduction dynamique du lexique par la méthode tabou. In *Colloque International Francophone sur l'ecrit et le Document*, pages 24–31.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256.

[Gorski et al., 1999] Gorski, N., Anisimov, V., Augustin, E., Baret, O., Price, D., and Simon, J.-C. (1999). A2ia check reader : A family of bank check recognition systems. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pages 523–526. IEEE.

[Graves, 2012] Graves, A. (2012). *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer.

[Graves, 2013a] Graves, A. (2013a). Generating sequences with recurrent neural networks. *arXiv preprint arXiv :1308.0850*.

[Graves, 2013b] Graves, A. (2013b). Rnnlib : A recurrent neural network library for sequence learning problems. https://sourceforge.net/projects/rnnl.

[Graves et al., 2009] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5) :855–868.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5) :602–610.

[Graves and Schmidhuber, 2009] Graves, A. and Schmidhuber, J. (2009). Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, pages 545–552.

[Grosicki and El Abed, 2009] Grosicki, E. and El Abed, H. (2009). Icdar 2009 handwriting recognition competition. In *ICDAR*, pages 1398–1402.

[Grosicki and El-Abed, 2011] Grosicki, E. and El-Abed, H. (2011). Icdar 2011-french handwriting recognition competition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1459–1463. IEEE.

[Guillevic and Suen, 1995] Guillevic, D. and Suen, C. Y. (1995). Cursive script recognition applied to the processing of bank cheques. In *Document Analysis and Recognition,*

1995., Proceedings of the Third International Conference on, volume 1, pages 11–14. IEEE.

[Guillevic and Suen, 1998] Guillevic, D. and Suen, C. Y. (1998). Recognition of legal amounts on bank cheques. *Pattern Analysis and Applications*, 1(1) :28–41.

[Guthrie et al., 2006] Guthrie, D., Allison, B., Liu, W., Guthrie, L., and Wilks, Y. (2006). A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pages 1–4. sn.

[Hamdani et al., 2014] Hamdani, M., Doetsch, P., Kozielski, M., Mousa, A. E.-D., and Ney, H. (2014). The rwth large vocabulary arabic handwriting recognition system. In *IAPR International Workshop on Document Analysis Systems*, pages 111–115.

[Hamdani et al., 2013] Hamdani, M., Mousa, A. E.-D., and Ney, H. (2013). Open vocabulary arabic handwriting recognition using morphological decomposition. In *ICDAR*, pages 280–284. IEEE.

[Hartley and Rao, 1968] Hartley, H. O. and Rao, J. N. (1968). Classification and estimation in analysis of variance problems. *Revue de l'Institut International de Statistique*, pages 141–147.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7) :1527–1554.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02) :107–116.

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

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8) :2554–2558.

[Hull, 1998] Hull, J. J. (1998). Document image skew detection : Survey and annotated bibliography. In *Document Analysis Systems II*, pages 40–64. World Scientific.

[Jaeger, 2001] Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany : German National Research Center for Information Technology GMD Technical Report*, 148 :34.

[Jaeger, 2002] Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach.* GMD-Forschungszentrum Informationstechnik.

[Johansson, 1986] Johansson, S. (1986). The tagged {LOB} corpus : User\'s manual.

[Johansson et al., 1978] Johansson, S., Leech, G. N., and Goodluck, H. (1978). *Manual of information to accompany the Lancaster-Oslo/Bergen Corpus of British English, for use with digital computer.* Department of English, University of Oslo.

[Kaltenmeier et al., 1993] Kaltenmeier, A., Caesar, T., Gloger, J. M., and Mandler, E. (1993). Sophisticated topology of hidden markov models for cursive script recognition. In *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pages 139–142. IEEE.

[Kim and Govindaraju, 1997] Kim, G. and Govindaraju, V. (1997). A lexicon driven approach to handwritten word recognition for real-time applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4) :366–379.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

[Knerr et al., 1998] Knerr, S., Augustin, E., Baret, O., and Price, D. (1998). Hidden markov model based word recognition and its application to legal amount reading on french checks. *Computer Vision and Image Understanding*, 70(3) :404–419.

[Koerich et al., 2003] Koerich, A. L., Sabourin, R., and Suen, C. Y. (2003). Large vocabulary off-line handwriting recognition : A survey. *Pattern Analysis & Applications*, 6(2) :97–121.

[Kozielski et al., 2013a] Kozielski, M., Doetsch, P., and Ney, H. (2013a). Improvements in rwth's system for off-line handwriting recognition. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 935–939. IEEE.

[Kozielski et al., 2013b] Kozielski, M., Rybach, D., Hahn, S., Schlüter, R., and Ney, H. (2013b). Open vocabulary handwriting recognition using combined word-level and character-level language models. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8257–8261. IEEE.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Kumar et al., 2007] Kumar, S., Gupta, R., Khanna, N., Chaudhury, S., and Joshi, S. D. (2007). Text extraction and document image segmentation using matched wavelets and mrf model. *Image Processing, IEEE Transactions on*, 16(8) :2117–2128.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields : Probabilistic models for segmenting and labeling sequence data.

[Le Cun et al., 1997] Le Cun, Y., Bottou, L., and Bengio, Y. (1997). Reading checks with multilayer graph transformer networks. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 1, pages 151–154. IEEE.

[LeCun, 1985] LeCun, Y. (1985). Une procedure d'apprentissage ponr reseau a seuil asymetrique. *proceedings of Cognitiva 85*, pages 599–604.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553) :436–444.

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a backpropagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

[Likforman-Sulem et al., 2007] Likforman-Sulem, L., Zahour, A., and Taconet, B. (2007). Text line segmentation of historical documents : a survey. *International journal on document analysis and recognition*, 9(2) :123–138.

[Louloudis et al., 2009] Louloudis, G., Gatos, B., Pratikakis, I., and Halatsis, C. (2009). Text line and word segmentation of handwritten documents. *Pattern Recognition*, 42(12) :3169–3183.

[Lu and Shridhar, 1996] Lu, Y. and Shridhar, M. (1996). Character segmentation in handwritten words—an overview. *Pattern recognition*, 29(1) :77–96.

[Madhvanath and Govindaraju, 1996] Madhvanath, S. and Govindaraju, V. (1996). Holistic lexicon reduction for handwritten word recognition. In *Document Recognition III*, pages 224–234.

[Madhvanath et al., 2001] Madhvanath, S., Krpasundar, V., and Govindaraju, V. (2001). Syntactic methodology of pruning large lexicons in cursive script recognition. *Pattern Recognition*, 34(1) :37–46.

[Marti and Bunke, 2001] Marti, U.-V. and Bunke, H. (2001). Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system. *International journal of Pattern Recognition and Artificial intelligence*, 15(01) :65–90.

[Marti and Bunke, 2002] Marti, U.-V. and Bunke, H. (2002). The iam-database : an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1) :39–46.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133.

[Menasri et al., 2012] Menasri, F., Louradour, J., Bianne-Bernard, A.-L., and Kermorvant, C. (2012). The a2ia french handwriting recognition system at the rimes-icdar2011 competition. In *Document Recognition and Retrieval XIX*, pages 82970Y–82970Y.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv :1301.3781*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Mioulet et al., 2015] Mioulet, L., Bideault, G., Chatelain, C., Paquet, T., and Brunessaux, S. (2015). Exploring multiple feature combination strategies with a recurrent neural network architecture for off-line handwriting recognition. In *Document Recognition and Retrieval*, pages 94020F–94020F.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv :1312.5602*.

[Mohri, 1996] Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(1) :61–80.

[Morillot et al., 2013] Morillot, O., Likforman-Sulem, L., and Grosicki, E. (2013). New baseline correction algorithm for text-line recognition with bidirectional recurrent neural networks. *Journal of Electronic Imaging*, 22(2) :023028–023028.

[Morita et al., 2001] Morita, M., El Yacoubi, A., Sabourin, R., Bortolozzi, F., and Suen, C. Y. (2001). Handwritten month word recognition on brazilian bank cheques. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 972–976. IEEE.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Ng, 2004] Ng, A. Y. (2004). Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM.

[Nigam et al., 1998] Nigam, K., McCallum, A., Thrun, S., Mitchell, T., et al. (1998). Learning to classify text from labeled and unlabeled documents. *AAAI/IAAI*, 792.

[O'Gorman, 1993] O'Gorman, L. (1993). The document spectrum for page layout analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(11) :1162–1173.

[Otsu, 1979] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1) :62–66.

[Palumbo et al., 1992] Palumbo, P. W., Srihari, S. N., Soh, J., Sridhar, R., and Demjanenko, V. (1992). Postal address block location in real time. *Computer*, 25(7) :34–42.

[Pan and Yang, 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10) :1345–1359.

[Papandreou and Gatos, 2012] Papandreou, A. and Gatos, B. (2012). Word slant estimation using non-horizontal character parts and core-region information. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 307–311. IEEE.

[Paquet and Lecourtier, 1993] Paquet, T. and Lecourtier, Y. (1993). Recognition of handwritten sentences using a restricted lexicon. *Pattern Recognition*, 26(3) :391–407.

[Parisse, 1996] Parisse, C. (1996). Global word shape processing in off-line recognition of handwriting. *IEEE transactions on pattern analysis and machine intelligence*, 18(4) :460–464.

[Pastor et al., 2004] Pastor, M., Toselli, A., and Vidal, E. (2004). Projection profile based algorithm for slant removal. *Image Analysis and Recognition*, pages 183–190.

[Pesch et al., 2012] Pesch, H., Hamdani, M., Forster, J., and Ney, H. (2012). Analysis of preprocessing techniques for latin handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 280–284. IEEE.

[Pham et al., 2014] Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *International Conference on Frontiers in Handwriting Recognition*, pages 285–290.

[Plamondon and Srihari, 2000] Plamondon, R. and Srihari, S. N. (2000). Online and offline handwriting recognition : a comprehensive survey. *IEEE PAMI*, 22(1) :63–84.

[Plötz and Fink, 2009] Plötz, T. and Fink, G. A. (2009). Markov models for offline handwriting recognition : a survey. *International Journal on Document Analysis and Recognition*, 12(4) :269–298.

[Povey et al., 2011] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., et al. (2011). The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society.

[Poznanski and Wolf, 2016] Poznanski, A. and Wolf, L. (2016). Cnn-n-gram for handwriting word recognition. In *CVPR*, pages 2305–2314.

[Puigcerver, 2017] Puigcerver, J. (2017). Are multidimensional recurrent layers really necessary for handwritten text recognition ? pages 67–72.

[Puskorius and Feldkamp, 1994] Puskorius, G. V. and Feldkamp, L. A. (1994). Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *Neural Networks, IEEE Transactions on*, 5(2) :279–297.

[Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1) :145–151.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1) :81–106.

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

[Richarz et al., 2014] Richarz, J., Vajda, S., Grzeszick, R., and Fink, G. A. (2014). Semi-supervised learning for character recognition in historical archive documents. *Pattern Recognition*, 47(3) :1011–1020.

[Romero et al., 2015] Romero, V., Sanchez, J. A., Bosch, V., Depuydt, K., and de Does, J. (2015). Influence of text line segmentation in handwritten text recognition. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 536–540. IEEE.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net : Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386.

[Rosenfeld, 2000] Rosenfeld, R. (2000). Two decades of statistical language modeling : Where do we go from here ? *Proceedings of the IEEE*, 88(8) :1270–1278.

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb : An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, DTIC Document.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). Artificial intelligence : a modern approach.

[Sánchez et al., 2016] Sánchez, J. A., Romero, V., Toselli, A. H., and Vidal, E. (2016). Icfhr2016 competition on handwritten text recognition on the read dataset. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 630–635. IEEE.

[Sanger, 1989] Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, 2(6) :459–473.

[Sauvola and Pietikäinen, 2000] Sauvola, J. and Pietikäinen, M. (2000). Adaptive document image binarization. *Pattern recognition*, 33(2) :225–236.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11) :2673–2681.

[Scudder, 1965] Scudder, H. (1965). Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11(3) :363–371.

[Seni and Cohen, 1994] Seni, G. and Cohen, E. (1994). External word segmentation of off-line handwritten text lines. *Pattern Recognition*, 27(1) :41–52.

[Senior and Robinson, 1996] Senior, A. and Robinson, T. (1996). Forward-backward retraining of recurrent neural networks. *NIPS*, pages 743–749.

[Senior and Robinson, 1998] Senior, A. W. and Robinson, A. J. (1998). An off-line cursive handwriting recognition system. *IEEE transactions on pattern analysis and machine intelligence*, 20(3) :309–321.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System technical journal*, pages 379–423.

[Shridhar et al., 1997] Shridhar, M., Houle, G., and Kimura, F. (1997). Handwritten word recognition using lexicon free and lexicon directed word recognition algorithms. In *ICDAR*, volume 2, pages 861–865.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676) :354–359.

[Srihari, 1993] Srihari, S. N. (1993). Recognition of handwritten and machine-printed text for postal address interpretation. *Pattern recognition letters*, 14(4) :291–302.

[Srihari and Kuebert, 1997] Srihari, S. N. and Kuebert, E. J. (1997). Integration of hand-written address interpretation technology into the united states postal service remote computer reader system. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 2, pages 892–896. IEEE.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1) :1929–1958.

[Steinherz et al., 1999] Steinherz, T., Rivlin, E., and Intrator, N. (1999). Offline cursive script word recognition–a survey. *International Journal on Document Analysis and Recognition*, 2(2) :90–110.

[Stolcke et al., 2002] Stolcke, A. et al. (2002). Srilm-an extensible language modeling toolkit. In *Interspeech*, volume 2002, page 2002.

[Stuner et al., 2016a] Stuner, B., Chatelain, C., and Paquet, T. (2016a). Cascade de réseaux blstm vérifiée par le lexique pour la reconnaissance d'écriture. In *Reconnaissance de Formes et Intelligence Artificielle.*

[Stuner et al., 2016b] Stuner, B., Chatelain, C., and Paquet, T. (2016b). Cascading blstm networks for handwritten word recognition. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 3416–3421. IEEE.

[Stuner et al., 2016c] Stuner, B., Chatelain, C., and Paquet, T. (2016c). Cohort of lstm and lexicon verification for handwriting recognition with gigantic lexicon. *arXiv preprint arXiv :1612.07528.*

[Stuner et al., 2016d] Stuner, B., Chatelain, C., and Paquet, T. (2016d). A lexicon verification strategy in a blstm cascade framework. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 234–239. IEEE.

[Stuner et al., 2017a] Stuner, B., Chatelain, C., and Paquet, T. (2017a). Lv-rover : Lexicon verified recognizer output voting error reduction. *arXiv preprint arXiv :1707.07432.*

[Stuner et al., 2017b] Stuner, B., Chatelain, C., and Paquet, T. (2017b). Self-training of blstm with lexicon verification for handwriting recognition. In *Document Analysis and Recognition (ICDAR), 2017 14th International Conference on*. IEEE.

[Sun et al., 2017] Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. *arXiv preprint arXiv :1707.02968.*

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning : An introduction*, volume 1. MIT press Cambridge.

[Swaileh et al., 2016] Swaileh, W., Lerouge, J., and Paquet, T. (2016). A unified french/english syllabic model for handwriting recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 536–541. IEEE.

[Tay et al., 2001] Tay, Y. H., Lallican, P.-M., Khalid, M., Knerr, S., and Viard-Gaudin, C. (2001). An analytical handwritten word recognition system with word-level discriminant training. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 726–730. IEEE.

[Thireou and Reczko, 2007] Thireou, T. and Reczko, M. (2007). Bidirectional long short-term memory networks for predicting the subcellular localization of eukaryotic proteins. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(3) :441–446.

[Thomas et al., 2010] Thomas, S., Chatelain, C., Heutte, L., and Paquet, T. (2010). An information extraction model for unconstrained handwritten documents. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3412–3415. IEEE.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude. *COURSERA : Neural networks for machine learning*, 4(2) :26–31.

[Vapnik, 1998] Vapnik, V. (1998). *Statistical learning theory. 1998*. Wiley, New York.

[Vapnik, 1995] Vapnik, V. N. (1995). The nature of statistical learning theory.

[Viana and Oliveira, 2017] Viana, M. P. and Oliveira, D. A. B. (2017). Fast cnn-based document layout analysis. In *Computer Vision Workshop (ICCVW), 2017 IEEE International Conference on*, pages 1173–1180. IEEE.

[Viard-Gaudin and Barba, 1991] Viard-Gaudin, C. and Barba, D. (1991). A multi-resolution approach to extract the address block on flat mail pieces. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 2701–2704. IEEE.

[Vinciarelli and Luettin, 2001] Vinciarelli, A. and Luettin, J. (2001). A new normalization technique for cursive handwritten words. *Pattern recognition letters*, 22(9) :1043–1050.

[Vinyals et al., 2015] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell : A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *CVPR*, volume 1, pages I–511. IEEE.

[Viterbi, 1967] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory*, 13(2) :260–269.

[Voigtlaender et al., 2016] Voigtlaender, P., Doetsch, P., and Ney, H. (2016). Handwriting recognition with large multidimensional long short-term memory recurrent neural networks. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 228–233. IEEE.

[Wang and Jiang, 1994] Wang, L. and Jiang, T. (1994). On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4) :337–348.

[Welch, 2003] Welch, L. R. (2003). Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4) :10–13.

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

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2) :270–280.

[Wolf et al., 2002] Wolf, C., Jolion, J.-M., and Chassaing, F. (2002). Text localization, enhancement and binarization in multimedia documents. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 1037–1040. IEEE.

[Wolf et al., 1997] Wolf, M., Niemann, H., and Schmidt, W. (1997). Fast address block location on handwritten and machine printed mail-piece images. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 2, pages 753–757. IEEE.

[Wöllmer et al., 2010] Wöllmer, M., Metallinou, A., Eyben, F., Schuller, B., and Narayanan, S. S. (2010). Context-sensitive multimodal emotion recognition from speech and facial expression using bidirectional lstm modeling. In *INTERSPEECH*, pages 2362–2365.

[Wong et al., 1982] Wong, K. Y., Casey, R. G., and Wahl, F. M. (1982). Document analysis system. *IBM journal of research and development*, 26(6) :647–656.

[Wu et al., 2017] Wu, Q., Shen, C., Wang, P., Dick, A., and van den Hengel, A. (2017). Image captioning and visual question answering based on attributes and external knowledge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell : Neural image caption generation with visual attention. *arXiv preprint arXiv :1502.03044*.

[Yarowsky, 1995] Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics.

[Yeh et al., 1987] Yeh, P.-S., Antoy, S., Litcher, A., and Rosenfeld, A. (1987). Address location on envelopes. *Pattern Recognition*, 20(2) :213–227.

[Young et al., 2002] Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., et al. (2002). The htk book. *Cambridge university engineering department*, 3 :175.

[Zamora-Martinez et al., 2014] Zamora-Martinez, F., Frinken, V., España-Boquera, S., Castro-Bleda, M. J., Fischer, A., and Bunke, H. (2014). Neural network language models for off-line handwriting recognition. *Pattern Recognition*, 47(4) :1642–1652.

[Zaremba et al., 2014] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv :1409.2329*.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta : an adaptive learning rate method. *arXiv preprint arXiv :1212.5701*.

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

[Zhang, 2013] Zhang, B. (2013). Reliable classification of vehicle types based on cascade classifier ensembles. *Intelligent Transportation Systems*, 14(1) :322–332.

[Zhang et al., 2007] Zhang, P., Bui, T. D., and Suen, C. Y. (2007). A novel cascade ensemble classifier system with a high recognition performance on handwritten digits. *Pattern Recognition*, 40(12) :3415–3429.

[Zhang et al., 2016] Zhang, R., Isola, P., and Efros, A. A. (2016). Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer.

[Zhu, 2005] Zhu, X. (2005). Semi-supervised learning literature survey.