Université de Montréal

Adaptation de modèles de traduction dans le cadre du projet TransType

par
Laurent Nepveu

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de M.Sc.
en informatique

Avril 2004

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé :
Adaptation de modèles de traduction
dans le cadre du projet TransType

présenté par :
Laurent Nepveu

a été évalué par un jury composé des personnes suivantes :

Miklós Csürös
président-rapporteur

Guy Lapalme
directeur de recherche

Philippe Langlais
codirecteur

Yoshua Bengio
membre du jury

Résumé

TransType est un logiciel de traduction assistée par ordinateur. L'usager entre le texte source qu'il désire traduire. Le programme propose ensuite au traducteur en cours de traduction les mots et les phrases qu'il juge être les plus probables d'être immédiatement frappés. Le système de traduction présentement contenu dans TransType est statique, c'est-à-dire, qu'une fois entraîné, il fera toujours les mêmes prédictions dans les mêmes contextes. Par sa nature interactive, TransType aurait avantage à apprendre de l'utilisateur pour améliorer ses prédictions. Nous avons donc incorporé aux systèmes de traduction de TransType des modèles de langue cache qui s'inspirent des travaux de Kuhn et de Mori [13]. Nous avons aussi étendu l'idée des modèles de langue cache au cas bilingue en développant un modèle de traduction cache. Nous présentons les résultats tant théoriques (perplexité des modèles) que pratiques (simulation d'un utilisateur) de ces modèles adaptatifs. Les modèles de langue cache entraînent des baisses de perplexité significatives. Les performances des modèles de traduction cache sont plus modestes, mais ouvre la voie à plus de recherche dans le domaine.

Mots clés : traduction assistée par ordinateur, traduction statistique, modèles adaptatifs, modèles cache, traitement des langues naturelles, linguistique informatique, intelligence artificielle.

Abstract

TransType is a program for computer-assisted translation. The user enters the source text he wishes to translate. The program then proposes to the translator, during his translation, words and sentences that it judges to be the most probable to be immediately typed. The translation system contained in TransType is static. In other words, once trained, it will always make the same predictions in the same context. By its interactive nature, TransType would take advantage of learning from its user, to improve its predictions. We thus incorporated in the translation systems of TransType cache-based language models inspired by the work of Kuhn and de Mori [13]. We also have extended the idea of the cache-based language model to the bilingual case by developing a cache-based translation model. We present theoretical (perplexity of the models) and pratical (simulation of a user) results of these adaptive models. Cache-based language model prooved to lead to very interesting perplexity drops. As for the cache-based translation model, the results obtained were more modest, but showed great research promise.

Keywords : computer-assisted translation, statistical translation, adaptive models, cache-based models, natural language processing, computer linguistics, artificial intelligence.

Table des matières

1 Introduction **1**

2 TransType **6**

3 L'approche statistique à la traduction automatique **10**

3.1 Le canal bruité 10

3.2 Les modèles de langue 13

3.3 Les modèles de traduction 17

3.3.1 L'alignement de textes 18

3.3.2 Les modèles IBM 20

3.3.3 Modèles Entropie Maximale Divergence Minimale 26

3.3.4 Les modèles de TransType 28

4 Adaptation de modèles de langue **33**

4.1 Les modèles de langue cache 36

4.2 Résultats des modèles de langue cache dans un contexte unilingue . . . 41

4.3 Résultats des modèles de langue cache dans un contexte de traduction . 47

4.3.1 IBM2 et modèle de langue cache 48

4.3.2 MDI2B avec modèle de langue cache dans la distribution de référence 50

4.4 Discussion 53

5 Adaptation de modèles de traduction **56**

5.1 Le modèle MDI2BCache 56

5.1.1 Poids des fonctions de trait cache 59

5.1.2 Alignement de Viterbi pour l'ajout de paires dans la cache . . . 60

5.2 Résultats du modèle MDI2BCache 61

5.3 Discussion 64

5.3.1 Améliorations possibles au modèle MDI2BCache 68

6 Évaluation des modèles dans le cadre de TransType **70**

6.1 Résultats du modèle MDI2B avec modèle de langue cache dans la distribution de référence 71

6.2 Résultats du modèle MDI2BCache 73

6.3 Discussion 74

7 Conclusion **77**

A Les corpora utilisés **83**

A.1 Le Hansard 83

A.2 *Sniper* 84

B Soutien mathématique **85**

B.1 La perplexité 85

Table des figures

2.1 Capture d'écran d'un prototype de TransType 7

3.1 Problème de sous-représentation des données dans un modèle bigramme 15
3.2 Exemple d'un alignement entre deux phrases 21
3.3 Exemple d'alignement de Viterbi produit par le modèle IBM1. 25
3.4 Exemple d'alignement de Viterbi produit par le modèle IBM2. 25
3.5 Exemple de prédicition d'un modèle MDI de TransType 30

4.1 Schéma de l'adaptation de modèles de langue 34
4.2 Exemple motivant l'utilisation de modèles de langue cache 36
4.3 Problème de propagation d'erreurs dans les modèles de langue cache. 40

5.1 Échantillon du contenu de la cache pour différentes configurations du modèle MDI2BCache 65

Liste des tableaux

4.1 Perplexités des modèles cache de Martin, Liermann et Ney 39
4.2 Perplexités des modèles cache de Clarkson et Robinson 40
4.3 Perplexités des modèles de langue avec composante cache sur la tranche test du corpus Hansard 44
4.4 Perplexité des modèles de langue avec composante cache sur la tranche test du corpus *sniper* 45
4.5 Perplexités du modèle de traduction IBM2 interpolé avec un modèle de langue avec composante cache (Hansard) 49
4.6 Perplexités du modèle de traduction IBM2 interpolé avec un modèle de langue avec composante cache (*sniper*) 49
4.7 Perplexités du modèle MDI2B interpolé avec une composante cache sur la tranche test du corpus Hansard. 51
4.8 Perplexités du modèle MDI2B interpolé avec une composante cache sur le corpus *sniper*. 52
4.9 Perplexités du modèle MDI2B avec composante cache intégrée au modèle trigramme de référence avec réentraînement du modèle MDI2B complet 53
4.10 Perplexités du modèle MDI2B avec composante cache intégrée au modèle trigramme de référence avec réentraînement du modèle MDI2B complet 53

5.1 Perplexités des modèles de traduction cache MDI2BCache (version avec un seul poids cache, Hansard) 62
5.2 Perplexités des modèles de traduction cache MDI2BCache (version avec un poids cache par paire, Hansard) 62
5.3 Perplexités des modèles de traduction cache MDI2BCache (version avec un poids cache par paire et alignement de Viterbi, Hansard) 63
5.4 Perplexités des modèles de traduction cache MDI2BCache (version avec un poids cache par paire et alignement de Viterbi, *sniper*) 64

6.1 Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de langue (Hansard) 71
6.2 Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de langue (*sniper*) 72
6.3 Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de traduction (Hansard) 72
6.4 Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de traduction (*sniper*) 73

A.1 Les différentes tranches du corpus Hansard utilisées. 84
A.2 La tranche du corpus *sniper* utilisée. 84

À mes parents.

Remerciements

J'aimerais remercier mon directeur de recherche Guy Lapalme pour m'avoir judicieusement guidé tout au long de ce projet et mon codirecteur Philippe Langlais pour son temps et ses conseils.

Je remercie également mes parents, Julie et Jacques, pour m'avoir permis de devenir la personne que je suis aujourd'hui et ma copine Geneviève pour sa présence et sa gentillesse de tous les instants.

Je remercie tout spécialement George Foster d'avoir partagé avec moi son savoir et ses travaux avec grande disponibilité et grande générosité.

Finalement, je remercie le DIRO, la FES et le CRSNG pour leur soutien financier tout au long de mes études graduées.

Chapitre 1

Introduction

La traduction d'un texte d'une langue vers une autre est une des tâches pour laquelle l'ordinateur n'a pas encore surpassé les capacités humaines. En fait, ses performances sont très pauvres comparativement à celles de l'humain.

Avec l'avènement de l'Internet et des plus récentes technologies de communications, l'accès à l'information croît présentement de façon exponentielle. Bien que la très grande majorité de l'information disponible soit en anglais, il existe tout de même des quantités énormes d'information dans les autres langues mondiales, majoritairement en chinois, espagnol, français et arabe. Cette quantité d'information est donc disponible pour tous, mais non accessible pour la majorité étant donné la barrière de la langue.

Cette information ne peut pas être systématiquement traduite dans plusieurs langues étant donné le coût associé à la traduction humaine. Il serait donc très intéressant de pouvoir exploiter la rapidité de calcul de l'ordinateur en traduction de la même façon que c'est fait, par exemple, en recherche d'information où on entre une requête de quelques mots et des milliers de documents pertinents sont retournés et où cette même tâche serait extrêmement fastidieuse si elle devait être faite de façon humaine.

Le problème de la traduction automatique se situe au niveau des résultats qui ne sont pas du tout à la hauteur de la traduction humaine. Les textes produits par l'état

de l'art de la traduction automatique ne permettent que de connaître le sujet du texte et de déceler quelques phrases clés, sans plus.

Le projet TransType a donc tenté de s'attaquer à la traduction automatique en utilisant les points forts des ordinateurs actuels et en laissant de coté ses points faibles. Les points faibles de l'ordinateur en traduction automatique se situent au niveau de la grammaticalité de la phrase traduite et de l'ambiguïté probable dans la phrase source. L'ordinateur est par contre capable, étant donné une phrase source, d'identifier avec une bonne précision un certain nombre de mots se trouvant potentiellement dans la phrase cible correspondante.

L'application TransType effectue de la traduction *assistée* par ordinateur. Elle comprend un moteur de traduction et un traitement de texte. L'usager soumet son texte source au logiciel et débute la frappe de la traduction de ce texte. Au fur et à mesure que le traducteur frappe sa traduction, TransType propose un ou plusieurs mots qu'il juge comme ayant une forte probabilité de survenir à ce point du texte.

TransType utilise un moteur de traduction pour réaliser ces propositions. Ce moteur de traduction contient deux composantes : un modèle de traduction et un modèle de langue. Un modèle de langue vise à estimer la probabilité d'apparition de toute chaîne de mots d'une langue, alors que le modèle de traduction vise à établir la probabilité qu'a une chaîne de mots d'une langue T d'être la traduction d'une autre chaîne de mots d'une langue S. Les modèles de traduction et de langue, dans leurs versions les plus performantes à ce jour, n'arrivent pas à produire un texte grammaticalement correct représentant exactement le texte source. Par contre, ils arrivent avec une bonne précision à identifier, étant donné une phrase source, les mots qui ont une probabilité élevée de se retrouver dans la phrase cible et à ordonner ces mots de façon logique. TransType utilise ces deux modèles à travers son moteur de traduction pour faire des propositions sur les mots prochainement frappés par l'utilisateur. Celui-ci peut alors accepter ou refuser ces

propositions. Ceci permet au traducteur d'améliorer sa productivité en économisant un très grand nombre de frappes au clavier étant donné qu'il accepte le plus grand nombre possible des propositions faites par le programme.

Les modèles de traduction et de langue sont des distributions de probabilités sur les différents événements que ce sont les mots de la langue cible. En tout point de la traduction, on demande aux modèles le ou les mots les plus probables d'être immédiatement frappés par le traducteur. Ces distributions de probabilités sont estimées à partir de corpora, unilingues pour le modèle de langue et bilingues pour le modèle de traduction. Pour que ces distributions soient représentatives de la langue, des millions de mots et de phrases sont nécessaires. L'application TransType est, quant à elle, entraînée à partir du corpus Hansard (Annexe A.1), le journal des débats parlementaires canadiens.

La nécessité d'utiliser de très grands corpora fait en sorte que l'entraînement de tels modèles (l'estimation des distributions de probabilités) est une tâche très coûteuse en temps. Dans la version courante de TransType, les modèles sont entraînés une fois, préalablement à l'utilisation. Ceci implique que les textes traduits par l'utilisateur de TransType n'ont aucune influence sur les modèles.

Or, plusieurs facteurs nous laissent croire qu'il serait très intéressant que de tels modèles, une fois entraînés, soient dynamiques et non pas statiques, c'est-à-dire qu'ils se modifient au fur et à mesure de leur utilisation. Premièrement, un même traducteur peut traduire plusieurs documents similaires, l'information d'un document préalablement traduit est utile à la traduction du suivant. Ensuite, pour un texte donné, l'information contenue dans la partie déjà traduite peut être indicatrice de ce qui est à venir dans le texte. Enfin, chaque traducteur a son propre "style" de traduction qu'il serait certainement utile de modéliser. Il serait donc très utile d'utiliser les documents déjà traduits, ainsi que la partie du document déjà traduite pour préciser les prédictions futures.

Ce phénomène qui consiste à modifier un modèle en fonction des textes déjà traduits

par ce même modèle est appelé *adaptation*. Le modèle s'adapte donc au contenu des textes qu'il voit passer lors de traduction automatique ou d'assistance à la traduction.

Le présent mémoire vise donc l'étude du type d'adaptation qui consiste à utiliser la partie du document déjà traduite dans le document en cours de traduction. Cette forme d'adaptation est implémentée avec des modèles de type cache. Ces modèles sont inspirés des mémoires cache utilisées dans les ordinateurs contemporains. Ces mémoires cache sont motivées par le principe de la localité qui dit que plusieurs accès consécutifs à la mémoire ont tendance à utiliser les mêmes blocs de la mémoire principale. Ainsi, on conserve dans une cache dont l'accès est beaucoup plus rapide les derniers blocs démandés par l'utilisateur puisqu'ils sont plus susceptibles d'être redemandés. De la même façon, lorsqu'il écrit ou il parle, l'humain a tendance à utiliser les mêmes mots et les mêmes tournures de phrases et ce de façon répétitive. On voudrait donc, dans nos modèles cache, conserver en mémoire les quelques dernières phrases entrées par l'utilisateur en vue de modifier les distributions de probabilités en fonction du contenu de ces phrases. Ces modèles cache tentent donc de modéliser le sous-ensemble d'une langue qui est utilisé par le traducteur.

Nos travaux se sont appuyés sur ceux de George Foster qu'il présente dans sa thèse de doctorat : *Text Prediction For Translators* [8]. Nos modèles mathématiques sont des extensions des modèles qui y sont développés . De plus, toutes les implémentations de ces modèles ont été réalisées à partir des *glm* et *gtm toolkits* de George Foster.

Le présent mémoire contient 7 chapitres et deux annexes. Le chapitre 2 présente l'application TransType, alors que le chapitre 3 présente l'approche statistique à la traduction par ordinateur. Cette approche est celle du canal bruité dans laquelle on modélise séparemment le processus de génération de la langue cible et le processus de traduction de la langue source vers la langue cible dans un modèle de langue (section 3.2) et un modèle de traduction (section 3.3) respectivement.

Le chapitre 4 présente les modèles de langue adaptatifs, ainsi que les résultats théoriques de nos implémentations de ces modèles dans le contexte unilingue et dans le contexte de la traduction. Le chapitre 5 présente les modèles de traduction adaptatifs, en particulier le modèle MDI2BCache que nous avons développé. On y présente aussi les résultats théoriques obtenus par ce modèle dans le contexte de la traduction, ainsi que ceux obtenus par un modèle combiné comprenant une composante cache tant au niveau du modèle de langue que du modèle de traduction.

Le chapitre 6 présente les résultats de l'évaluation de nos modèles dans le contexte bien précis de l'application TransType. À l'aide d'un programme simulant une utilisation de TransType par un traducteur, on y évalue les économies de frappes au clavier qu'entraînent nos modèles.

Finalement, on retrouve la conclusion de ce mémoire au chapitre 7. L'annexe A présente les corpora que nous avons utilisés tout au long des travaux de ce mémoire, alors que l'annexe B offre un soutien mathématique.

Chapitre 2

TransType

Le projet TransType [16, 10, 15] consiste en la réalisation d'un outil de traduction assistée par ordinateur. Cet outil est constitué d'un éditeur de texte et d'un moteur de traduction probabiliste. Le traducteur doit d'abord soumettre son texte source à TransType. Il débute ensuite, à même l'interface graphique de TransType, la frappe de sa traduction. Au cours de la traduction, le moteur propose des complétions des mots et des phrases qui sont en train d'être frappés. À l'aide d'une seule touche, la traducteur peut accepter les propositions. La figure 1 présente une capture d'écran d'un prototype de TransType.

Parmi les nombreuses tentatives en traduction assistée par ordinateur, la pré-édition et la post-édition de traduction se sont avérées être des échecs [9]. La pré-édition consiste à annoter un texte source dans le but d'améliorer la performance de l'ordinateur à traduire automatiquement, alors que la post-édition voit le traducteur corriger les erreurs faites par l'ordinateur lors d'une traduction automatique. Il s'est avéré que le travail d'annotation (pré-édition) ou de correction (post-édition) rendait la tâche totale de traduction plus longue et plus laborieuse qu'une simple traduction humaine.

Ces échecs ont amené les chercheurs à s'intéresser à une forme de traduction assistée par ordinateur où l'ordinateur et le traducteur travaillent de pair. Cette idée est à la

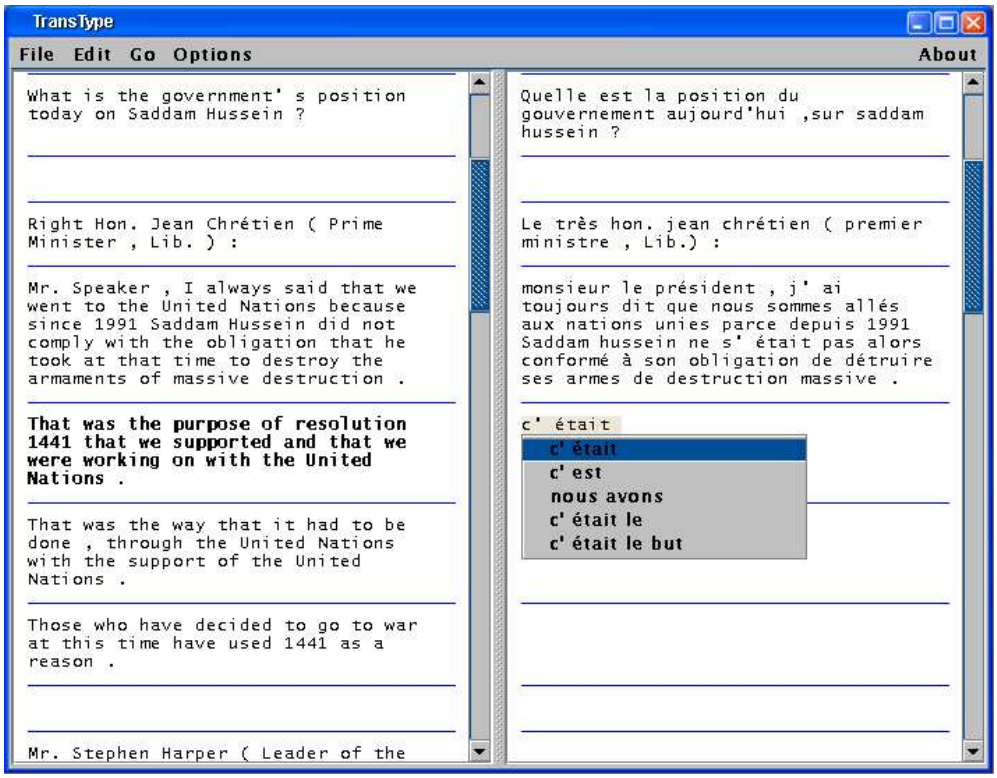

Fig. 2.1: Capture d'écran d'un prototype de TransType. Le texte source apparaît dans la partie gauche de l'écran, alors que le texte cible se situe dans la partie droite. La proposition courante apparaît dans le texte cible à l'endroit où se trouve le curseur.

base du projet TransType.

Ainsi, TransType propose à tout moment de la traduction une liste d'unités lexicales (mots ou phrases) que le moteur de traduction a déterminées comme étant les plus probables d'être prochainement tapées. La liste de propositions apparaît à l'écran de manière à ne pas déranger le travail du traducteur, mais en lui donnant la chance d'accepter ou non les propositions. Pour réaliser ses prédictions, le moteur de traduction utilise les mots de la phrase source et les quelques derniers mots cibles tapés par l'utilisateur.

TransType permet donc d'améliorer de deux façons la productivité des traducteurs. Premièrement, Foster et al. [9] montrent qu'une utilisation idéale de TransType entraîne

une économie de frappes au clavier de l'ordre de 75%. Ceci permet donc à l'utilisateur expérimenté d'accélérer sa traduction. De plus, TransType permet d'améliorer la qualité des traductions par l'ajout de connaissances qu'il amène au traducteur. Dans la plupart des cas, TransType proposera le mot désiré par le traducteur, lui permettant ainsi d'économiser du temps de frappe. Aussi, dans certains cas, le traducteur ne connait pas ou ne se rappelle plus de la traduction d'un certain mot ou d'un certain bout de phrase. Il arrive aussi que TransType propose un mot que le traducteur juge meilleur que le mot qu'il avait en tête. Dans de telles situations, TransType amène une connaissance ajoutée qui permet d'améliorer la qualité de la traduction.

Le moteur de traduction de TransType est constitué de deux modèles : un modèle de langue et un modèle de traduction. Le modèle de traduction s'occupe de la traduction des mots du texte source alors que le modèle de langue voit à ce que les prédictions générées une à la suite de l'autre forment des phrases grammaticalement correctes.

Les modèles de langue et modèles de traduction qu'utilise présentement TransType sont statiques, c'est-à-dire qu'ils sont d'abord entraînés[1] à l'aide de corpus parallèles[2] et ne se seront pas modifiés au cours de leur utilisation. Pour réaliser une bonne estimation des paramètres d'un modèle, des quantités énormes de données sont requises. Les modèles sont donc entraînés sur de grands corpora préalablement à leur utilisation. Dans la plupart des domaines liés aux documents qui seront traduits, de tels corpora ne sont pas disponibles. C'est pourquoi des corpora tels que des débats parlementaires, des textes juridiques ou encore des articles de journaux sont souvent employés pour entraîner ces modèles.

Un tel entraînement nous permet d'obtenir des modèles représentant le mieux possible la langue. Par contre, ces modèles sont piètrement adaptés à une tâche précise de

[1] L'entraînement d'un modèle de traduction ou de langue consiste en l'estimation des paramètres du modèle à partir de grandes quantités de données.

[2] Un corpus parallèle est constitué de deux groupes de textes en deux langues dont l'un est la traduction de l'autre.

traduction, puisqu'ils ne sont pas familiers avec le domaine des textes qu'ils aident à traduire. Ceci entraîne des performances moindres que désirées. Par exemple, un modèle entraîné avec des textes journalistiques aura des performances moindres lorsqu'utilisé pour prédire la traduction de textes scientifiques ou sportifs.

La répétition d'erreurs est un autre inconvénient associé au caractère statique des modèles. Par exemple, si le modèle fait une erreur sur la prédiction d'un mot dans un certain contexte, l'utilisateur doit corriger la sortie du modèle. Si ce même contexte se présente plus tard dans la traduction, cette même erreur sera à nouveau faite par le modèle et l'utilisateur devra la corriger une fois de plus. Cet inconvénient a des répercussions sur la performance de l'application, mais aussi sur la perception de l'utilisateur puisqu'il peut devenir irritant de corriger toujours la même erreur.

Le présent mémoire vise donc à étudier des moteurs de traduction adaptatifs qui pourront être intégrés à TransType. Les modèles adaptatifs se modifieront en fonction de la tâche de traduction courante, améliorant ainsi les prédictions au fur et à mesure de leur utilisation et palliant les inconvénients soulevés plus haut.

Chapitre 3

L'approche statistique à la traduction automatique

3.1 Le canal bruité

L'approche classique de la traduction automatique est celle symbolique. Cette approche consiste à générer un texte cible à partir d'un texte source en se basant sur des analyses lexicale, syntaxique et sémantique. L'approche statistique, quant à elle, estime une distribution de probabilité sur les événements que sont les mots du vocabulaire cible. Ces probabilités sont conditionnées sur l'historique dans le texte cible et sur le texte source. Pour générer un texte cible, on trouve la suite de mots ayant la plus grande probabilité étant donné le texte source selon la distribution estimée. L'approche statistique la plus commune se base sur l'analogie du canal bruité.

L'analogie du canal bruité est reprise du domaine de la reconnaissance de la parole. On considère un canal (bruité) dans lequel on entre (de façon parlée ou écrite) une phrase $\mathbf{t}$ d'une certaine langue T ayant $\mathcal{T}$ comme vocabulaire. Le canal est si bruité que la phrase $\mathbf{s}$ qui ressort est dans une langue S ayant $\mathcal{S}$ comme vocabulaire. $\mathbf{s}$ est bien sûr

traduction de $\mathbf{t}$.

La traduction automatique consiste donc, en présence d'une phrase $\mathbf{s}$ de la langue S, à trouver la phrase $\mathbf{t}$ de la langue T qui, en entrée au canal, a produit $\mathbf{s}$ en sortie. Le canal bruité, une boîte noire, représente donc le processus inverse de la traduction : la phrase cible est entrée dans le canal et la phrase source en ressort.

La modélisation statistique du processus de traduction cherche donc à créer un décodeur (équation 3.1) qui modélise le processus classique de traduction d'une langue source S vers une langue cible T. Étant donné une phrase source $\mathbf{s}$, on trouve $\hat{\mathbf{t}}$ à partir de $\mathbf{s}$ par le décodeur suivant :

$$\begin{aligned} \hat{\mathbf{t}} &= \underset{\mathbf{t}\in\mathcal{T}}{argmax}\ p(\mathbf{t}|\mathbf{s}) \\ &= \underset{\mathbf{t}\in\mathcal{T}}{argmax}\ \frac{p(\mathbf{s}|\mathbf{t})\times p(\mathbf{t})}{p(\mathbf{s})} \\ &= \underset{\mathbf{t}\in\mathcal{T}}{argmax}\ \underbrace{p(\mathbf{s}|\mathbf{t})}_{traduction} \times \underbrace{p(\mathbf{t})}_{langue\ cible} \end{aligned} \tag{3.1}$$

On utilise la loi de Bayes pour rendre la modélisation du processus plus simple. Si on tentait de modéliser la distribution $p(\mathbf{t}|\mathbf{s})$ directement, il faudrait alors voir à ce que les phrases $\mathbf{t}$ ayant une probabilité élevée contiennent des mots qui sont des traductions des mots de $\mathbf{s}$, mais aussi à ce que la phrase $\mathbf{t}$ soit grammaticalement correcte. Construire une telle distribution $p(\mathbf{t}|\mathbf{s})$ serait une tâche très complexe et requerrait une très grande quantité de données. Avec la loi de Bayes, on transforme le problème en une modélisation séparée des processus de traduction et de génération de la phrase cible de façon grammaticale. Ainsi, en estimant la distribution $p(\mathbf{s}|\mathbf{t})$, on n'a pas à se soucier que $\mathbf{t}$ ait ou non du sens au niveau grammatical, il faut simplement que les mots de $\mathbf{t}$ soient des traductions des mots de $\mathbf{s}$ et ce dans un ordre logique. Dans la distribution $p(\mathbf{t})$, on s'occupe de voir à ce que seules les phrases grammaticalement correctes aient une probabilité élevée.

Cette approche statistique vise à estimer une distribution de probabilité sur la sortie

du canal et ce, à partir d'une très grande quantité d'exemples d'entrée/sortie du canal. Ces exemples se présentent sous la forme de corpora bilingues.

La tâche de traduction est donc divisée en trois tâches distinctes. La première consiste à modéliser la langue cible, donc à estimer la distribution $p(\mathbf{t})$. La seconde consiste à modéliser le processus de traduction mot à mot, soit la distribution $p(\mathbf{s}|\mathbf{t})$. Finalement, la troisième tâche est de rechercher la phrase $\mathbf{t}$ qui maximise la probabilité $p(\mathbf{s}|\mathbf{t}) \times p(\mathbf{t})$. Nous nous attarderons dans le présent mémoire à rendre adaptatifs les modèles de traduction et de langue du moteur de traduction de TransType, c'est-à-dire de voir à ce que les modèles qui déterminent une probabilité sur la traduction mot à mot et sur le texte cible se modifient au fur et à mesure de l'utilisation. Les modèles feraient potentiellement de meilleures prédictions puisqu'ils estimeraient mieux le processus de traduction et la langue cible utilisée par le traducteur. La troisième tâche n'est pas abordée dans ce mémoire. Elle consiste, telle que mentionné plus haut, à rechercher, parmi toutes les prédictions possibles, la phrase cible qui est la plus probable d'être immédiatement frappée par l'utilisateur. Les tâches de détermination des distributions de probabilités et de recherche pouvant être considérées indépendamment, nous nous permettons ici de ne pas discuter de cette dernière et d'utiliser directement les techniques décrites dans [8].

Il est par contre à noter que le processus de recherche dans le cadre de l'application TransType est légèrement modifié puisqu'il consiste à ne rechercher que quelques mots et non la phrase complète. Ainsi, la recherche est plus rapide puisque seuls quelques mots sont recherchés à la fois. Cette rapidité est nécessaire étant donné la nature temps réel de l'application.

3.2 Les modèles de langue

Les modèles de langue permettent, comme le nom l'indique, de modéliser une langue en établissant une distribution probabiliste sur l'ensemble des chaînes d'unités lexicales de cette langue. Les modèles basés sur des mots attribueront donc à chaque chaîne de mots une probabilité de survenir dans une telle langue, alors que les modèles basés sur les caractères attribueront une probabilité de survenir à chaque chaîne de caractères d'une langue.

De manière générale, la distribution sur les mots doit respecter la contrainte probabiliste présentée à l'équation 3.2 où $\mathbf{s}$, pour un modèle basé sur les mots, est défini comme une suite de mots quelconque.

$$\sum_{\mathbf{s}} P(\mathbf{s}) = 1 \tag{3.2}$$

Ainsi, l'équation 3.3 définit la probabilité $P(\mathbf{s})$ sans perte d'information où $\mathbf{h}$ est appelé l'*historique* (on ajoute $n-1$ mots w en début de phrase pour assurer la normalisation de la distribution) [14].

$$P(\mathbf{s}) = \prod_{i=1}^{|s|+1} P(w_i | \underbrace{w_1 ... w_{i-1}}_{\mathbf{h}}) \tag{3.3}$$

Modéliser la probabilité d'un mot en la conditionnant sur la totalité des mots qui le précèdent n'est évidemment pas une approche viable. Il est possible, afin d'obtenir une bonne représentation des données avec les corpora disponibles et afin d'obtenir un temps d'entraînement raisonnable, de réduire la longueur de l'historique en faisant une approximation markovienne d'ordre $n-1$. Ainsi, la probabilité d'un mot est conditionnée sur les $n-1$ mots qui le précèdent. De tels modèles sont nommés ngrammes. $P(\mathbf{s})$ y est donc ainsi définie :

$$P(s) \approx \prod_{i=1}^{|s|} P(w_i|w_{i-n+1}^{i-1}) \tag{3.4}$$

Les ngrammes les plus connus et les plus utilisés sont les unigrammes, les bigrammes et les trigrammes. Ils conditionnent respectivement la probabilité d'un mot sur les zéro, un et deux mots qui le précèdent.

Il existe d'autres techniques pour modéliser une langue, notamment les grammaires hors contexte probabilistes, mais nous nous concentrons ici sur les modèles ngrammes. Ceux-ci se sont avérés les modèles les plus fiables et les plus performants et ce, de façon générale.

Les modèles développés dans ce mémoire sont entraînés avec un estimateur à vraisemblance maximale qui est, dans ce cas-ci, la fréquence relative de chaque évènement. La formule d'estimation des paramètres du modèle est donc la suivante dans le cas général du ngramme (avant lissage), on utilise l'opérateur $|x|$ pour représenter la fréquence dans le corpus d'entraînement :

$$\hat{P}(w_i|w_{i-n+1}^{i-1}) = \frac{|w_{i-n+1}^{i-1}w_i|}{\sum_w |w_{i-n+1}^{i-1}w|} = \frac{|w_{i-n+1}^{i-1}w_i|}{|w_{i-n+1}^{i-1}|} \tag{3.5}$$

et dans le cas particulier, plus simple, du bigramme :

$$\hat{P}(w_i|w_{i-1}) = \frac{|w_{i-1}w_i|}{\sum_w |w_{i-1}w|} = \frac{|w_{i-1}w_i|}{|w_{i-1}|} \tag{3.6}$$

Cette dernière formule est très intuitive. On établit la probabilité d'un mot étant donné le mot qui le précède comme étant le nombre de fois que ces deux mots surviennent un à la suite de l'autre ($|w_{i-1}w|$) divisé par le nombre de fois où le mot qui précède survient ($|w_{i-1}|$). Cet estimateur de probabilité est donc la fréquence relative avec laquelle le mot w_i suit le mot w_{i-1} dans le corpus d'entraînement.

Avec de tels modèles apparaît un problème de sous-représentation des données. La

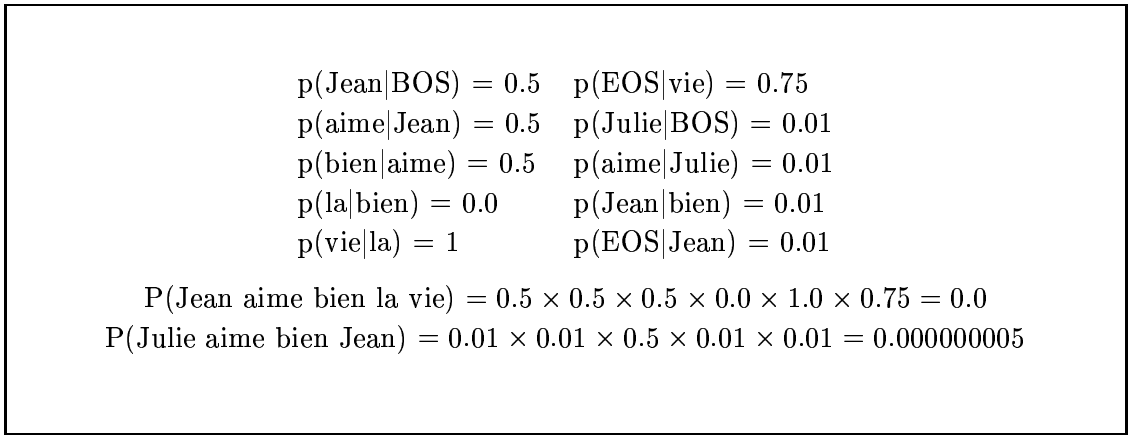

Fig. 3.1: Problème de sous-représentation des données dans un modèle bigramme. BOS et EOS sont des mots spéciaux ajoutés pour représenter un début de phrase et une fin de phrase.

probabilité d'une chaîne est modélisée comme le produit des probabilités conditionnées de chacun des mots la formant. Ainsi, une certaine chaîne peut avoir une probabilité nulle puisqu'un seul des mots a une probabilité nulle. La figure 3.1 illustre le problème relié à la sous-représentation des données. Dans cette exemple, la phrase *Jean aime bien la vie* semble beaucoup plus probable que la phrase *Julie aime bien Jean*, mais puisque les mots *bien la* n'ont jamais été vus un à la suite de l'autre dans le corpus d'entraînement, la phrase obtient une probabilité de 0. Par contre, la phrase *Julie aime bien Jean*, bien que la plupart de ses mots soient improbables dans le contexte donné, obtient une probabilité non nulle.

Pour pallier ce problème de sous-représentation des données, nous utilisons différentes techniques de lissage dont le but est d'attribuer des probabilités non nulles aux événements non vus dans le corpus d'entraînement. Idéalement, nous aimerions que les probabilités attribuées soient les plus représentatives de la probabilité réelle associée à ces événements.

Une des techniques de lissage les plus connues est celle attribuable à Jelinek et Mercer [12]. Elle consiste à faire une interpolation linéaire de plusieurs modèles markoviens d'ordres différents. Ainsi, tel qu'utilisé dans les travaux du présent mémoire, un

trigramme serait défini de la façon suivante :

$$\begin{aligned} p(w|\mathbf{h}) &= \lambda_1(w_{i-2}w_{i-1}) \times p(w_i|w_{i-2}w_{i-1}) \\ &+ \lambda_2(w_{i-2}w_{i-1}) \times p(w_i|w_{i-1}) \\ &+ \lambda_3(w_{i-2}w_{i-1}) \times p(w_i) \\ &+ \lambda_4(w_{i-2}w_{i-1}) \times \frac{1}{|V|+1} \end{aligned} \tag{3.7}$$

où $\lambda_1(w_{i-2}w_{i-1}) + \lambda_2(w_{i-2}w_{i-1}) + \lambda_3(w_{i-2}w_{i-1}) + \lambda_4(w_{i-2}w_{i-1}) = 1$ et $|V| + 1$ est le nombre d'événements de la distribution. Ainsi, on peut voir que si une certaine séquence de mots $w_1w_2w_3$ n'a pas été vue dans le corpus d'entraînement, sa probabilité sera déterminée par les modèles d'ordres inférieurs. Cette probabilité sera donc proportionnelle à la probabilité de la séquence w_2w_3.

On comprend intuitivement la motivation d'un tel modèle par l'exemple suivant. Si les chaînes *cette belle vie* et *cette extraordinaire vie* n'ont pas été vues dans le corpus d'entraînement, elles auront une probabilité nulle au niveau du trigramme. Leurs probabilités seront donc déterminées par les modèles d'ordres inférieurs. Ainsi, la probabilité $p(\text{vie}|\text{cette belle}) \approx p(\text{vie}|\text{belle})$ et $p(\text{vie}|\text{cette extraordinaire }) \approx p(\text{vie}|\text{extraordinaire})$. On peut donc s'attendre à ce que *cette belle vie* ait une probabilité plus grande que *cette extraordinaire vie*, ce qui est tout à fait logique dans la langue française, puisqu'on parle plus souvent de belle vie que d'extraordinaire vie. Un autre avantage d'une telle technique est que deux probabilités égales au niveau du trigramme seront départagées par les modèles d'ordres inférieurs.

Les modèles de langue ngrammes présentés ici sont statiques, c'est-à-dire que leurs paramètres sont estimés à partir d'un corpus préalablement à leur utilisation et qu'ils ne se modifient pas en fonction des textes qu'ils prédisent. Donc, par exemple, si la suite de mots *cette extraordinaire vie* n'a pas été vue dans le corpus d'entraînement lors de l'entraînement d'un modèle trigramme, bien que cette même suite se retrouve à plusieurs reprises dans un document que nous sommes en train de traiter avec notre modèle de langue, sa probabilité sera toujours zéro de la première à la dernière fois que

nous la rencontrons. Intuitivement, il est permis de penser que si nous rencontrons la même suite de trois mots neuf fois dans le document que nous analysons, sa probabilité devrait au moins être supérieure à zéro la dixième fois que nous la rencontrons. Un tel problème sera corrigé par l'adaptation des modèles de langue.

3.3 Les modèles de traduction

Alors que les modèles de langue estiment la probabilité d'apparition d'une certaine phrase dans une certaine langue, les modèles de traduction estiment la probabilité qu'une phrase dans une certaine langue soit traduction d'une autre phrase dans une autre langue. Les paramètres de ces modèles (famille IBM) sont les probabilités qu'un mot soit la traduction d'un autre pour toutes les paires de mots du produit vectoriel des vocabulaires source et cible et les probabilités qu'un mot à une certaine position dans la phrase source soit en relation de traduction avec un mot à une certaine position dans la phrase cible. On appelle ces paramètres les probabilités de transfert et les probabilités d'alignement.

Les modèles de traduction font partie intégrante de l'application TransType. Ils nous permettent d'obtenir, pour une certaine phrase source, une liste de mots pour lesquels la probabilité qu'ils soient en relation de traduction avec les mots de cette phrase source est grande.

TransType utilise présentement une variante des modèles de la famille IBM. Cette variante utilise la modélisation par EMDM (entropie maximale divergence minimale). Le présent chapitre vise donc à décrire les modèles IBM ainsi que la variante utilisée dans TransType.

3.3.1 L'alignement de textes

La ressource première utilisée pour créer des modèles de traduction est le bitexte. Un bitexte consiste en deux textes dont l'un est la traduction de l'autre et pour lesquels nous connaissons les liens de traduction qui lient les paragraphes et/ou les phrases. Plusieurs bitextes forment un corpus bilingue. Les corpora bilingues sont utilisés pour entraîner des modèles de traduction.

Avec la croissance exponentielle de l'Internet et de l'accès à l'information, la disponibilité de corpora bilingues est de plus en plus grande. Ce sont pour la plupart des textes officiels, tels des débats parlementaires et des documents juridiques de pays ayant plus d'une langue officielle tels le Canada, la Suisse et Hong-Kong.

La première opération faite sur un corpus bilingue pour que celui-ci soit exploitable est l'alignement de textes au niveau des paragraphes et/ou des phrases. Cette tâche consiste à établir quels paragraphes sources sont traductions de quels paragraphes cibles et, à même un paragraphe et sa traduction, quelles phrases sources sont traductions de quelles phrases cibles.

Le principal problème de l'alignement de texte est le fait qu'une phrase cible n'est pas systématiquement traduction d'une seule phrase source ou encore qu'une phrase source est parfois traduite par plusieurs phrases cibles. Si toutes les phrases étaient traduites une à une, c'est-à-dire qu'une phrase cible est la traduction d'une seule phrase source et que cette phrase source n'est traduite que par cette même phrase cible, la tâche serait triviale. On rapporte [17] que, pour un certain corpus analysé, 90% des phrases sont traduites une à une. C'est donc une petite portion des phrases qui, de manière générale, apporte toute la difficulté à l'exercice.

Church et Gale [4] introduisent un algorithme de programmation dynamique permettant l'alignement de textes. Ils ne permettent que les relations de traduction suivantes : 1 : 1, 1 : 2, 2 : 1, 2 : 2, 0 : 1, 1 : 0. Ainsi, une phrase source peut être traduite par zéro,

une ou deux phrases cibles, deux phrases sources ne peuvent être traduites que par une ou deux phrases cibles, ou encore une phrase cible peut être traduction d'aucune phrase source. Aussi, ils empêchent le croisement de traductions. Si on désigne le texte source comme S et le texte cible comme T, un croisement serait alors une phrase s_i traduite par une phrase t_k et une phrase s_j traduite par une phrase t_l où $i < j$ et $k > l$. Un algorithme permettant le croisement serait beaucoup plus coûteux en temps. Ils considèrent donc que les croisements sont trop peu fréquents pour y porter une attention particulière.

Pour trouver l'alignement optimal, on cherche à minimiser une fonction de coût $D(i,j)$ définie à l'équation 3.8, qui correspond au coût associé à l'alignement des i premières phrases du texte source avec les j premières phrases du texte cible.

$$D(i,j) = min \begin{cases} \text{D(i,j-1)} & + & \text{cost(0 :1 align } \emptyset, t_j) \\ \text{D(i-1,j)} & + & \text{cost(1 :0 align } s_i, \emptyset) \\ \text{D(i-1,j-1)} & + & \text{cost(1 :1 align } s_i, t_j) \\ \text{D(i-1,j-2)} & + & \text{cost(1 :2 align } s_i, t_{j-1}, t_j) \\ \text{D(i-2,j-1)} & + & \text{cost(2 :1 align } s_{i-1}, s_i, t_j) \\ \text{D(i-2,j-2)} & + & \text{cost(2 :2 align } s_{i-1}, s_i, t_{j-1}, t_j) \end{cases} \tag{3.8}$$

Pour définir le coût associé à un certain alignement de phrases, Church et Gale utilisent une méthode basée sur la longueur des phrases, sous l'hypothèse que les traducteurs ont tendance à conserver la longueur des phrases constantes lorsqu'ils traduisent. Donc, plus les longueurs des phrases $\mathbf{s}$ et $\mathbf{t}$ sont proches, plus le coût associé à un tel alignement est faible. Dans le calcul de ce coût, on prend aussi en compte la probabilité des différentes relations d'alignement possibles, ainsi l'alignement 1 : 1 aura une probabilité beaucoup plus grande de survenir.

D'autres techniques pour calculer le coût associé aux alignements ont été présentées. Entre autre, Simard [21] utilise le *cognate* pour calculer les coûts. Deux mots sont dits cognates s'ils partagent une propriété commune à un quelconque niveau (sémantique, orthographique, phonographique, etc.). Pratiquement, Simard utilise les préfixes communs ou encore les parties numérales des mots pour déterminer les cognates.

L'alignement de phrases a fait l'objet de plusieurs compétitions au cours des années 90 et les performances atteintes au niveau de la précision et du rappel sont très proches de 100% [23]. Voilà pourquoi le domaine a été un peu délaissé par la recherche ces dernières années pour se concentrer sur les alignements de mots.

3.3.2 Les modèles IBM

L'approche naïve à la résolution de l'équation 3.1 est de passer toutes les phrases possibles de T dans le canal (voir section 3.1) pour vérifier si la phrase qui en ressort est $\mathbf{s}$. Or, nous savons qu'une telle approche est non seulement sous-optimale, mais consiste en un problème non décidable, puisqu'il existe une infinité de phrases dans la plupart des langues naturelles. De la même façon, un traducteur professionnel qui désire traduire une certaine phrase $\mathbf{s}$ ne fait pas appel à un traducteur professionnel qui traduit de l'anglais vers le français pour traduire toutes les phrases possibles de l'anglais dans le but de trouver la phrase traduction de $\mathbf{s}$.

Un traducteur professionnel essaie plutôt de comprendre les deux langues en jeu et d'établir une relation entre elles. C'est en utilisant sa compréhension des deux langues et de ce qui les relie qu'il arrive à produire une traduction plus facilement qu'en énumérant toutes les phrases possibles. C'est dans cet esprit que Brown et al. [2] ont introduit les désormais célèbres modèles de traduction IBM.

L'équation 3.1 divise le problème de la traduction automatique en trois parties. $p(\mathbf{t})$ représente la modélisation de la langue cible. $p(\mathbf{s}|\mathbf{t})$ représente la modélisation de la traduction de $\mathbf{s}$ vers $\mathbf{t}$. Puis $\underset{\mathbf{t}\in\mathcal{T}}{argmax}\ p(\mathbf{s}|\mathbf{t}) \times p(\mathbf{t})$ représente le problème de la recherche de la meilleure phrase $\mathbf{t}$ comme traduction de $\mathbf{s}$. La modélisation de la langue a été présentée au chapitre 3.2. Les modèles de traduction IBM ne concernent donc que l'aspect $p(\mathbf{s}|\mathbf{t})$, soit la traduction de $\mathbf{s}$ vers $\mathbf{t}$.

Les modèles IBM (1-5) sont des modèles de traduction basés sur l'alignement des

mots entre les phrases source et cible. L'alignement de mots consiste à établir quels mots d'une phrase cible traduisent quels mots d'une phrase source. La figure 3.2 donne un exemple d'alignement mot à mot de deux phrases.

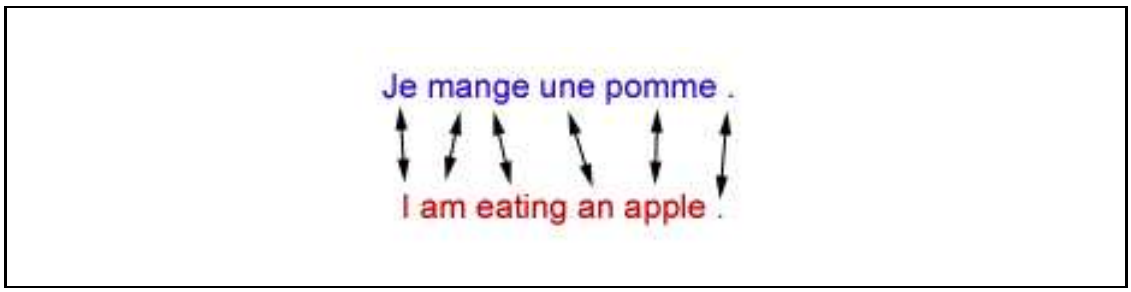

Fig. 3.2: Exemple d'un alignement entre deux phrases. Chaque flèche symbolise une relation de traduction entre les deux mots aux extrémités de cette flèche.

On note que la longueur de la phrase source est notée l, alors que la longueur de la phrase cible est notée m. L'ensemble des alignements possibles entre deux phrases est noté $\mathcal{A}(\mathbf{s}, \mathbf{t})$. Un alignement particulier est noté $a(\mathbf{s}, \mathbf{t})$. Ainsi, $\sum_{\mathbf{a} \in \mathcal{A}} p(\mathbf{a}(\mathbf{s}, \mathbf{t})) = 1$. L'équation 3.9 exprime la probabilité qu'une phrase $\mathbf{t}$ soit traduction de la phrase $\mathbf{s}$. À noter que nous inversons ici les textes source et cible. Nous passons donc d'une distribution $p(\mathbf{s}|\mathbf{t})$ vers une distribution $p(\mathbf{t}|\mathbf{s})$. Ayant inversé $\mathcal{T}$ et $\mathcal{S}$, on ne change rien à la distribution, il est seulement plus intuitif de parler de probabilité d'une phrase cible étant donné une phrase source.

$$p(\mathbf{t}|\mathbf{s}) = \sum_{\mathbf{a} \in \mathcal{A}} p(\mathbf{t}, \mathbf{a}|\mathbf{s}) \tag{3.9}$$

L'équation 3.10 donne donc, sans perte, la probabilité qu'une phrase $\mathbf{t}$ soit traduction d'une phrase $\mathbf{s}$ avec un certain alignement $\mathbf{a}$. a_j y représente l'indice du mot de la phrase source avec lequel le mot à l'indice j de la phrase cible est aligné et $p(m|\mathbf{s})$ la probabilité que la phrase cible ait une longueur de m étant donné la phrase source

$$p(\mathbf{t}, \mathbf{a}|\mathbf{s}) = p(m|\mathbf{s}) \times \prod_{j=1}^{m} p(a_j|a_i^{j-1}, t_1^{j-1}, m, \mathbf{s}) \times p(t_j|a_1^j, t_1^{j-1}, m, \mathbf{s}) \tag{3.10}$$

On peut voir dans l'équation 3.10 que chaque mot cible n'est aligné qu'avec un et

un seul mot source. Cette contrainte rend la modélisation plus simple. On ajoute un mot e_0 dans la phrase source qui permet d'exprimer qu'un mot cible soit la traduction d'aucun mot source. On peut voir, à travers l'équation 3.10, la façon dont le modèle génère les traductions :

- On choisit la longueur de la phrase cible étant donné la phrase source.
- Pour chaque position dans la phrase cible :
  - → On choisit l'alignement de cette position avec une position source étant donné les alignements des précédentes positions, les mots qui précèdent cette position dans la phrase cible, la longueur de la phrase cible et la longueur de la phrase source.
  - → On génère le mot cible à cette position étant donné les alignements des précédentes positions, les mots qui précèdent cette position dans la phrase cible, la longueur de la phrase cible et longueur de la phrase source.

Évidemment, comme bien souvent en modélisation statistique, le modèle sans perte est beaucoup trop grand et requiert beaucoup trop de ressources pour être implémenté. Dans ce cas-ci, les contextes conditionnant sont beaucoup trop grands et entraîneraient une quantité énorme de paramètres à estimer. Les modèles IBM (1-5) modifient tous à leur façon ce modèle général dans le but de le ramener à une forme implémentable sur nos ordinateurs et estimable à partir des données disponibles tout en tentant de représenter le mieux possible le phénomène de la traduction automatique.

IBM1

Le modèle IBM 1 fait des beaucoup de simplifications. On considère tout d'abord que la longueur de la phrase cible est indépendante de la phrase source, c'est-à-dire que chaque longueur de phrase cible a une probabilité égale. Ainsi, $p(m|\mathbf{s}) = \epsilon$. La valeur d'$\epsilon$ sera déterminé par la longueur maximale des phrases traitées. Ensuite, on ne considère

pas l'aspect aléatoire relié à l'alignement des deux phrases. Ainsi, un mot cible a une probabilité égale d'être traduit par un mot à n'importe quelle position de la phrase source, $p(a_j) = \frac{1}{l+1}$. Finalement, la probabilité d'un mot de la phrase source ne dépend que du mot avec lequel il est aligné. Ainsi, $p(t_j|s_{a_j}) = t(t_j|s_{a_j})$ où les $t(t_j|s_{a_j})$ sont les *probabilités de transfert*. Le modèle IBM 1 ne permet donc que d'estimer la probabilité qu'a un mot d'en traduire un autre.

Avec ces simplifications, le modèle IBM 1 s'exprime donc par :

$$\begin{aligned} p(\mathbf{t}, \mathbf{a}|\mathbf{s}) &= \epsilon \times \prod_{j=1}^{m} \frac{t(t_j|s_{a_j})}{(l+1)} \\ &= \frac{\epsilon}{(l+1)^m} \times \prod_{j=1}^{m} t(t_j|e_{s_j}) \end{aligned} \tag{3.11}$$

Pour obtenir la probabilité de $\mathbf{t}$ sachant $\mathbf{s}$, on somme sur tous les alignements possibles. Le résultat est présenté à l'équation 3.12.

$$\begin{aligned} p(\mathbf{t}|\mathbf{s}) &= \frac{\epsilon}{(l+1)^m} \sum_{a_1=0}^{l} \cdots \sum_{a_m=0}^{l} \prod_{j=1}^{m} t(t_j|s_{a_j}) \\ &= \frac{\epsilon}{(l+1)^m} \prod_{j=1}^{m} \sum_{i=0}^{l} t(t_j|s_i) \end{aligned} \tag{3.12}$$

La probabilité d'une phrase $\mathbf{t}$ d'être la traduction d'une autre phrase $\mathbf{s}$ est donc égale à la somme des probabilités des mots t_j de traduire chacun des mots s_i. C'est une modélisation assez simple qui mène à des résultats plutôt modestes.

Les paramètres du modèle sont les $t(t_j|s_i)$ pour toutes les paires de mots rencontrées dans le corpus d'entraînement. On utilise un algorithme de type EM pour entraîner un tel modèle. Les formules de réestimation des paramètres sont ici épargnées au lecteur et peuvent être consultées dans [3].

IBM2

Le modèle IBM2 est un modèle un peu plus réaliste que le modèle IBM1. IBM2 introduit des probabilités d'alignement, c'est-à-dire qu'on associe une probabilité d'ali-

gnement $a(i|j,l,m)$ à une position i dans la phrase source étant donné une position j dans la phrase cible, ainsi que les longueurs respectives des deux phrases, l et m. En d'autres mots, la position i dans la phrase source a une probabilité $a(i|j,l,m)$ d'être alignée avec la position j de la phrase cible étant donné des longueurs de l et m pour les phrases source et cible. Le modèle conserve l'uniformité sur la longueur de la phrase cible. Les équations 3.13 et 3.14 décrivent le modèle IBM2.

$$p(t,a|s) = \epsilon \times \prod_{j=1}^{m} a(a_j|j,l,m)t(t_j|s_{a_j}) \tag{3.13}$$

$$p(t|s) = \epsilon \times \prod_{j=1}^{m} \sum_{i=0}^{l} a(a_j|j,l,m)t(t_j|s_{a_j}) \tag{3.14}$$

L'avantage du modèle 2 sur le modèle 1 est marqué. La figure 3.3 montre un exemple d'alignement de Viterbi produit par un modèle IBM1. L'alignement de Viterbi est l'alignement le plus probable selon le modèle pour un couple de phrases source et cible. On peut voir que, dans le cas du modèle IBM1, étant donné que les probabilités d'alignement sont uniformes, un mot cible peut être en relation de traduction avec n'importe quel mot de la phrase source et ce, avec la même probabilité. C'est pourquoi les trois articles *a* de la phrase cible de la figure 3.3 sont alignés optimalement avec le premier article à survenir dans la phrase source, soit *une*. Ceci est une grande lacune du modèle IBM1. La figure 3.4 montre quant à elle un résultat obtenu avec un modèle IBM2. On peut voir que, grâce aux probabilités d'alignement, chacun des *a* (*an*) est "attiré" par l'article correspondant dans la phrase source. Par exemple le *a* qui précède *cookie* est à la position six dans la phrase cible, il a donc beaucoup plus de chance d'être aligné avec le *un* qui précède *biscuit* étant donné que ce dernier est à la position sept. Le modèle IBM 1 est en fait un cas particulier du modèle IBM 2 où toutes les probabilités d'alignement ont la même valeur. L'ajout de l'estimation de ces probabilités d'alignement fait en sorte que le modèle IBM 2 est un modèle plus performant que le modèle IBM 1.

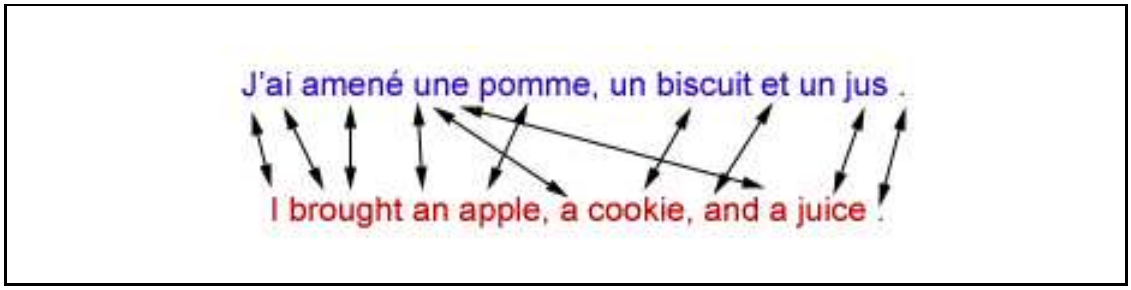

Fig. 3.3: Exemple d'alignement de Viterbi produit par le modèle IBM1.

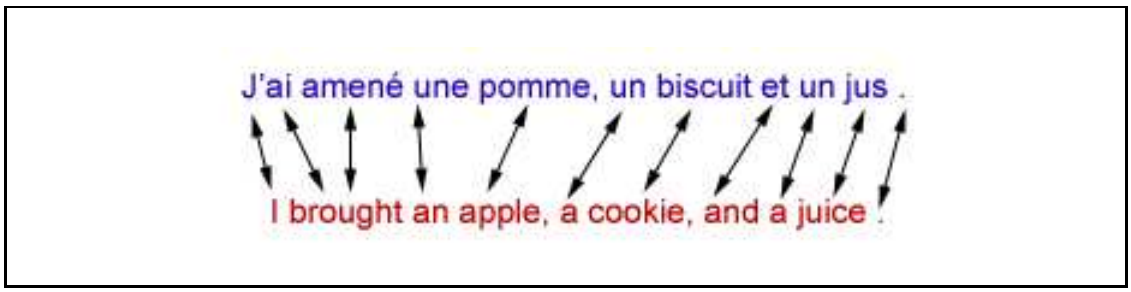

Fig. 3.4: Exemple d'alignement de Viterbi produit par le modèle IBM2.

IBM 3,4,5

Les modèles 3, 4 et 5 sont des améliorations des deux premiers modèles. Ils intègrent la notion de fertilité d'un mot. La fertilité (notée Φ) représente la distribution du nombre de mots cibles que génère un mot source. Ainsi, par exemple, les mots *potato* et *bill* auront une probabilité élevée pour une fertilité de valeur 3, puisqu'on les traduit généralement par *pomme de terre* et *projet de loi* respectivement. Ces trois modèles sont basés sur une décomposition différente que celle des modèles 1 et 2 pour intégrer la notion de fertilité.

C'est un modèle semblable au modèle IBM2 qui est présentement utilisé dans l'application TransType. Le modèle IBM2 a été modifié pour utiliser une technique de modélisation par entropie maximale. Les modèles d'entropie maximale sont présentés à la section 3.3.3.

3.3.3 Modèles Entropie Maximale Divergence Minimale

L'approche EMDM est une forme de modélisation statistique particulièrement utilisée pour la combinaison de sources d'information en un seul et unique modèle. Par exemple, dans le cas de modèles de traduction s'inspirant des modèles IBM, on veut combiner les sources d'information que sont les probabilités de transfert et les probabilités d'alignement. La méthode la plus répandue pour la combinaison de sources d'information est l'interpolation linéaire qui consiste à faire la somme de plusieurs modèles en les combinant à l'aide de coéfficients, par exemple $p(x|y) = \lambda_1 q(x|y) + \lambda_2 r(x|y)$. L'approche EMDM pallie certaines lacunes de l'interpolation linéaire comme moyen de combiner plusieurs sources d'information. Tout d'abord, l'interpolation linéaire fait une utilisation sous-optimale de ces sources étant donné que les poids attribués à chaque modèle sont optimisés globalement [20]. De plus, un modèle combiné par interpolation linéaire aura tendance à ne pas respecter les distributions marginales (distribution de probabilités obtenue en fixant le contexte conditionnant) de chacun des sous-modèles étant donné, encore une fois, que les poids sont optimisés globalement [20].

Les modèles EMDM ont la forme définie à l'équation 3.15. On y modélise le comportement d'une variable aléatoire X étant donné une autre variable aléatoire, Y. Par exemple, en traduction, on définirait X comme le prochain mot à prédire et Y comme le contenu de la phrase source.

$$p(x|y) = \frac{ref(x|y) \times exp(\sum_i \alpha_\mathbf{i} \times f_i(x,y))}{Z(y)} \tag{3.15}$$

$ref(x|y)$ correspond à la distribution de référence du modèle, α correspond au vecteur de poids du modèle, alors que les f_i définissent les fonctions de trait du modèle. Z est un facteur de normalisation qui dépend du contexte conditionnant, y.

La distribution de référence, $ref(x|y)$, contient l'information que nous avons *a priori* sur la distribution finale. Le vecteur de poids, α, contient les paramètres du modèle.

C'est ce vecteur qui permettra de définir notre distribution finale.

Dans un modèle EMDM, chaque source d'information définit un certain nombre de contraintes sur la distribution de probabilités du modèle. Ces contraintes sont de manière générale les distributions marginales étant donné le contexte conditionnant de chaque source d'information. Ces contraintes sont exprimées par l'entremise des fonctions de trait f_i.

Par exemple, si on tente de modéliser les mots qui peuvent suivre les mots *monsieur le* dans la langue française et que nous savons que 7% du temps le mot *président* suit les mots *monsieur le*. Alors, nous définirons la fonction de trait suivante dans notre modèle :

$$f(h,w) = \begin{cases} 1 & \text{si w=}\textit{président}\text{ et }\textit{monsieur le}\text{ sont les deux derniers mots de h} \\ 0 & \text{sinon} \end{cases} \tag{3.16}$$

On calcule ensuite l'espérance de ces fonctions de trait (équation 3.17) selon la distribution empirique, $\hat{p}(h,w)$, déterminée à partir d'un corpus d'entraînement. Pour la fonction de trait présentée à l'équation 3.16, l'espérance serait la valeur de la probablité que *président* suive *monsieur le* établie à 0.07.

$$\hat{p}(f) = \sum_{x,y} \hat{p}(x,y) \times f(x,y) \tag{3.17}$$

L'espérance de cette même fonction f selon, cette fois, la distribution recherchée est présentée à l'équation 3.18. On cherche alors une probabilité $p(x)$ qui respecte la contrainte suivante : $\hat{p}(f) = p(f)$ et ce, pour toutes les fonctions de trait. Ainsi, l'information de chaque fonction de trait sera inclue dans la distribution finale.

$$p(f) = \sum_{y} \hat{p}(y) \times p(x|y) \times f(x,y) \tag{3.18}$$

Parmi l'ensemble des fonctions probabilistes respectant les contraintes, on recherche alors la fonction ayant une divergence minimale avec la distribution de référence $ref(x|y)$

ou ayant une entropie maximale si $ref(x|y)$ est uniforme. La raison pour laquelle on recherche une telle fonction est que, mise à part l'information contenue dans les contraintes (les fonctions de trait), toute l'information qu'on a à propos de la distribution finale est contenue dans $ref(x|y)$. On cherche donc à approcher le plus possible de cette distribution tout en respectant les contraintes imposées par les différentes sources d'information [1, 7].

Les modèles de traduction implémentés dans TransType utilisant la technique d'entropie maximale divergence minimale sont présentés à la section 3.3.4.

3.3.4 Les modèles de TransType

MDI1

Le modèle MDI1 est un modèle à divergence minimale s'inspirant du modèle IBM1 présenté à la section 3.3.2. Sa forme générale est présentée à l'équation 3.19.

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h},\mathbf{s})exp(\vec{\alpha}\cdot\mathbf{f}(w,\mathbf{h},\mathbf{s}))}{Z(\mathbf{h},\mathbf{s})} \tag{3.19}$$

Pour tous les modèles MDI de TransType, la distribution $q(w|\mathbf{h},\mathbf{s})$ est un trigramme interpolé avec lissage Jelinek-Mercer (chapitre 3.2). Les sources d'information modélisées à travers les fonctions de trait f_i sont donc les informations de traduction. On cherche la distribution de probabilité, respectant les contraintes sur les fonctions de trait, qui s'approche le plus possible de la distribution du trigramme.

Dans le cas du modèle MDI1, on ne tient compte, comme dans le cas du modèle IBM1, que des traductions mot à mot de chaque paire de mots des phrases source et cible. Ainsi, les fonctions de trait ont la forme suivante :

$$f_{st}(w,\mathbf{h},\mathbf{s}) = \begin{cases} 1 & \text{si s} \in \mathbf{s} \text{ and } t = w \\ 0 & \text{sinon} \end{cases} \tag{3.20}$$

Chaque paire de mots source et cible s, t rencontrée dans le corpus d'entraînement

a une fonction de trait correspondante. On peut donc réécrire le modèle de la façon suivante :

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h})exp(\sum_{s\in\mathbf{s}}\alpha_{sw})}{Z(\mathbf{h},\mathbf{s})} \tag{3.21}$$

où α_{sw} est le poids attribué à la fonction de trait f_{sw}.

Ce modèle est donc très similaire au modèle IBM1 utilisé en conjonction avec un trigramme. L'équation 3.21 montre que la probabilité d'un mot w est proportionnelle aux poids α_{sw} pour tous les mots s de $\mathbf{s}$, ce qui est très similaire à la probabilité du même mot w dans le modèle IBM1. Les poids α_{sw} sont l'équivalent des probabilités de transfert $t(t_j|s_i)$.

L'utilisation du concept EMDM dans le modèle MDI1 lui permet d'être un modèle plus robuste et plus performant que le modèle IBM1 [8]. L'approche EMDM effectue un produit des probabilités de ses différentes sources, alors qu'un modèle IBM1 effectue une somme. C'est entre autres pourquoi un tel modèle est plus performant, puisqu'avec un produit, toutes les sources doivent donner une probabilité élevée.

Sélection de paires dans les modèles MDI

Les modèles MDI (MDI1 et MDI2B) sont soumis à une sélection de paires. Étant donné la taille du modèle en matière de paramètres à estimer et du temps requis pour faire une bonne estimation, on tente de réduire la taille du modèle en ne conservant que les paires qui sont utiles au modèle, c'est-à-dire les paires qui ont une certaine valeur de traduction. Par exemple, la paire (*lait*, *milk*) serait une paire considérée utile au modèle puisque bien souvent ces deux mots se traduisent l'un par l'autre. Par contre, la paire (*chaise*, *milk*) ne serait normalement pas considérée comme utile étant donné que jamais on ne traduira le mot *chaise* par le mot *milk*. Ainsi, il est donc préférable que le temps d'entraînement soit utilisé à l'estimation de poids de paires utiles au modèle

qu'à l'estimation de paires inutiles au modèle.

Pour décider des paires à inclure dans le modèle, on utilise un plus petit modèle, nommément IBM1, qui est plus rapidement entraîné. Pour chaque paire de mots du produit cartésien des vocabulaires source et cible, on effectue un calcul approché nous permettant d'estimer le gain, c'est-à-dire la baisse de perplexité (annexe B.1) qu'apporte la présence de cette paire dans le modèle. On fait ce calcul à tour de rôle pour chaque paire du modèle IBM1 et on calcule la différence de perplexité sur un corpus d'entraînement. Ainsi, les paires entraînant les plus fortes hausses de perplexité lorsqu'on les retire du modèle IBM1 sont choisies comme paires utiles au modèle de traduction.

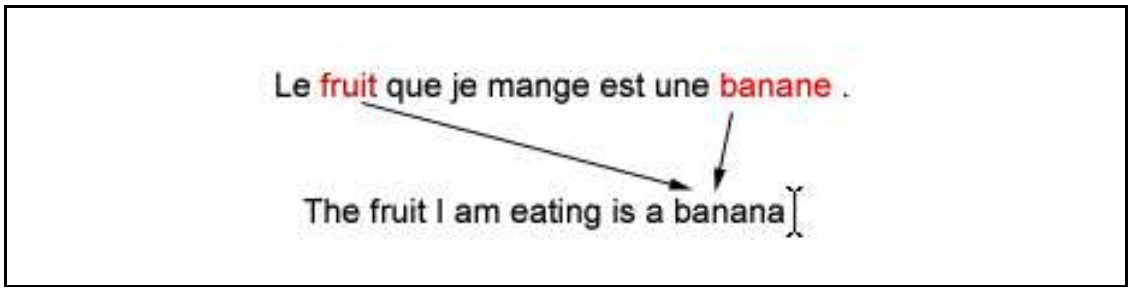

Fig. 3.5: Exemple de prédicition d'un modèle MDI de TransType. Le mot *banana* est prédit par le modèle. Les flèches représentent les paires actives lors de la prédicition.

La figure 3.5 montre un exemple de prédiction du modèle. La probabilité de *banana* dans ce contexte serait déterminée par la probabilité du trigramme *is a banana* et des poids des paires (*fruit, banana*) et (*banane, banana*). On voit aussi que toutes les autres paires (x, *banana*) ne sont pas considérées par le modèle puisqu'elles n'ont pas été retenues lors de la sélection de paires (parce qu'elles n'auraient rien amené au modèle). On nomme ces paires utilisées lors de la prédiction d'un mot *paires actives*.

MDI2B

Le modèle MDI2B s'inspire, quant à lui, du modèle IBM2. Il tente donc d'intégrer au modèle MDI1 les probabilités d'alignement[6] que l'on retrouve dans le modèle IBM2. L'alignement mot à mot des phrases source et cible dans le modèle IBM2 étant une

variable cachée, on ne peut l'intégrer directement dans un modèle EMDM. Seuls les événements observables peuvent y être intégrés puisqu'on doit connaître leur espérance dans le corpus d'entraînement.

Le problème qui survient avec l'intégration des probabilités d'alignement dans un tel modèle est celui de la sous-représentation due à un manque de données d'entraînement. Le modèle MDI2B introduit donc les probabilités d'alignement en utilisant des classes.

Ainsi, toutes les paires de mots s, t ayant des poids α_{st} similaires sont regroupées dans des classes dont les instances sont notées A. On regroupe aussi chaque configuration de position i, j, J (où J est la longueur de la phrase cible) ayant une probabilité d'alignement IBM2 semblable dans des classes dont les instances sont notées B.

Ceci permet d'obtenir des ensembles de configurations s, t, i, j, J avec une fréquence raisonnable pour en estimer une probabilité. La partition est faite de manière à obtenir une fréquence à peu près égale dans chaque classe.

Chaque paire de classes (A, B) a donc sa propre fonction de trait définie comme suit :

$$f_{A,B}(w, i, \mathbf{s}) = \sum_{j=1}^{J} \delta[(i, j, J) \in A \wedge (s_j, w) \in B \wedge j = \hat{j}_{s_j}] \tag{3.22}$$

où $\delta[X]$ est égale 1 si X est vrai, 0 sinon. De plus amples détails peuvent être obtenus dans [8].

Le modèle résultant est donc :

$$p(w|\mathbf{h}, \mathbf{s}) = \frac{q(w|\mathbf{h}) exp(\sum_{s \in \mathbf{s}} \alpha_{sw} + \alpha_{A(i,\hat{j}_s,J),B(s,t)})}{Z(\mathbf{h}, \mathbf{s})} \tag{3.23}$$

L'équation 3.23 montre que la probabilité d'un certain mot est proportionnelle aux poids α_{sw}, mais aussi à la classe de probabilités d'alignement dans laquelle se trouve le mot. Ce modèle s'est montré plus performant que le modèle MDI1 sur des ensembles de test du Hansard canadien [8] et est celui présentement utilisé dans les prototypes de

TransType.

Nous avons donc présenté dans ce chapitre une approche statistique à la traduction automatique. Se basant sur l'approche du canal bruité, la traduction statistique sépare les processus de traduction et de génération de la langue cible. Nous avons présenté ces deux processus aux sections 3.3 et 3.2. Les distributions de probabilités présentées dans ce chapitre sont statiques, c'est-à-dire qu'elles sont estimées lors de l'entraînement et qu'elles ne changent en aucun temps lors de leur utilisation. Nous présentons donc dans les deux chapitres suivantes l'adaptation dynamique des modèles de langue et des modèles de traduction. Cette adaptation permettra de modifier les distributions de probabilités en fonction de l'utilisation courante des modèles. Ces modifications permettront de mieux représenter les processus de traduction et de génération de la langue cible, entraînant ainsi de meilleures performances du système d'aide à la traduction.

Chapitre 4

Adaptation de modèles de langue

L'adaptation de modèles de langue consiste à adapter le modèle en fonction de la partie du document qui a déjà été analysée. Tel que mentionné au chapitre précédent, les modèles de langue classiques sont statiques. Ils n'utilisent donc pas l'information qui se trouve dans le texte couramment analysé. Or, il est reconnu que, de manière générale, dans un document, quelle que soit sa nature et sa longueur, l'auteur utilise un vocabulaire restreint. Ainsi, d'une phrase à l'autre et d'un paragraphe à l'autre, les mêmes mots et les mêmes tournures de phrase reviennent souvent. Il y a donc une très grande quantité d'information dans la partie d'un document déjà analysée qui n'est pas prise en compte par les modèles de langue classique. L'adaptation de modèles de langue tente de tirer partie de cette information.

La figure 4.1 schématise l'adaptation de modèles de langue dans le cas particulier du bigramme. On voudrait alors, plutôt que de n'utiliser que les deux mots qui précèdent pour prédire le mot courant, utiliser h, la partie du document déjà analysée, pour injecter de l'information dans le modèle et, ainsi, améliorer les prédictions.

Plusieurs techniques sont utilisées pour faire l'adaptation des modèles de langue. Parmi celles-ci, nous retrouvons les modèles de langue avec déclencheurs (*triggers*) [20], le regroupement par similitude (*topic-specific clustering*) [18] et les modèles de langue

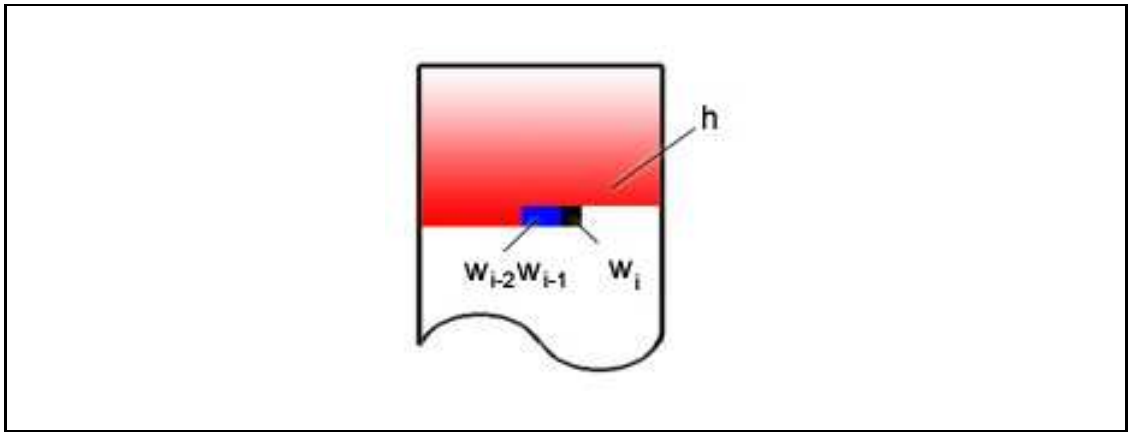

Fig. 4.1: Adaptation de modèles de langue. Le tout représente un document traité avec un modèle de langue. w_i représente le mot présentement prédit, alors que w_{i-2} et w_{i-1} sont les deux mots qui le précèdent. h, quant à lui, représente tout le texte déjà analysé.

cache [13].

Les modèles de langue avec déclencheurs

Ils consistent à établir une corrélation entre la présence d'un certain mot et d'un autre. Ainsi, pour une certaine paire de mots (A, B), la présence du mot A dans l'historique, H, vient augmenter la probabilité d'apercevoir le mot B. Des méthodes statistiques permettent de déterminer l'ensemble de paires adéquates permettant la plus grande amélioration du modèle de langue. Ils fonctionnent un peu comme les modèles de langue cache que nous décrivons plus loin, mais où chaque mot plutôt que d'augmenter sa propre probabilité de survenir augmente la probabilité d'un groupe de mots, dans lequel lui-même peut être présent. Ils permettent de façon générale une baisse de perplexité (voir section B.1) [20], puisque effectivement, dans la plupart des langues parlées ou écrites, la présence d'un certain mot dans une phrase nous indique la présence d'un autre mot dans cette même phrase. Ces modèles ont par contre une lacune au niveau de l'adaptation. C'est que les paires de mots sont déterminées au préalable sur le corpus d'entraînement qui est parfois très peu relié au domaine des textes qui utilisent le modèle. Donc, il est possible qu'une paire de mots soit très bonne en général (ou dans le corpus

d'entraînement), mais qu'elle ne s'applique pas du tout au contexte d'un certain document, ou encore qu'une paire ne soit pas choisie de façon générale (ou dans le corpus d'entraînement), mais soit excellente dans le contexte particulier d'un document. Par exemple, si on entraîne un modèle avec des débats parlementaires des années 2000, la paire de mots *guerre* et *Irak* sera probablement choisie par le modèle puisqu'on entend rarement parler de l'*Irak* sans entendre parler de *guerre*. Or, si un traducteur traduit un texte sur la flore irakienne ou encore sur la première guerre mondiale, on ne voudra alors pas que la présence du mot *guerre* entraîne une hausse de la probabilité du mot *Irak* et vice-versa. On verra plus loin qu'avec les modèles cache ce problème ne se présente pas.

Le regroupement par similitude

Il consiste à diviser le corpus d'entraînement en groupes de documents similaires. La métrique utilisée pour calculer la similitude peut être déterminée de plusieurs façons. Une des façons les plus courantes consiste à minimiser une fonction de perplexité pour l'ensemble des groupes. Nous entraînons un modèle de langue pour chacun des groupes. Puis, nous utilisons une mélange de modèles pour déterminer la probabilité d'un mot. Les coefficients de la mélange sont déterminés par un algorithme EM en fonction de l'historique H, c'est-à-dire que nous trouvons les coefficients qui minimisent la perplexité de H selon la mélange de modèles. Leur avantage marqué est qu'étant donné que nous utilisons la partie du corpus qui ressemble le plus à la langue présentement utilisée, celle-ci est mieux modélisée [5, 18] . Par contre, étant donné que nous divisons le corpus en N groupes, les modèles sont chacun entraînés sur moins de données, ils subissent donc une perte de performance à ce niveau.

Les modèles cache

Ces modèles sont étudiés de façon plus approfondie à la section suivante (4.1).

4.1 Les modèles de langue cache

Kuhn et De Mori [13] ont introduit le concept de modèle de langue cache qui permet l'adaptation dynamique de modèles de langue. Ces modèles fonctionnent de la même façon que les mémoires cache des ordinateurs. Les mémoires cache ont comme motivation le principe de localité qui dit que plusieurs accès mémoire consécutifs ont tendance à toujours accéder aux mêmes blocs en mémoire. De la même façon lorsque nous parlons et écrivons les langues naturelles, nous avons tendance à utiliser de phrases en phrases et de paragraphes en paragraphes un vocabulaire très similaire. D'abord, nous savons qu'un individu particulier utilise un vocabulaire très restreint, ainsi, les mêmes mots et tournures de phrase reviennent souvent dans un document. Aussi, un document traitant d'un sujet précis aura un vocabulaire propre à ce sujet. On voudrait donc que les mots de ce vocabulaire aient une probabilité plus élevée. La figure 4.2 montre une situation où un modèle cache sera très utile.

- Dans un système de reconnaissance de la parole, on analyse un son qui suit la séquence de mots *"the old"*.
- La composante acoustique du système retourne comme mots les plus probables pour ce son :

$$p(man) \simeq p(band)$$

- Le modèle trigramme de notre système donne :

$$p(man|the\ old) = 0.1$$

$$p(band|the\ old) = 0.01$$

- Dans cette circonstance, un modèle de reconnaissance de la parole ayant un modèle de langue statique opterait pour *"man"*.
- Mais si, dans les quelques phrases qui précédaient cette analyse :

$$|the\ old\ band| \gg |the\ old\ man|$$

- Intuitivement, on croirait que les chances sont plus grandes que ce soit *"band"* qui doive être prédit.
- C'est dans de telles situations que le modèle cache est puissant.

Fig. 4.2: Exemple motivant l'utilisation de modèles de langue cache. L'opérateur |x| représente la fréquence.

Les modèles de langue cache consistent donc à entraîner un modèle ngramme avec comme seul corpus d'entraînement l'historique H. De façon générale, nous utilisons une fenêtre glissante des N derniers mots de l'historique pour la cache. La langue qu'un tel modèle tente de modéliser est le sous-ensemble de la langue naturelle dans laquelle le texte est écrit ou parlé qui est présentement utilisé par l'auteur.

Comme pour les modèles de langue, les caches peuvent être unigrammes, bigrammes, trigrammes ou ngrammes. Kuhn et De Mori utilisent, quant à eux, un modèle 3g-gramme, une variante du ngramme conventionnel, comme modèle principal et une cache unigramme sur les POS (*part-of-speech*). Ils conservent donc les N derniers mots pour chaque POS. Un POS est une étiquette donnée à un mot représentant sa fonction syntaxique dans la phrase. DET et NOMC sont des exemples de POS permettant d'étiquetter respectivement un déterminant et un nom commun. Les POS sont notés g, alors que $\mathcal{G}$ représente l'ensemble de tous les POS utilisés lors de l'analyse syntaxique.

Le modèle qu'ils proposent est le suivant :

$$p(w_i|g_{i-2}g_{i-1}) = \sum_{g \in \mathcal{G}} p(w_i|g) \times p(g_i = g|g_{i-2}g_{i-1}) \tag{4.1}$$

où

$p(w_i|g)$ unigramme entraîné pour chaque POS

$p(g_i = g|g_{i-2}g_{i-1})$ trigramme où le vocabulaire est $\mathcal{G}$

Leur cache se situe au niveau du modèle $p(w_i|g)$ qui devient :

$$p(w_i|g) = k_{M,g} \times f(w_i|g_i = g) + k_{C,g} \times C_g(w_i, i) \tag{4.2}$$

où

$$k_{M,g} + k_{C,g} = 1$$

C_j constitue la composante cache de leur modèle. Pour chaque POS, on conserve les n derniers mots vus auxquels on a attribué ce POS. $C_j(w_i, i)$ est donc égal à la fréquence de w_i dans la cache pour le POS g_j divisé par n.

Kuhn et De Mori rapportent une amélioration de perplexité de presque 66%. La perplexité de leur modèle 3g-gramme passe de 332 à 107 sur une tranche de 200 000 mots d'un corpus nommé LOB (Lancaster-Oslo/Bergen), corpus très varié comprenant plusieurs type de textes (journalistique, scientifique, etc.). La perplexité (présentée à l'annexe B.1) est une mesure de la qualité d'une distribution probabiliste à représenter un phénomène aléatoire réel. On tente souvent de l'expliquer intuitivement en disant que c'est le nombre moyen d'essais que prend un modèle avant de trouver le bon choix lors de la prédiction d'un événement d'une certaine distribution. Le résultat de Kuhn et De Mori est donc excellent. Le modèle est en quelque sorte trois fois meilleur pour prédire les mots avec la composante cache que sans celle-ci.

Pour leur part, Martin, Liermann et Ney [18] utilisent une cache bigramme interpolée avec un modèle de langue trigramme. La cache bigramme consiste donc à conserver les N dernières séquences de deux mots.

$$p(w_i|h) = \lambda\, p_{tri}(w_i|w_{i-2}w_{i-1}) + (1-\lambda)\, p(w_i|w_{i-1}, h) \tag{4.3}$$

Le tableau 4.1 montre les résultats que rapportent Martin, Liermann et Ney avec leur cache bigramme. On voit que ce sont d'excellentes améliorations. Leur modèles ont été entraînés à l'aide de deux corpus soient un de quatre millions de mots et un de trente-neuf millions de mots. On voit que le trigramme entraîné sur trente-neuf millions de mots est moins aidé par la cache bigramme pour la simple et bonne raison qu'il est lui-même meilleur pour représenter la langue, il a donc moins de possibilités d'être amélioré.

	4M	%	39M	%
trigramme	152.1	-	96.8	-
+cache	120.5	-20.8%	82.8	-14.5%

TAB. 4.1 – Perplexités des modèles cache de Martin, Liermann et Ney. La colonne % représente la baisse de perplexité relative par rapport au modèle de base trigramme, ce pour des corpus de test de 4 et 39 millions de mots.

Clarkson et Robinson [5] ont quant à eux introduit une forme de composante cache très intéressante. C'est une cache à affaiblissement exponentiel. Ceci consiste en une composante cache normale ajoutée à un modèle ngramme, mais dont le poids associé à chaque élément dans la cache est exponentiellement décroissant en fonction de sa distance par rapport au mot présentement analysé. L'équation 4.4 définit formellement le modèle de Clarkson et Robinson.

$$p_{cache}(w_i|h) = \beta \sum_{j=1}^{i-1} I_{\{w_i=w_j\}} e^{-\alpha\times(i-j)} \tag{4.4}$$

où α est le taux d'affaiblissement, c'est-à-dire à quel point le poids de l'événement décroît en fonction de sa distance au mot courant, β est un facteur de normalisation et I est une fonction indicatrice qui vaut 1 si $w_i = w_j$, 0 sinon.

Ce modèle mathématique s'explique intuitivement par le fait qu'un mot w_1 qui se trouve dans l'historique à une distance de 1000 mots du mot présentement analysé a *a priori* une influence moins grande qu'un mot w_2 se trouvant à une distance de 10 mots du mot présentement analysé. De manière générale, on peut croire que le mot w_2 a beaucoup plus de chance d'être répété que le mot w_1.

Le tableau 4.2 montre les résultats obtenus par Clarkson et Robinson avec leur modèle cache à affaiblissement exponentiel. On peut y voir que leur modèle améliore un modèle trigramme de 14.4%, ce qui est un bon résultat. Par contre, on peut voir que le modèle à affaiblissement exponentiel n'améliore le modèle cache uniforme que de 2%.

	Perp.	%	%
baseline (trigramme)	165.52	-	-
+cache	144.73	-12.5%	-
+cache à affaibl. exp.	141.75	-14.4%	-2.0%

TAB. 4.2 – Perplexités des modèles cache de Clarkson et Robinson. La première colonne % donne la baisse de perplexité relative au modèle *baseline*, en pourcentage, alors que la seconde donne la baisse, en pourcentage, du modèle cache exponentiel par rapport au modèle cache non exponentiel.

USAGER	Donnez moi l'heure
RECON	*Donnez moi le beurre*
USAGER	Non je veux avoir l'heure
RECON	*Non je veux avoir le beurre*
USAGER	L'heure est grave
RECON	*Le beurre est gras*

Fig. 4.3: Problème de propagation d'erreurs dans les modèles de langue cache.

À première vue, les modèles de langue cache semblent donc très performants. Par contre, on remarque qu'ils ne sont pas universellement adoptés par les utilisateurs de modèles de langue, ce bien qu'ils amènent des améliorations d'environ 15% de la perplexité.

Le problème est que les résultats que présentent les auteurs cités plus haut sont théoriques et impliquent une correction des prédictions du modèle entre le moment où un mot est prédit et le moment où il se retrouve dans la cache. En d'autres mots, on assume que ce qui se trouve dans la cache est toujours bon, ce qui dans un contexte pratique n'est pas souvent possible. La figure 4.3 [14] montre le problème qui peut survenir avec les modèles cache dans un contexte pratique de reconnaissance de la parole. À la première phrase, le système de reconnaissance fait une erreur et reconnaît *le beurre* plutôt que *l'heure*. Cette erreur se retrouve alors dans la cache et contribue à l'erreur de la deuxième phrase. À la troisième phrase, étant donné qu'on a vu *le beurre* deux

fois dans les deux précédentes phrases, le système fait l'erreur pour une troisième fois de suite, car il assume que ce qui se trouve dans la cache est bon et se dit donc que les chances sont beaucoup plus fortes de parler de beurre que d'heure. De plus, cette erreur se répercute sur le reste de la phrase et le système reconnaît *gras* plutôt que *grave*, car *gras* est beaucoup plus probable dans un contexte de beurre.

Il y a donc un problème assez sérieux de propagation d'erreurs dans des systèmes réels utilisant des modèles de langue cache [11] comme la reconnaissance de la parole et la traduction automatique. Pour cette raison les résultats expérimentaux sont dans la plupart des cas bien moins bons que les résultats théoriques [18].

C'est exactement sur ce point que TransType gagne énormément à utiliser les modèles de langue cache. Comme on se trouve dans un contexte de traduction assistée par ordinateur où l'ordinateur et l'usager travaillent de pair, on est en plein droit d'assumer que ce qui est entré dans la cache est bon, car il y a correction de l'usager. Si le moteur de prédiction fait une mauvaise prédiction, on assume que celle-ci sera corrigée par l'usager. Ainsi, tout ce qui se trouve dans la cache sera toujours bon et on peut s'attendre à obtenir des résultats expérimentaux similaires aux résultats théoriques.

4.2 Résultats des modèles de langue cache dans un contexte unilingue

À la lumière des résultats présentés dans la littérature sur le sujet et résumés à la section 4.1, nous avons décidé de procéder à nos propres expérimentations avec les modèles de langue cache. Ces expérimentations nous permettent d'obtenir nos propres résultats et de confirmer la puissance des modèles de langue cache.

Nous avons utilisé comme modèle de base pour fin de comparaison un modèle trigramme utilisant un lissage de type Jelinek-Mercer tel que présenté au chapitre 3.2. Le modèle est donc de la forme :

$$\begin{aligned} p(w|h) &= \lambda_1(w_{i-2}w_{i-1}) \times p_{tri}(w_i|w_{i-2}w_{i-1}) \\ &+ \lambda_2(w_{i-2}w_{i-1}) \times p_{bi}(w_i|w_{i-1}) \\ &+ \lambda_3(w_{i-2}w_{i-1}) \times p_{uni}(w_i) \\ &+ \lambda_4(w_{i-2}w_{i-1}) \times p_{unif} \end{aligned} \tag{4.5}$$

où $\lambda_1(w_{i-2}w_{i-1}) + \lambda_2(w_{i-2}w_{i-1}) + \lambda_3(w_{i-2}w_{i-1}) + \lambda_4(w_{i-2}w_{i-1}) = 1$ et p_{unif} est une distribution uniforme $= \frac{1}{|V|+1}$ où V est l'ensemble des événements. Les paramètres des modèles trigrammes, bigrammes et unigrammes sont estimés par vraisemblance maximale, alors que les paramètres λ sont entraînés avec un algorithme *EM* qui trouve un ensemble de paramètres optimaux pour chaque historique $w_{i-2}w_{i-1}$.

Nous avons ensuite ajouté à ce modèle une composante cache. Nous avons testé des caches unigrammes, bigrammes et trigrammes, ainsi que différentes tailles de caches ($N = 100, 200, 500, 1000, 2000, 5000$).

Le modèle résultant de l'ajout d'une composante cache avait donc la forme :

$$\begin{aligned} p(w|h) &= \lambda_1(w_{i-2}w_{i-1}) \times p_{tri}(w_i|w_{i-2}w_{i-1}) \\ &+ \lambda_2(w_{i-2}w_{i-1}) \times p_{bi}(w_i|w_{i-1}) \\ &+ \lambda_3(w_{i-2}w_{i-1}) \times p_{uni}(w_i) \\ &+ \lambda_4(w_{i-2}w_{i-1}) \times p_{unif} \\ &+ \lambda_5(w_{i-2}w_{i-1}) \times p_{cache}(w_i|h_N) \end{aligned} \tag{4.6}$$

où h_N représente les N derniers mots de l'historique. La contrainte sur les poids λ est toujours présente.

Le lissage de Jelinek-Mercer que l'on utilise dans nos modèles permet effectivement de lisser les distributions grâce aux modèles d'ordre inférieur. La présence des modèles d'ordre inférieur (bigramme et unigramme dans le cas du trigramme) amène aussi une information additionnelle utile au modèle, même dans les cas où la probabilité retournée par le trigramme n'est pas nulle. C'est en s'inspirant de ce type de modèle que nous avons aussi testé un modèle qui comprend les trois types de cache, soit unigramme, bigramme et trigramme. Ce modèle a la forme :

$$
\begin{aligned}
p(w|h) &= \lambda_1(w_{i-2}w_{i-1}) \times p_{tri}(w_i|w_{i-2}w_{i-1}) \\
&+ \lambda_2(w_{i-2}w_{i-1}) \times p_{bi}(w_i|w_{i-1}) \\
&+ \lambda_3(w_{i-2}w_{i-1}) \times p_{uni}(w_i) \\
&+ \lambda_4(w_{i-2}w_{i-1}) \times p_{unif} \\
&+ \lambda_5(w_{i-2}w_{i-1}) \times p_{c-uni}(w_i|h_N) \\
&+ \lambda_6(w_{i-2}w_{i-1}) \times p_{c-bi}(w_i|h_N) \\
&+ \lambda_7(w_{i-2}w_{i-1}) \times p_{c-uni}(w_i|h_N)
\end{aligned} \tag{4.7}
$$

Les tableaux 4.3 et 4.4 présentent donc les perplexités obtenues par ces modèles sur nos corpora de test. Dans le premier cas, les tests ont été effectués sur la tranche **test** (50K phrases) du corpus Hansard français (présenté à l'annexe A.1), alors que dans le second cas les tests ont été effectués sur le corpus *sniper* (4K phrases, annexe A.2). Les poids λ optimaux ont été trouvés grâce à un algorithme EM en utilisant la tranche **train.ho2** (50K phrases) du corpus Hansard. Les modèles ngrammes originaux ont été créés avec la tranche **train.main** (1.6M phrases) du corpus Hansard.

La tranche **test** du corpus Hansard est constituée de textes qui n'ont pas été utilisés lors de l'entraînement des modèles. Par contre, le texte est très similaire à ceux employés pour l'entraînement. Dans le cas du corpus *sniper*, il est seulement utilisé pour des tests et contient un langage très différent de celui utilisé dans le corpus Hansard. Il permet donc de noter la puissance de tels models dans un contexte où le document analysé est très différent des documents du corpus d'entraînement.

Les résultats obtenus démontrent donc que les modèles cache entraînent effectivement une baisse de perplexité, dans notre cas par rapport à un trigramme de type Jelinek-Mercer. Ceci confirme donc les résultats constatés dans la littérature.

De façon générale, on observe que les meilleurs résultats (les plus grandes baisses de perplexité) sont obtenus avec la cache bigramme et avec la cache combinée unigramme, bigramme et trigramme. La cache bigramme fonctionne mieux que la cache unigramme pour les mêmes raisons qu'un modèle de langue bigramme fonctionne mieux qu'un modèle de langue unigramme. Le fait d'utiliser le dernier mot du contexte aide à mieux

Taille	UNI	%	BI	%	TRI	%	1+2+3	%
base	I3G(+CACHE)=61.0128							
100	59.76	-2.05%	58.00	-4.93%	59.46	-2.54%	57.20	-6.25%
200	59.56	-2.37%	57.26	-6.15%	58.91	-3.45%	56.33	-7.68%
500	59.56	-2.38%	56.69	-7.09%	58.33	-4.39%	55.69	-8.73%
1000	59.66	-2.22%	56.54	-7.33%	58.07	-4.83%	55.49	-9.05%
2000	59.77	-2.03%	56.52	-7.36%	57.85	-5.18%	55.39	-9.21%
5000	59.92	-1.79%	56.58	-7.27%	57.62	-5.55%	55.38	-9.24%

TAB. 4.3 – Perplexités des modèles de langue avec composante cache sur la tranche test du corpus Hansard. Les colonnes UNI, BI et TRI donnent les perplexités du modèle trigramme avec composante cache unigramme, bigramme et trigramme respectivement. Alors que la colonne 1+2+3 donne les perplexités du modèle trigramme avec caches unigramme, bigramme et trigramme interpolées. Les rangées donnent les différentes tailles de cache. Les colonnes % donnent les baisses de perplexité en pourcentage par rapport au modèle de base.

préciser la probabilité du mot à prédire. Le bigramme représente donc mieux la langue. Dans le cas de la cache trigramme, on peut voir qu'elle souffre de sous-représentation des données. C'est que plus on augmente l'ordre d'un modèle, plus les événements similaires sont rares. Dans le cas d'un modèle cache, comme on travaille avec un corpus d'entraînement très restreint (les N derniers mots), il devient rare qu'une suite de mots $(w_{i-2}w_{i-1}w_i)$ pour laquelle on désire obtenir une probabilité se retrouve dans la cache, donc les probabilités retournées par la composante cache sont le plus souvent nulles. Pour obtenir de bons résultats avec cette cache, il faut augmenter sa taille.

Pour sa part, la cache bigramme constitue donc un juste milieu entre une bonne représentation des données et une fréquence d'événements dans la cache assez grande. Ces deux éléments contribuent aux bonnes performances de ce modèle.

On peut par contre voir dans les résultats que les trois types de cache (unigramme, bigramme et trigramme) semblent apporter de l'information au modèle de langue. C'est pourquoi nous avons tenté l'expérience d'un modèle auquel on a ajouté les trois caches. Les résultats pour ce modèle sont concluants. Ceci s'explique par le fait que le modèle

Taille	**UNI**	**%**	**BI**	**%**	**TRI**	**%**	**1+2+3**	**%**
base	I3G(+CACHE)=540.080							
100	398.77	-26.17%	364.97	-32.42%	444.04	-17.78%	318.50	-41.03%
200	379.26	-29.78%	322.31	-40.32%	405.79	-24.86%	279.19	-48.31%
500	372.57	-31.02%	282.87	-47.62%	363.88	-32.62%	245.93	-54.46%
1000	398.77	-26.17%	364.97	-32.42%	340.31	-36.99%	229.05	-57.59%
2000	375.84	-30.41%	250.11	-53.69%	323.52	-40.10%	216.07	-59.99%
5000	384.07	-28.89%	238.50	-55.84%	306.35	-43.28%	205.65	-61.92%

TAB. 4.4 – Perplexité des modèles de langue avec composante cache sur la tranche test du corpus *sniper*. Les colonnes UNI, BI et TRI donnent les perplexités du modèle trigramme avec composante cache unigramme, bigramme et trigramme respectivement. Alors que la colonne 1+2+3 donne les perplexités du modèle trigramme avec caches unigramme, bigramme et trigramme interpolées. Les rangées donnent les différentes tailles de cache. Les colonnes % donnent les baisses de perplexité en pourcentage par rapport au modèle de base.

cache trigramme seul n'est pas bon, car les probabilités qu'il retourne sont trop souvent nulles, comme mentionné plus haut. Par contre, lorsqu'il retourne une probabilité forte pour un tel mot dans un certain contexte, alors nous devons croire que les chances que ce mot soit le bon sont fortes. Par exemple, si nous tentons d'obtenir la probabilité de $p(\text{président}|\text{monsieur le})$ et que la composante cache trigramme nous retourne une probabilité de 1, nous savons que *monsieur le* a toujours précédé *président* dans les N mots qui précédaient la présente analyse. On doit alors intuitivement considérer les chances de *président* d'être le bon mot comme étant très fortes. En conservant les modèles cache unigramme, bigramme et trigramme dans un même modèle, on évite les problèmes de sous-représentation des données et on va chercher le plus d'information possible pour améliorer la prédiction.

On remarque que les meilleurs résultats ne sont pas toujours obtenus avec les plus grandes tailles de cache. Ceci s'explique par le fait que plus on augmente la taille de la cache, plus on ajoute dans celle-ci des mots qui sont éloignés du mot présentement analysé. On se trouve donc à ajouter des mots contenant moins d'information que les

mots plus rapprochés. Comme on donne à tous les mots des poids égaux dans la cache, à un certain point, on dillue l'information contenue dans la cache.

On observe donc dans le cas du test sur le corpus Hansard une baisse de perplexité maximale de 7.36% pour le modèle cache bigramme et de 9.24% pour la cache combinant unigramme, bigramme et trigramme. Ces résultats démontrent que malgré le fait que les documents testés sont très similaires à ceux utilisés pour l'entraînement, l'utilisation de composante cache dans un modèle de langue améliore les prédictions du modèle. Ces modèles permettent donc de capter l'information prédictive contenue dans les quelques dernières phrases analysées d'un texte.

Dans le cas des tests effectués sur le corpus *sniper* les résultats sont d'autant plus évocateurs de la puissance des modèles cache. Dans ces tests, on utilise un modèle entraîné à partir du Hansard pour prédire un document tout à fait différent, soit un manuel pour apprendre le métier de tireur d'élite. On remarque que notre modèle de base obtient une perplexité 540 qui démontre bien la difficulté qu'a le modèle de base à prédire de tels documents. Les résultats démontrent que dans une telle situation, l'utilisation de modèles adaptatifs est d'autant plus importante. On voit que la perplexité est coupée de moitié dans le cas des configurations de modèles les plus performantes. Le modèle cache entraîne une baisse de perplexité de 55.84%, alors que le modèle cache combinant unigramme, bigramme et trigramme réduit la perplexité de près de 62%. Ces résultats sont comparables à ceux obtenus par Kuhn et de Mori [13].

Un avantage de ce modèle est le traitement des mots inconnus. Dans le cas du modèle de base, le trigramme interpolé, les mots inconnus ne sont pas traités, c'est-à-dire qu'il existe un mot spécial (UNK) qui a une certaine probabilité et c'est cette probabilité qui est attribuée à tous les mots inconnus du texte. Dans le cas du modèle cache, le vocabulaire est dynamique. Donc, les mots inconnus se retrouvent dans la cache comme tous les autres mots. Ainsi, si on rencontre un mot, auparavant inconnu, à quelques

reprises dans le texte, sa probabilité sera grandement améliorée étant donné qu'il se retrouve dans la cache.

Ces résultats expérimentaux sont en accord avec la littérature : l'adaptation des modèles de langue par les modèles cache est possible et mène à de bonnes réductions de perplexité et ce autant dans le cas de documents "connus" (dont le contenu, point de vue langue, est similaire au corpus d'entraînement) que dans le cas de document très peu "connus". Ces modèles de langue adaptatifs pourront donc être incorporés dans des systèmes de traduction pour rendre adaptative la tâche de traduction. Tel que mentionné, la nature de l'application TransType permet, contrairement à d'autres contextes pratiques, de tirer un plein avantage d'une telle adaptation. La section suivante (4.3) présente les résultats de ces modèles utilisés dans un système de traduction.

4.3 Résultats des modèles de langue cache dans un contexte de traduction

Tel que mentionné à la fin de la section 4.1, l'application TransType permet notamment, par sa nature interactive, de prendre avantage de l'adaptation dynamique des modèles de langue avec les modèles cache. Puisqu'il y a correction humaine des propositions du modèle, on peut assumer que tous les mots qui entrent dans la cache sont les bons, c'est-à-dire qu'ils sont ceux désirés par le traducteur. Cette section présente donc les tests que nous avons menés sur différents systèmes de traduction complets en faisant l'adaptation au niveau du modèle de langue. Cette adaptation est implémentée grâce aux modèles cache.

Cette série de tests consistait donc à tester un système de traduction complet avec une composante de langue adaptative et une composante de traduction statique.

Nous obtenons ici des résultats théoriques en calculant la perplexité de nos modèles sur des corpora de test. Cette perplexité (annexe B.1) nous permet de quantifier la

performance du modèle à bien représenter le processus de traduction d'une langue vers une autre.

Le chapitre 6 présentera une évaluation de ces mêmes modèles dans le cadre strict de TransType et ce en donnant une mesure de l'économie de frappes que peut entraîner l'utilisation de TransType.

4.3.1 IBM2 et modèle de langue cache

Le premier test que nous avons effectué consistait à utiliser nos modèles de langue trigrammes avec composante cache (section 4.1) en conjonction avec un modèle de traduction IBM2 (section 3.3.2).

Les modèles de langue utilisés dans ces expériences sont ceux présentés aux équations 4.6 et 4.7. Il consiste en un modèle trigramme de type Jelinek-Mercer interpolé avec une composante cache.

Nous avons testé les mêmes configurations de cache que celles présentées à la section 4.2, c'est-à-dire une cache unigramme, une cache bigramme, une cache trigramme, ainsi qu'une composante cache où nous avons interpolé chacune des trois caches.

Le modèle de traduction résultant est présenté à l'équation 4.8. Nous avons utilisé l'interpolation pour combiner les composantes de langue et de traduction. p_{i3g+c} représente le modèle de langue auquel nous avons ajouté une composante cache, alors que p_{ibm2} représente le modèle IBM2.

$$p(w_i|\mathbf{s}, h) = \mu \times p_{i3g+c}(w_i|h) + (1 - \mu) \times p_{ibm2}(w_i|\mathbf{s}) \tag{4.8}$$

Ces tests ont été effectués pour évaluer la performance des modèles de langue cache dans un contexte de traduction. Étant donné que les modèles de langue et de traduction sont interpolés, l'entraînement du modèle complet est simple.

Les résultats sont présentés aux tableaux 4.5 et 4.6. Nous avons utilisé un algorithme

Size	UNI	%	BI	%	TRI	%	1+2+3	%
base	IBM2+I3G(+CACHE)=35.246							
100	35.35	0.30%	34.64	-1.72%	35.02	-0.63%	34.62	-1.78%
200	35.35	0.28%	34.47	-2.19%	34.89	-1.01%	34.42	-2.33%
500	35.36	0.32%	34.36	-2.50%	34.72	-1.49%	34.28	-2.74%
1000	35.38	0.38%	34.36	-2.51%	34.64	-1.73%	34.26	-2.81%
2000	35.40	0.44%	34.39	-2.42%	34.57	-1.93%	34.25	-2.84%
5000	35.43	0.52%	34.47	-2.24%	34.51	-2.09%	34.26	-2.79%

TAB. 4.5 – Perplexités du modèle de traduction IBM2 interpolé avec un modèle de langue avec composante cache. Tests effectués sur la tranche **test** du corpus Hansard.

Size	UNI	%	BI	%	TRI	%	1+2+3	%
base	IBM2+I3G(+CACHE)=137.452							
100	120.60	-12.26%	110.16	-19.85%	125.45	-8.73%	102.68	-25.30%
200	117.24	-14.70%	101.82	-25.92%	118.41	-13.85%	94.65	-31.14%
500	116.29	-15.40%	95.31	-30.66%	111.55	-18.84%	88.80	-35.39%
1000	116.74	-15.07%	92.27	-32.87%	107.61	-21.71%	85.91	-37.49%
2000	117.27	-14.68%	89.97	-34.54%	104.88	-23.70%	83.63	-39.15%
5000	118.77	-13.59%	88.14	-35.88%	101.79	-25.95%	81.66	-40.59%

TAB. 4.6 – Perplexités du modèle de traduction IBM2 interpolé avec un modèle de langue avec composante cache. Tests effectués sur le corpus *sniper*.

de type EM pour trouver les μ optimaux et ce avec la tranche **train.ho2** du corpus Hansard (annexe A.1). Nous avons ensuite calculé la perplexité du modèle sur la tranche **test** du Hansard, ainsi que sur le corpus *sniper* (annexe A). Les modèles IBM2 et trigrammme ont été créés à partir de la tranche **train.main** du corpus Hansard.

La première chose que nous remarquons en regardant les résultats obtenus avec ces modèles est que les baisses de perplexité obtenues sont à peu près proportionnelles à celles obtenues lors du test des modèles de langue dans un contexte unilingue, présentées à la section 4.2. Ceci nous confirme donc que l'adaptation de modèles de langue dans un système de traduction entraîne une baisse de la perplexité sur un corpus de test. Tout comme dans le cas des modèles de langue, les deux types de cache les plus performants

sont les caches bigrammes et les caches unigramme, bigramme et trigramme interpolées avec des tailles de 2000 et 5000. On remarque que la baisse de perplexité se situe pour ces modèles aux alentours de 2.75% pour le test sur le corpus Hansard et de 40% sur le corpus *sniper*.

L'interpolation linéaire est reconnue pour être une technique de combinaison de sources d'information moins performante que la technique d'entropie maximale. À la lumière de ces résultats, il nous est donc permis d'espérer de meilleurs résultats en incorporant nos modèles de langue cache dans des modèles à entropie maximale.

4.3.2 MDI2B avec modèle de langue cache dans la distribution de référence

Nous avons ensuite incorporé nos modèles de langue cache dans le modèle MDI2B, un modèle à entropie maximale.

On se rappelle que les modèles MDI sont des modèles de type EMDM. Ces sont des modèles qui combinent les sources d'information sous la forme de fonctions de trait qui permettent d'exprimer des contraintes sur la distribution de probabilité recherchée. Cette distribution est celle se rapprochant le plus d'une distribution de référence. Dans le cas des modèles MDI de TransType, la distribution de référence est la distribution de probabilités de la langue cible sous la forme d'un trigramme interpolé.

Théoriquement, lorsque l'on change la distribution de référence, il faut faire un réentraînement complet du modèle MDI puisqu'il peut alors exister une nouvelle distribution plus approchée de la distribution de référence.

En ajoutant une composante cache à la distribution de référence, on rend celle-ci dynamique, c'est-à-dire qu'elle est maintenant dépendante de l'historique du document qu'elle analyse. Cette dépendance entraîne un très faible mouvement dans la distribution. La distance entre la distribution dynamique et la distribution originale est donc

Taille	UNI	%	BI	%	TRI	%	1+2+3	%
base	MDI2B[ref=I3G](+CACHE)=17.6584							
100	18.23	3.26%	17.29	-2.10%	17.35	-1.74%	17.56	-0.55%
200	18.11	2.56%	17.21	-2.54%	17.26	-2.26%	17.38	-1.57%
500	17.92	1.49%	17.13	-2.98%	17.18	-2.68%	17.17	-2.75%
1000	17.82	0.94%	17.10	-3.14%	17.16	-2.80%	17.08	-3.29%
2000	17.75	0.52%	17.09	-3.21%	17.16	-2.85%	17.00	-3.70%
5000	17.69	0.16%	17.05	-3.41%	17.13	-2.97%	16.92	-4.18%

TAB. 4.7 – Perplexités du modèle MDI2B interpolé avec une composante cache sur la tranche test du corpus Hansard.

très faible, puisque les deux distributions ne diffèrent que pour les ngrammes présents dans l'historique. De plus, cette différence n'est pas connue au moment de l'entraînement puisqu'elle est dépendante de l'historique du document analysé.

Nous avons donc effectué nos tests en ne réentraînant pas le modèle MDI2B. Nous avons ajouté directement la composante cache à la distribution de référence, sans réentraînement. Nous faisons donc l'hypothèse qu'un réentraînement mènerait à un changement négligeable des paramètres du modèle.

Ces tests ont donc consisté à remplacer la distribution de référence par les modèles de langue cache de la section 4.2, en utilisant encore une fois les mêmes configurations. Les résultats sont présentés aux tableaux 4.7 et 4.8. La perplexité est calculée sur la tranche **test** du corpus Hansard et sur le corpus *sniper*.

On remarque que les baisses de perplexités obtenues par ces modèles sont légèrement supérieures à celles obtenues par le modèle IBM2. Une fois de plus, ce sont les modèles cache bigramme et unigramme, bigramme et trigramme interpolés qui obtiennent les meilleurs résultats. Sur le corpus Hansard, on obtient une amélioration de 3.41% de la perplexité dans le cas du modèle cache bigramme et de 4.18% dans le cas du modèle cache unigramme, bigramme et trigramme interpolé. Alors que sur le corpus *sniper*, on obtient une baisse de perplexité de 47.36% pour le modèle cache bigramme et de 51.60%

Taille	UNI	%	BI	%	TRI	%	1+2+3	%
base	MDI2B[ref=I3G](+CACHE)=135.808							
100	125.47	-7.61%	102.45	-24.56%	117.56	-13.44%	101.96	-24.92%
200	118.05	-13.07%	92.80	-31.66%	108.82	-19.87%	90.50	-33.36%
500	112.45	-17.20%	83.53	-38.50%	99.37	-26.83%	80.08	-41.03%
1000	110.58	-18.57%	78.74	-42.02%	94.03	-30.76%	74.51	-45.14%
2000	108.71	-19.95%	74.99	-44.78%	90.17	-33.60%	90.50	-33.36%
5000	109.01	-19.73%	71.50	-47.36%	85.92	-36.73%	65.73	-51.60%

TAB. 4.8 – Perplexités du modèle MDI2B interpolé avec une composante cache sur le corpus *sniper*.

pour le modèle avec cache unigramme, bigramme et trigramme interpolées.

Ceci nous démontre que l'intégration de la composante cache dans la distribution de référence d'un modèle EMDM permet de tirer plus grand profit de la puissance des modèles de langue cache.

À la lumière de ces résultats, nous avons tenté un réentraînement complet de quatre configurations de modèles. Soit le modèle avec cache bigramme de tailles 2000 et 5000 et le modèle cache unigramme, bigramme et trigramme interpolé, aussi de tailles 2000 et 5000. Le réentraînement complet consiste en une réestimation de tous les poids des paires de mots du modèle, ainsi que des poids d'alignements. Cette réestimation est faite étant donné que la distribution de référence a changé. Un tel test est effectué pour vérifier s'il est possible d'obtenir une distribution qui, "en moyenne", est plus approchée de la distribution de référence avec composante cache. Le terme "en moyenne" est utilisé car la distribution de référence est constamment changeante étant donné qu'elle est dépendante de l'historique.

Les résultats de ces tests sont présentés aux tableaux 4.9 et 4.10.

On peut voir que le réentraînement a permis à la distribution de probabilité totale d'être légèrement plus rapprochée de la distribution de référence et ainsi offrir une meilleure performance. Les perplexités sont donc inférieures. On obtient une baisse

Taille	**BI**	**%**	**1+2+3**	**%**
base	MDI2B[ref=I3G(+CACHE)]=17.6584			
2000	16.937	-4.09%	16.840	-4.63%
5000	16.903	-4.28%	16.777	-4.99%

TAB. 4.9 – Perplexités du modèle MDI2B avec composante cache intégrée au modèle trigramme de référence avec réentraînement du modèle MDI2B complet. Entraînement effectué avec la trache **train.main** du corpus Hansard et tests effectués avec la tranche **test** de ce même corpus.

Taille	**BI**	**%**	**1+2+3**	**%**
base	MDI2B[ref=I3G(+CACHE)]=135.808			
2000	73.936	-45.6%	67.780	-50.1%
5000	70.514	-48.1%	64.204	-52.7%

TAB. 4.10 – Perplexités du modèle MDI2B avec composante cache intégrée au modèle trigramme de référence avec réentraînement du modèle MDI2B complet. Entraînement effectué avec la trache **train.main** du corpus Hansard et tests effectués avec le coprus *sniper*.

maximale de 4.99% dans le cas des tests sur le corpus Hansard et de 52.7% sur le corpus *sniper*.

On doit noter que les performances des modèles réentraînés sont biaisées, puisque nous avons utilisé les poids du modèle MDI2B de base pour débuter l'entraînement au lieu d'utiliser des poids nuls. Tous les poids du modèle ont donc été sujets à un plus grand nombre d'itérations d'entraînement. Ceci pourrait aussi expliquer la baisse de perplexité.

4.4 Discussion

Les résultats de la section précédente nous démontrent donc que l'ajout d'une composante cache dans un modèle de traduction MDI2B nous permet d'abaisser la perplexité du modèle de 5% pour des documents similaires au corpus d'entraînement et de 50%

pour des documents très différents du corpus d'entraînement.

Ces résultats montrent sans aucun doute la performance ainsi que la robustesse des modèles de langue cache. Les résultats de la section 4.2 avaient démontré la performance de ces modèles dans le cas unilingue. Nous avons ici obtenu des résultats similaires dans un cas bilingue dans lequel nos modèles agissent au niveau du texte cible. Les modèles de langue cache s'avèrent donc utilisables et efficaces dans un contexte de traduction.

La baisse de perplexité obtenue sur les corpora Hansard et *sniper* nous démontre bien que de l'information très utile à la prédiction de mots se retrouve dans l'historique récent du texte cible et que la façon dont sont conçus ces modèles cache nous permet d'en tirer profit à travers un modèle de traduction.

En analysant la baisse de perplexité obtenue dans le cas des tests effectués sur le corpus *sniper*, on remarque que pour des documents très différents du corpus d'entraînement, l'information contenue dans l'historique récent du texte cible est d'autant plus importante. Dans de telles situations, on pourrait considérer la composante cache comme étant le modèle de langue qui contient la véritable information concernant la langue cible et le trigramme de base comme venant lisser cette distribution de probabilités sur les événements du langage cible. La perplexité du modèle observée avec le corpus *sniper*, 64.2, n'est tout de même pas comparable à celle observée avec la tranche **test** du Hansard, 16.8. Cependant la baisse de perplexité obtenue avec le corpus *sniper* est de 50%, ce qui est considérable.

De plus, ces résultats nous démontrent à quel point il est utile et important d'utiliser des modèles adaptatifs dans le cas de documents très peu semblables au corpus d'entraînement.

En effet, comme il est très difficile d'obtenir des corpora suffisamment grands pour l'entraînement de modèles de traduction, les documents disponibles en grandes quantités sont souvent dans des domaines particuliers qui ne sont pas reliés aux documents

à traduire. Par exemple, les documents de notre corpus d'entraînement proviennent de débats parlementaires canadiens. L'utilisation de documents d'un domaine particulier implique que les modèles entraînés représentent bien la distribution du langage de ce domaine. Toutefois, on ne peut considérer que ces distributions permettront de bien représenter des langages associés à d'autres domaines plus ou moins distants du domaine d'entraînement. L'utilisation des modèles cache pallie en partie ce problème. Dans de tels circonstances, les résultats obtenus sont donc très encourageants et permettent d'entrevoir une utilisation de ces modèles dans des domaines variés sans voir les performances du système se dégrader trop rapidement.

Finalement, on note une hausse négligeable du temps de calcul de la traduction suite à l'ajout de composante cache dans les modèles.

Chapitre 5

Adaptation de modèles de traduction

Dans le chapitre précédent, l'adaptation du processus de traduction a été effectué au niveau unilingue seulement, c'est-à-dire que seulement l'historique du document cible est utilisé pour adapter le modèle. Nous avons donc tenté d'étendre au cas bilingue l'idée du modèle de langue cache. Un modèle de traduction adaptatif utilisera donc aussi l'historique du document source pour adapter le modèle. Ce chapitre décrit le modèle de traduction adaptatif que nous avons développé et présente ses résultats.

5.1 Le modèle MDI2BCache

Le modèle MDI2BCache est un modèle MDI2B auquel nous avons ajouté une composante cache au niveau des paires de mots, ainsi qu'une fonction de trait permettant d'exprimer la présence ou non d'une paire de mots dans la cache.

Tel que mentionné à la section 3.3.4, les modèles de traduction MDI sont basés sur les paires de mots. Ainsi, pour établir la probabilité d'un mot cible w, on utilise (entre autres) les poids des paires (w_s, w) et ce pour chacun des mots w_s de la phrase source.

Ainsi, plus les poids de ces paires sont élevés, plus le mot cible en question aura une probabilité élevée d'être le mot traduit dans le contexte donné. La section 3.3.4 montre, quant à elle, que ce ne sont pas toutes les paires de mots qui sont considérées. Pour réduire la taille du modèle et augmenter sa puissante prédictive, nous éliminons un très grand nombre de paires de mots puisqu'on sait *a priori* que ces paires ne seront pas utiles au modèle. On nomme ces paires utiles au modèle, paires actives. La figure 3.5 donne un exemple de paires actives considérées lors de la prédiction d'un mot. Dans ce cas-ci, les paires (*fruit*, *banana*) et (*banane*, *banana*) sont utilisées pour attribuer une probabilité au mot cible *banana*.

Notre modèle, au fil de la prédiction des mots d'un document, conserve donc toutes les paires actives rencontrées, c'est-à-dire toutes les paires qui ont servi à la prédiction de chacun des mots du document. Par exemple, dans le cas de la figure 3.5, si l'usager qui utilise présentement le modèle entre bel et bien le mot *banana*, les deux paires (*fruit*, *banana*) et (*banane*, *banana*) seront ajoutées dans la cache.

Cette cache est implémentée sous forme d'une structure de données de type queue et a une certaine taille N. Ainsi, on ne conserve dans la cache que les N dernières paires qui ont servi à faire la prédiction des mots du document.

On ajoute au modèle la fonction de trait suivante associée à la composante cache qui permet d'exprimer la présence dans la cache d'une certaine paire :

$$f_{cache\ st}(w, \mathbf{h}, \mathbf{s}) = \begin{cases} 1 & \text{si s} \in \mathbf{s},\ t = w,\ (s,t) \in \text{la cache et } \alpha_{st} > p \\ 0 & \text{sinon} \end{cases} \tag{5.1}$$

Cette fonction de trait ajoutée au modèle nous permet d'augmenter le poids α_{st} (par le biais du poids $\alpha_{cache\ st}$) d'une certaine paire (s,t) lors de la prédiction d'un mot t si cette paire a été vue (utilisée pour la prédiction d'un mot) parmi les N dernières paires rencontrées par le modèle. De façon intuitive, le poids d'une paire dans l'évaluation de la probabilité d'un mot est augmenté si cette paire a été vue récemment dans le

document. On se retrouve donc à tenter d'augmenter la probabilité des mots w pour lesquels il existe des paires (w_s,w) (pour tous les mots w_s de la phrase source courante) qui ont été vues dans l'historique récent du texte.

Dans le cas unilingue, si un modèle statique donne des probabilités similaires à deux mots de survenir dans un certain contexte, on voudra favoriser le mot qui a une plus grande fréquence dans l'historique récent du texte (figure 4.2). De la même façon, dans le cas bilingue, pour deux mots cibles ayant une probabilité à peu près égale de traduire un certain mot source, on voudra alors favoriser le mot cible qui a été vu en relation de traduction avec ce mot source dans l'historique récent du texte. C'est ce que tente de faire ce modèle.

Ainsi, si dans les quelques phrases tout juste traduites par l'usager, on retrouve le mot source *sound* qui a été traduit par le mot cible *sain*, alors la paire (*sound*, *sain*) se retrouve dans notre cache. Si une phrase source contenant le mot *sound* survient par la suite, on voudrait alors augmenter la probabilité relative du mot *sain* par rapport, par exemple, au mot *son*, une traduction plus courante du mot anglais *sound*), et ce puisque la paire (*sound*,*sain*) se retrouve dans la cache. Nous n'affirmons toutefois pas que la probabilité de *sain* sera de façon absolue plus grande que la probabilité de *son*, la cache n'entraînera qu'une augmentation relative de la probablité.

Nous avons inclus dans la fonction de trait une valeur plancher p sur le poids des paires à ajouter dans la cache. Une analyse rapide des poids α_{st} d'un modèle MDI2B nous permet de constater que la plupart des poids ayant de petites valeurs sont attribués à des paires contenant des mots utilitaires (par exemple (,, *and*) ou encore (*then*, *alors*)). On ne désire pas ajouter ces paires de mots dans la cache puisque ce ne sont pas des paires dont la présence dans une phrase augmente nécessairement la probabilité de présence dans les quelques phrases qui suivent. L'ajout de cette valeur plancher p joue un rôle similaire à la sélection de paires (section 3.3.4). On tente de n'ajouter dans la

cache que des paires qui contiennent de l'information quant à la prédiction des mots dans un futur rapproché du texte.

5.1.1 Poids des fonctions de trait cache

Nous avons implémenté deux versions de ce modèle. Dans la première version, nous avons attribué un seul et unique poids à toutes les fonctions $f_{cache\ st}$, c'est-à-dire que toutes les paires voient leur poids augmenté identiquement. Dans la deuxième version, chaque paire (s,t) a son propre poids $\alpha_{cache\ st}$.

La première version (équation 5.2), avec un seul et unique poids, est une version plus simple qui permet d'estimer plus facilement un poids α_{cache} optimal. On assume donc que toutes les paires ont la même probabilité de se répéter et que cette répétition contient la même quantité d'information pour toutes les paires.

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h})exp(\sum_{s\in\mathbf{s}}(\alpha_{sw} + \alpha_{A(i,\hat{j}_s,J),B(s,t)} + \alpha_{cache}))}{Z(\mathbf{h},\mathbf{s})} \tag{5.2}$$

Dans la deuxième version (équation 5.3), on ajoute autant de poids à estimer qu'il existe de paires dans le modèle, ce qui ajoute une difficulté au niveau de l'estimation des paramètres. Par contre, un tel modèle permet d'évaluer la propension qu'a une paire de se répéter d'une phrase à l'autre dans un texte. Ainsi. une certaine paire ayant un poids $\alpha_{cache\ st}$ élevé serait une paire qui aurait tendance à se répéter d'une phrase à l'autre, alors qu'une paire ayant un poids $\alpha_{cache\ st}$ petit serait une paire dont la présence à un certain point du texte nous donne peu d'information quant à la probabilité de cette paire de survenir prochainement dans le texte. On se trouve donc à augmenter le poids α_{st} proportionnellement à la propension qu'a une paire de se répéter dans un texte.

$$p(w|\mathbf{h},\mathbf{s}) = \frac{q(w|\mathbf{h})exp(\sum_{s\in\mathbf{s}}(\alpha_{sw} + \alpha_{A(i,\hat{j}_s,J),B(s,t)} + \alpha_{cache\ sw}))}{Z(\mathbf{h},\mathbf{s})} \tag{5.3}$$

5.1.2 Alignement de Viterbi pour l'ajout de paires dans la cache

Nous avons aussi implémenté une version du modèle où nous effectuons une sélection lors de l'ajout de paires dans le modèle, sélection qui s'ajoute à celle effectuée par la valeur plancher p. Cette sélection se fait avec un alignement de Viterbi des phrases source et cible avant l'ajout dans la cache. Plutôt que d'ajouter toutes les paires actives ayant un poids supérieur au seuil p, on effectue un alignement de Viterbi sur les phrases source et cible et on n'ajoute dans la cache que les paires contenues dans cet alignement.

Un alignement de Viterbi consiste à aligner chacun des mots de la phrase cible avec un mot de la phrase source correspondante de manière à maximiser la probabilité combinés de tous les mots de la phrase cible. Pour trouver cet alignement de Viterbi, nous avons utilisé les probabilités d'alignement d'un modèle IBM2 ainsi que les poids des paires de mots. Les probabilités d'alignement d'un modèle IBM2 sont définies comme suit : $a(i|j,l,m)$ est la probabilité qu'un mot en position i dans la phrase source soit en relation de traduction avec un mot en position j de la phrase cible étant donné une phrase source de taille l et une phrase cible de taille m. L'alignement de Viterbi consiste donc à trouver pour chaque mot cible le mot source associé qui maximise la probabilité suivante : $a(i|j,l,m) * \alpha_{st}$ pour une paire (s,t) où s est à la position i dans la phrase source et t est à la position j dans la phrase cible.

Une grande difficulté de la traduction par ordinateur se situe au niveau de la détermination de quel(s) mot(s) source(s) tradui(sen)t quel(s) mot(s) cible(s) et vice-versa (l'alignement des mots). Déterminer mathématiquement cet alignement avec une bonne précision est très compliqué, puisque même un être humain ne peut le faire parfaitement. Par exemple, si on donne les mêmes deux phrases (source et cible) à deux traducteurs différents et qu'on leur demande d'aligner les mots de deux phrases, il est très possible que les alignements produits par l'un et l'autre des traducteurs ne soient pas les mêmes. Étant donné qu'il est impossible de déterminer exactement quel mot tra-

duit quel autre entre deux phrases, la plupart des modèles de traduction font intervenir tous les mots sources pour prédire un mot cible, c'est le cas du modèle MDI2B.

Cette difficulté est aussi présente dans notre modèle cache bilingue. En ajoutant des paires de mots dans la cache, on ne peut jamais être certain que les deux mots que nous ajoutons étaient bel et bien en relation de traduction dans le texte. De façon optimale, nous ne voudrions ajouter dans la cache que les mots qui sont véritablement en relation de traduction, chose impossible de la même façon que déterminer l'alignement exact des mots de phrases source et cible [19].

L'alignement de Viterbi permet d'obtenir un ensemble de paires pour lesquelles il y a une forte probabilité que les mots aient été en relation de traduction. Ainsi, en n'ajoutant dans la cache que les mots faisant partie de l'alignement de Viterbi, on s'assure de n'ajouter que (ou presque) des paires de mots bel et bien en relation de traduction. Par contre, il est possible que certains mots qui devraient être dans la cache ne s'y retrouvent pas. Cela peut survenir lorsqu'un mot cible est en relation de traduction avec plus d'un mot source ou encore lorsque l'alignement de Viterbi est erroné.

5.2 Résultats du modèle MDI2BCache

Le tableau 5.1 présente les résultats obtenus avec la première version de notre modèle. Cette version ajoutait dans la cache toutes les paires ayant un poids supérieur ou égal au seuil p et ne contenait qu'un seul poids cache pour toutes les paires. Ce poids a été entraîné avec la tranche **train.ho2** et les perplexités sont calculées sur la tranche **test** du corpus Hansard (annexe A.1).

Nous avons testé les poids seuils de 0.3, 0.5 et 0.7. Ceux-ci engendrent un retranchement de 25%, 50% et 75% des paires du modèle. Par exemple, avec un poids seuil de 0.3, 75% des paires étaient considérées pour l'ajout dans la cache.

On obtient le meilleur résultat avec une cache de taille de 1000 et un poids seuil de

Taille	**0.3**	**%**	**0.5**	**%**	**0.7**	**%**
base	MDI2B=17.6584					
1000	17.568	-0.51%	17.576	-0.47%	17.598	-0.34%
2000	17.570	-0.50%	17.577	-0.46%	17.598	-0.34%
5000	17.574	-0.48%	17.578	-0.46%	17.597	-0.35%
10000	17.578	-0.46%	17.579	-0.45%	17.596	-0.35%

TAB. 5.1 – Perplexités des modèles de traduction cache MDI2BCache. Version avec un seul poids cache. Les colonnes représentent les différents seuils p testés alors que les rangées représentent les tailles de cache testées. Tests effectués sur la tranche **test** du corpus Hansard.

Taille	**0.3**	**%**	**0.5**	**%**	**0.7**	**%**
base	MDI2B=17.6584					
1000	17.581	-0.43%	17.586	-0.41%	17.606	-0.29%
2000	17.593	-0.37%	17.592	-0.38%	17.606	-0.30%
5000	17.585	-0.42%	17.587	-0.40%	17.608	-0.29%
10000	17.589	-0.39%	17.589	-0.39%	17.607	-0.29%

TAB. 5.2 – Perplexités des modèles de traduction cache MDI2BCache. Version avec un poids cache par paire. Tests effectués sur la tranche **test** du corpus Hansard.

0.3, soit une amélioration de 0.51% de la perplexité du modèle. Il nous est permis de constater que la taille de la cache ne semble pas influencer beaucoup la performance du modèle. D'autre part, le seuil de 0.7 fait en sorte que plusieurs paires qui devraient être ajoutées à la cache ne le sont pas.

Le tableau 5.2 présente, quant à lui, les résultats de notre deuxième version de modèle. Dans cette version du modèle, nous avons attribué un poids cache à chaque paire. Le modèle a été entraîné et testé de la même façon que pour la première version, soit avec les tranches **train.ho2** et **test** du corpus Hansard.

D'après nos résultats, il ne semble pas favorable d'attribuer un poids cache à chaque paire du modèle. Encore une fois, la meilleure configuration est celle avec un poids seuil de 0.3 et une cache de taille 1000, soit une amélioration de 0.43%. Ces performances

Taille	**0.3**	**%**	**0.5**	**%**	**0.7**	**%**
base	MDI2B=17.6584					
1000	17.560	-0.56%	17.570	-0.50%	17.594	-0.36%
2000	17.568	-0.51%	17.569	-0.50%	17.590	-0.39%
5000	17.561	-0.55%	17.569	-0.51%	17.592	-0.37%
10000	17.565	-0.53%	17.569	-0.51%	17.591	-0.38%

TAB. 5.3 – Perplexités des modèles de traduction cache MDI2BCache. Version avec un poids cache par paire et alignement de Viterbi. Tests effectués sur la tranche **test** du corpus Hansard.

moindres s'expliquent par la difficulté d'estimer de tels poids lors de l'entraînement. Comme mentionné à la section 5.1, le fait d'attribuer un poids cache à chaque paire permet d'exprimer la propension d'une paire à se répéter. Or, les résultats obtenus nous montrent qu'il est difficile d'estimer de tels poids, du moins avec le type d'entraînement effectué.

La dernière version de notre modèle comprend un alignement de Viterbi préalable à l'insertion dans la cache. C'est-à-dire que nous réalisons un alignement de Viterbi sur chaque paire de phrase avant d'ajouter les paires dans la cache. Cet alignement de Viterbi nous permet d'obtenir pour chaque mot cible le mot source le plus probablement en relation de traduction avec ce dernier. Ceci nous permet d'avoir plus de précision sur ce qui entre dans la cache. Les résultats de ce modèle sont présentés au tableau 5.3.

Les résultats de ce modèle sont légèrement supérieurs à ceux obtenus sans l'alignement de Viterbi. Le meilleur résultat est encore une fois obtenu avec le modèle ayant un poids seuil de 0.3 et une cache de taille 1000, soit une baisse de 0.56% de la perplexité. Ceci nous montre qu'il est préférable d'ajouter dans la cache des paires de mots pour lesquels la probabilité qu'ils soient réellement en relation de traduction est grande. Sans alignement de Viterbi nous ajoutions toutes les paires considérées lors de la prédiction d'un mot, alors qu'avec l'alignement de Viterbi nous ajoutons seulement une paire, la plus probable, par mot cible. On voit donc que ce sont ces paires qui contiennent l'infor-

Taille	**0.3**	**%**	**0.5**	**%**
base	MDI2B=135.808			
1000	132.86	-2.17%	132.75	-2.25%
2000	132.77	-2.23%	132.75	-2.25%
5000	132.73	-2.26%	132.63	-2.34%
10000	133.00	-2.07%	132.67	-2.31%

TAB. 5.4 – Perplexités des modèles de traduction cache MDI2BCache. Version avec un poids cache par paire et alignement de Viterbi. Tests effectués sur le corpus *sniper*.

mation désirée et que certaines paires ajoutées dans la cache sans alignement de Viterbi viennent plutôt nuire au modèle.

La tableau 5.4 montre finalement les résultats obtenus sur le corpus *sniper*. Nous avons ici repris seulement le modèle ayant donné les meilleurs résultats avec la tranche **test** du corpus Hansard, soit celui avec un seul poids pour toutes les paires, un alignement de Viterbi et un poids seuil de 0.3 ou 0.5. On voit que, de la même façon qu'avec le modèle cache unilingue, le modèle cache bilingue obtient de meilleurs résultats sur des documents très différents de ceux utilisés pour l'entraînement. La baisse de perplexité maximale est de 2.34% dans le cas d'un modèle cache de taille 5000 et un poids seuil de 0.5. Il est à noter que le modèle obtient une baisse relative de perplexité quatre fois plus grande sur le corpus *sniper* que sur le corpus Hansard.

5.3 Discussion

Les résultats obtenus par ces modèles nous permettent certainement d'affirmer que de l'information pouvant améliorer la prédiction d'un mot dans un modèle de traduction se trouve dans les paires récemment rencontrées lors de la traduction d'un texte. Chacun des modèles testés obtient une perplexité inférieure à celle du modèle de base. Par contre, ces baisses de perplexité ne sont pas aussi élevées que ce que l'on espérait.

Pour tenter d'expliquer ces résultats, nous avons effectué une analyse du contenu de

base	**0.3**	**viterbi + 0.3**
(*is*,*qu'*)	(*to*,*afin*)	(*offence*,*crime*)
(*.*,*sa*)	(*was*,*a*)	(*was*,*été*)
(*this*,,)	(*UNK*,*UNK*)	(*very*,*très*)
(*all*,*toutes*)	(*piece*,*législative*)	(*today*,*aujourd'hui*)
(*have*,*du*)	(*this*,*ce*)	(*jobs*,*emploi*)
(*the*,*pour*)	(*per*,*100*)	(*concern*,*inquiétude*)
(*on*,*du*)	(*that*,*soient*)	(*skin*,*peau*)
(*of*,*un*)	(,,,)	(*there*,*y*)
(*we*,*nous*)	(*?*,*il*)	(*government*,*le*)
(*the*,*du*)	(*any*,*tout*)	(*an*,*un*)
18	68	86

Fig. 5.1: Échantillon du contenu de la cache pour différentes configurations du modèle. La dernière ligne indique le nombre de paires jugées pertinentes dans un échantillon de 100 paires se retrouvant dans la cache. Les trois colonnes représentent les différents modèles testés. Un modèle sans poids seuil, un modèle avec 0.3 de poids seuil et un modèle avec poids seuil de 0.3 et avec alignement de viterbi. Les échantillons sont peuplés à partir du corpus hansard.

la cache. Nous avons donc regardé les paires de mots entrant dans la cache. Le but de cette analyse était de voir à quel point le contenu de la cache était susceptible d'amener de l'information au modèle pour prédire les mots cibles.

La figure 5.1 montre donc un échantillon de dix mots pris aléatoirement dans la cache et ce pour trois configurations différentes de nos modèles. La colonne de gauche montre l'échantillon obtenu avec un modèle où toutes les paires de mots sont ajoutées dans la cache avec un poids seuil de -999. C'est donc dire qu'absolument toutes les paires sont ajoutées à la cache. La colonne du centre montre un échantillon tiré de la cache avec un modèle similaires ayant un poids seuil de 0.3. Finalement, la colonne de droite montre l'échantillon obtenu avec un modèle ayant un seuil de 0.3 et effectuant un alignement de Viterbi avant l'ajout dans la cache. Pour chacun de ces trois modèles, nous avons aussi pris un échantillon de 100 paires de mots et, à la main, nous avons évalué chacune de ces paires pour savoir si, oui ou non, ces deux mots pouvaient logiquement être

considérés pertinentes, c'est-à-dire contenant de l'information utile à la prédicition des mots futurs. Par exemple, la paire (*of*,*un*) est considérée non pertinente, alors qu'une paire (*offence*,*crime*) est jugé pertinente. Ce résultat est présenté à la dernière ligne du tableau de la figure 5.1.

Cette analyse nous démontre donc que les paramètres des modèles ayant obtenu les meilleurs résulats au niveau de la perplexité engendrent aussi une plus grande pertinence des paires dans la cache. On voit que, de gauche à droite dans le tableau, la qualité des paires augmente. Ainsi, la plupart des paires se retrouvant dans la cache pour notre modèle avec un seuil de -999 semblent comporter très peu d'information sinon aucune. Au centre, les paires semblent légèrement plus pertinentes, mais des paires telles (*UNK*,*UNK*) et (,,,) sont des paires utiles pour un modèle de traduction complet, mais pas pour une composante cache. Finalement, dans la dernière colonne à droite, la plupart des paires semblent pertinentes. Intuitivement, ce sont de telles paires que l'on désire dans une cache. De plus, les chiffres se retrouvant à la dernière ligne concordent avec la précédente analyse. C'est le modèle effectuant un alignement de Viterbi qui obtient le meilleur : 86 des 100 paires contiennent des mots qui peuvent être logiquement mis en relation de traduction et contiennent de l'information utile aux prédicitions futures.

On peut faire un rapprochement entre le problème d'une telle composante cache dans un modèle de traduction et les métriques de précision et de rappel en recherche d'information. D'une part, on désire inclure dans la cache toutes les paires qui sont véritablement en relation de traduction dans le document et d'autre part, on désire que, parmi les paires ajoutées dans la cache, le plus grand nombre soit des paires en relation de traduction.

De plus, idéalement, on ne désire dans la cache que des paires pour lesquelles les mots en question se retrouvent aussi en relation de traduction avec d'autres mots dans les deux langages du modèle. Par exemple la paire (*today*,*aujourd'hui*), bien qu'une paire

très importante pour le modèle de traduction complet, n'est pas très utile dans une composante cache puisque ces deux mots sont très rarement traduits par d'autres mots, le modèle a déjà une facilité à traduire un mot comme *today*.

Cet idéal au niveau des paires présentes dans notre composante cache est donc le plus approché par notre modèle ayant un poids seuil de 0.3, une taille de 1000 et effectuant un alignement de Viterbi.

Lorsque l'on compare les résultats obtenus par le modèle cache bilingue à ceux obtenus avec le modèle cache unilingue (chapitre 4), on remarque que les résultats au niveau bilingue sont beaucoup moins intéressants. Un avantage qu'a le modèle cache unilingue par rapport à celui bilingue est le traitement des mots inconnus. Dans le cas de la cache unilinge, lorsqu'un mot inconnu est rencontré, il est ajouté au vocabulaire du modèle et est ajouté dans la cache comme tous les autres mots. Par contre, dans le cas bilingue, nous ne traitons pas les mots inconnus, en fait ceux-ci restent inconnus et le mot spécial *UNK* (mot représentant l'ensemble des mots inconnus dans le vocabulaire du modèle) peut se retrouver dans la cache faisant partie d'une paire. Nous n'avons pas traité les mots inconnus puisque l'architecture des modèles utilisés s'y prêtait mal, mais nous croyons qu'un tel traitement pourrait être bénéfique pour un modèle de traduction adaptatif.

En somme, les résultats démontrent le potentiel de l'adaption bilingue de modèles langagiers. Nous croyons qu'il est possible d'améliorer les résultats obtenus dans ce modèle, soit par une modification d'un modèle cache sur les paires ou encore par un ré-ingénierie complète du modèle pour permettre, entre autres, le traitement des mots et des paires inconnus. Nous proposons, à la section 5.3.1, quelques modifications pouvant être apportées à un modèle cache sur les paires.

Encore une fois, on note des hausses négligeables du temps de calcul de la traduction suite à l'ajout de composantes cache bilingues à nos modèles.

5.3.1 Améliorations possibles au modèle MDI2BCache

Voici quelques modifications qui pourrait être apportées au modèle MDI2BCache. Nous estimons que ces modifications permettraient de s'approcher de l'idéal au niveau des paires présentes dans la cache, entraînant ainsi une améliration des performances.

Une première amélioration se situerait au niveau de l'alignement de Viterbi. Nous avons implémenté pour ce mémoire deux versions de modèle : une première version qui ajoutait dans la cache toutes les paires rencontrées et une deuxième version qui ajoutait seulement les paires résultant de l'alignement de Viterbi. Il serait intéressant de voir s'il serait possible d'améliorer les performances du modèle en ajoutant dans la cache un certain nombre de paires qui se situerait à mi-chemin entre l'alignement de Viterbi et toutes les paires. Ce dans le but de maximiser le nombre de paires se retrouvant dans la cache qui sont bel et bien en relation de traduction tout en minimisant l'ajout dans la cache de celles qui ne le sont pas.

Une autre amélioration possible à notre modèle se situerait au niveau des poids des fonctions de trait. Nous avons aussi implémenté deux versions de notre modèle : une première dans laquelle nous avons attribué un seul poids cache à toutes les paires et une seconde dans laquelle nous avons attribué un poids cache à chaque paire du modèle. Nous supposons dans le premier cas que toutes les paires ont une même propension à se répéter dans un texte, tandis que dans la seconde nous exprimons à travers les poids la propension de chaque paire à se répéter. Or, il semble que la deuxième version du modèle ait souffert, entre autres, de sous-représentation des données, puisque ce modèle n'a pas obtenu des meilleures résultats que le modèle avec un seul poids pour toutes les paires. Une voie d'exploration possible serait de tenter une classification des paires, classification similaire à celle effectuée pour les probabilités d'alignement de modèle MDI2B (3.3.4). Une telle classification pourrait potentiellement permettre au modèle d'exprimer la propension qu'a une paire à se répéter à travers un certain nombre

de classes de paires sans toutefois souffrir de sous-représentation des données. Chaque classe représenterait un niveau de propension à se répéter.

Finalement, notre modèle de traduction cache n'utilise que la partie du texte source qui a déjà été traduite. Il pourrait être intéressant d'utiliser le texte source complet en tout moment de la traduction et ce puisque, dans le contexte de TransType, ce texte source est toujours disponible en entier.

Chapitre 6

Évaluation des modèles dans le cadre de TransType

Nous présentons dans ce chapitre les résultats expérimentaux des modèles de traduction étudiés dans les deux chapitres précédents dans le cadre d'un test de simulation de l'application TransType.

Nous avons argumenté dans ce mémoire que l'application TransType permettait de prendre un avantage complet des modèles cache et ce dans un contexte pratique. Les résultats de perplexité obtenus dans les précédents chapitres démontrent la puissance de tels modèles. Les résultats présentés dans ce chapitre tenteront de démontrer cette puissance dans un contexte pratique.

Ces tests s'effectuent en simulant un traducteur à l'oeuvre avec le logiciel TransType. Ce traducteur "virtuel" se voit offrir des propositions de mots par le moteur de traduction de la même façon qu'un véritable traducteur. Il accepte, rejette et corrige les propositions de façon réaliste en utilisant une fonction de bénéfice que lui apporte chacune de ces propositions.

Nous effectuons ces tests seulement avec les trois premiers fichiers de la tranche

Taille	BI	%	1+2+3	%
base	MDI2B[ref=I3G(+CACHE)]=27.435			
2000	27.784	+1.27%	27.719	+1.04%
5000	27.837	+1.46%	27.821	+1.41%

TAB. 6.1 – Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de langue. Tests effectués sur la tranche **test** du corpus Hansard. Les colonnes représentent les types de cache testés alors que les rangées correspondent aux tailles de cache testées.

test du corpus Hansard et du corpus *sniper*. Seulement quelques fichiers ont été utilisés puisqu'une telle simulation est extrêmement coûteuse en temps, car le moteur de traduction effectue des propositions à chaque caractère entré par le traducteur. Le temps de simulation est donc très près du temps qu'il en prendra au traducteur pour traduire les mêmes documents avec TransType.

Le résultat de ces simulations est un pourcentage de frappes au clavier économisées qu'entraîne l'utilisation de TransType par rapport au nombre de frappes totales nécessaires pour entrer l'entièreté du texte cible dans un logiciel de traitement de texte.

6.1 Résultats du modèle MDI2B avec modèle de langue cache dans la distribution de référence

Le tableau 6.1 présente les résultats obtenus avec le modèle MDI2B avec composante cache au niveau du modèle de langue sur les trois premiers fichiers de la tranche **test** du corpus Hansard, alors que le tableau 6.2 présente ceux obtenus avec le même modèle sur les trois premiers fichiers du corpus *sniper*.

On remarque que le modèle MDI2B avec composante cache intégrée dans la distribution de référence améliore la quantité de frappes économisées qu'entraîne l'utilisation de TransType. On note une amélioration de 1.46% pour la cache bigramme de taille

Taille	**BI**	**%**	**1+2+3**	**%**
base	MDI2B[ref=I3G(+CACHE)]=9.686			
2000	11.404	+15.1%	11.294	+14.2%
5000	11.498	+15.8%	11.623	+16.7%

TAB. 6.2 – Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de langue. Tests effectués sur le corpus *sniper*. Les colonnes représentent les types de cache testés alors que les rangées correspondent aux tailles de cache testées.

	0.3	**%**
base	MDI2B=27.4358	
1000	27.557	+0.44%
2000	27.531	+0.35%
5000	27.485	+0.18%
10000	27.468	+0.12%

TAB. 6.3 – Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de traduction. Tests effectués sur la tranche **test** du corpus Hansard. Un seul poids est testé, soit 0.3. Les rangées représentent les tailles des caches testées.

5000, alors que la cache unigramme, bigramme et trigramme interpolée entraîne une amélioration de 1.41%. Ces résultats confirment de façon pratique l'amélioration obtenue par ces modèles au niveau de la perplexité.

Au niveau des résultats obtenus avec le corpus *sniper*, les résultats sont d'autant plus intéressants. On obtient une hausse du pourcentage de frappes économisées maximale de 15.8% dans le cas du modèle cache bigramme et de 16.7% dans le cas du modèle cache unigramme, bigramme et trigramme interpolé. Ces résultats montrent que dans le cas d'un document analysé bien différent des documents contenus dans le corpus d'entraînement, la quantité de frappes économisées peut être significative.

	0.3	**%**
base	MDI2B=9.686	
1000	9.90	+2.17%
2000	10.02	+3.48%
5000	9.98	+3.07%
10000	9.96	+2.80%

TAB. 6.4 – Pourcentage de frappes économisées par l'utilisation du modèle MDI2B avec composante cache sur le modèle de traduction. Tests effectués sur le corpus *sniper*. Un seul poids est testé, soit 0.3. Les rangées représentent les tailles des caches testées.

6.2 Résultats du modèle MDI2BCache

Tout comme les résultats en matière de perplexité du modèle, l'amélioration du pourcentage de frappes économisées est moindre pour notre modèle MDI2BCache que pour le modèle de langue cache. Avec un poids seuil de 0.3 et une cache de taille 1000, on obtient une amélioration de 0.44% au niveau de l'économie de frappes engendrée par l'utilisation de TransType dans le cas du test effectué sur la tranche **test** du corpus Hansard (tableau 6.3). Pour ce qui est du corpus *sniper* (tableau 6.4), l'utilisation de TransType avec un modèle MDI2BCache entraîne une augmentation de l'économie de frappes de 3.48% avec une cache de taille 2000 et un poids seuil de 0.3. Il est à noter que l'évaluation de TransType avec un modèle MDI2BCache sur le corpus *sniper* est le seul test où les hausses de l'économie de frappes sont supérieures aux baisses de perplexité. Ceci démontre qu'un tel modèle entraîne de meilleures propositions à des endroits où TransType faisait des erreurs avec le modèle de base. Ceci est grandement désiré dans un contexte pratique par rapport à un modèle qui ne fait qu'augmenter les probabilités des mots pour lesquels TransType faisait déjà la bonne proposition.

6.3 Discussion

De façon générale, ces résultats démontrent les possibilités qu'offrent l'adaptation de modèles de traduction. Dans le cas de l'adaptation unilingue, les résultats sont concluants, c'est-à-dire que les améliorations théoriques qu'offrent l'adaptation se concrétisent au niveau pratique. On améliore d'environ 1.5% le pourcentage de frappes économisées dans un contexte de texte "connu" (intimement lié au corpus d'entraînement) et de plus de 18% dans le contexte où le document est très peu lié au corpus d'entraînement.

Les améliorations en pourcentage sont légèrement inférieures à celles obtenues dans les tests de perplexité. Ceci s'explique facilement. Dans le cas de la perplexité, on peut améliorer chacune des prédictions puisqu'on tente de hausser la probabilité de tous les mots du document cible. Dans le cas de l'évaluation pratique, on ne peut qu'améliorer le modèle dans le cas où il y avait erreur.

Dans le cas de l'évaluation en terme d'économie de frappes, il faut donc que le modèle soit à ce point amélioré pour faire en sorte de proposer des mots que le modèle de base ne proposait pas. Ce sont donc seulement les mots pour lesquels les prédictions du modèle changent et pour lesquels le modèle de base faisait erreur qui amène une amélioration pratique du modèle.

Dans le cas du modèle MDI2BCache, les hausses du pourcentage de frappes économisées sont à peu près proportionnelles aux baisses de perplexité. On obtient une hausse maximale de 0.44%. De la même façon qu'avec les baisses de perplexité, ces hausses nous démontrent bel et bien qu'une information est présente au niveau bilingue ce qui nous permet d'améliorer les prédictions d'un modèle de traduction. Des hausses plus importantes pourraient être obtenues avec un modèle adaptatif amélioré.

Il faut mentionner que l'évaluation faite par une simulation d'un traducteur n'est pas idéale. Premièrement, au niveau technique, cette évaluation n'a pas toute la robustesse désirée. De plus, elle ne capte pas l'aspect psychologique de l'adaptation de tels modèles

de traduction.

Tel que mentionné, la simulation nous permettant d'obtenir des résultats d'économie de frappes n'utilise qu'une tranche de trois fichiers de nos corpus. On ne peut utiliser plus de fichiers étant donné les coûts en temps d'une telle évaluation. Une telle évaluation est très longue, puisqu'un système comme TransType fait des propositions à l'utilisateur à chaque caractère entré par ce dernier. Par exemple, pour une évaluation d'une tranche de 1 million de mots d'un corpus, on demande au système plus de 6 millions de prédictions, ce qui est énorme.

Or, avec seulement trois fichiers, il est difficile d'admettre que de tels résultats représentent exactement l'amélioration qui a été apportée aux modèles par l'ajout des composantes cache. De plus, il faut noter qu'avec des composantes cache de taille 5000 au niveau du modèle de langue et de 1000 au niveau du modèle de traduction, il y a une certaine quantité de prédictions au début d'un document qui ne jouissent pas de toute la puissance du modèle étant donné que la composante cache n'est pas encore remplie à pleine capacité. Avec un très petit corpus de test, ceci peut influencer le résultat.

De plus, les modèles adaptatifs ont une valeur psychologique qui ne se calcule pas dans de telles simulations. Le simple fait que l'utilisateur sente que l'application s'adapte à son travail est une valeur ajoutée non négligeable. Ainsi, il peut être irritant pour l'utilisateur que le modèle ne propose pas un mot qui suit une tournure de phrase qui vient tout juste d'être utilisée ou qu'après l'utilisation d'un même mot dans un même contexte à plusieurs reprises, le modèle fasse encore et toujours la même proposition erronée. Donc, avec les modèles développés, nous savons que le modèle s'adapte à la tâche de l'utilisateur.

Nous considérons donc les résultats obtenus comme montrant la validité de l'approche d'adaptation des modèles de traduction tant au niveau du modèle de langue que du modèle de traduction.

Si on voulait démontrer l'utilité réelle de tels modèles, il faudrait alors les incorporer dans une session de test du logiciel par de véritables traducteurs. Les performances de ces modèles pourraient alors être mésurées de deux façons. Premièrement, par la vitesse de traduction lors de l'utilisation de TransType avec et sans les composantes cache. Deuxièmement, avec une appréciation qualitative de l'expérience liée à l'utilisation de TransType. Cette dernière façon permettrait de juger de l'aspect psychologique qu'entraîne l'utilisation de modèle adaptatifs.

Chapitre 7

Conclusion

Nous avons débuté ce mémoire par la présentation du projet TransType. Le projet TransType consiste à développer un logiciel de traduction assistée par ordinateur. Le logiciel comprend un traitement de texte et un moteur de traduction. À mesure que le traducteur produit sa traduction, le moteur de traduction propose des mots et des phrases qui ont une probabilité élevée d'être entrés immédiatement par le traducteur. L'ordinateur et l'utilisateur travaillent donc de pair.

Nous avons ensuite présenté la tâche de traduction telle que modélisée dans le domaine du traitement statistique des langues naturelles, soit l'approche canal bruité. Nous avons ensuite défini les modèles de langue et les modèles de traduction.

Ceci nous a amené à présenter l'adaptation des modèles de langue et, en particulier, les modèles de langue cache. L'adaptation des modèles de langue consiste à utiliser la partie déjà analysée d'un texte pour modifier le modèle et ainsi améliorer ses prédictions futures. Les modèles de langue cache constituent une des techniques d'adaptation de modèle de langue. L'adaptation de modèles de langue par les modèles cache est un domaine du traitement des langues naturelles qui a fait l'objet de beaucoup de recherche [18, 5, 20, 11, 13] sans jamais obtenir des résutats pratiques intéressants. Ce manque d'intérêt pratique s'est expliqué par le fait qu'on assumait une correction des mots

avant qu'ils entrent dans la cache, ce qui n'est pas le cas dans bien des domaines du traitement des langues naturelles tels que la reconnaissance de la parole et la traduction automatique. Par contre, cette correction des mots est faite dans TransType puisque l'ordinateur et le traducteur travaillent ensemble.

La nature interactive du programme TransType nous a donc permis de croire que les modèles présents dans le moteur de traduction de TransType étaient d'excellents candidats à l'adaptation, en particulier le modèle de langue par une adaptation de type cache.

Nous avons donc utilisé nos propres implémentations de modèles de langue cache pour les incorporer aux moteurs de traduction de TransType. Nous avons testé nos modèles sur deux corpus de test, soit une tranche du corpus Hansard et le corpus *sniper*, un corpus très différent du corpus Hansard. Les résultats sont intéressants et démontrent la puissance de l'utilisation de modèle de langue cache dans un contexte de traduction. Ces modèles de langue entraînent une baisse de perplexité de plus de 5% sur la tranche **test** du Hansard et de plus de 50% sur le corpus *sniper*. Ce dernier résultat est particulièrement intéressant en vue d'une utilisation de TransType par des traducteurs ayant besoin de traduire des documents dans des domaines particuliers.

Nous avons ensuite tenté d'étendre l'idée sous-jacente aux modèles de langue cache au cas bilingue. Pour ce faire, nous avons développé un modèle de traduction cache, le modèle MDI2BCache. Celui-ci est un modèle MDI2B auquel nous avons ajouté une composante cache sur les paires de mots. Les résultats de perplexité de ce modèle sur les mêmes corpus de test ne sont pas aussi intéressants que ceux obtenus avec le modèle de langue cache. Ils entraînent une amélioration de 0.5% sur le corpus Hansard et 2.3% sur le corpus *sniper*. Plusieurs facteurs peuvent expliquer ces résultats. Premièrement, l'aspect alignement de mots pose une grande difficulté à ce modèle. De plus, ce modèle cache ne tient compte ni des mots inconnus ni des paires inconnues, aspect qu'il serait

interessant d'ajouter à un tel modèle.

Finalement, nous avons évalué tous ces modèles dans un contexte pratique propre à TransType, c'est-à-dire par la simulation d'un utilisateur typique de TransType. Une telle simulation nous a permis d'obtenir des résultats représentant le pourcentage de frappes au clavier économisées qu'entraîne l'utilisation de tels modèles. Ainsi, les modèles de langue cache permettent d'économiser environ 1.5% plus de frappes sur le corpus Hansard et 18% sur le corpus *sniper*. Le modèle de traduction cache MDI2BCache permet lui d'économiser 0.4% plus de frappes sur le corpus Hansard et 3.5% sur le corpus *sniper*.

Après toutes ces expériences sur différents modèles de langue et de traduction adaptatifs, nous pouvons conclure que l'implémentation des modèles de langue cache utilisées dans les expériences présentées dans ce mémoire pourrait et devrait être utilisée dans l'utilisation de TransType, puisqu'ils ont démontré qu'ils entraînaient sans contredit une baisse de perplexité du système et une hausse de l'économie de frappes au clavier.

Nous concluons aussi que l'adaptation bilingue, au niveau du modèle de traduction, est possible. Les résultats démontrent que de l'information se trouve dans les paires de mots rencontrées tout au long d'une traduction. Par contre, avant d'être utilisable dans un cadre pratique, l'adaptation de modèles de traduction devrait être implémentée par des modèles permettant d'aller chercher d'avantage l'information se trouvant dans ces paires tout en faisant le traitement des mots inconnus.

Bibliographie

[1] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A Maximum Entropy approach to Natural Language Processing. *Computational Linguistics*, 22(1) :39–71, 1996.

[2] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, Robert L. Mercer, and Paul Roossin. A statistical approach to language translation. In *Proceedings of the International Conference on Computational Linguistics (COLING)*, pages 71–76, Budapest, Hungary, August 1988.

[3] Peter F. Brown, Stephen A. Della Pietra, Vincent Della J. Pietra, and Robert L. Mercer. The mathematics of Machine Translation : Parameter estimation. *Computational Linguistics*, 19(2) :263–312, June 1993.

[4] Kenneth W. Church and William A. Gale. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 1993.

[5] Philip R. Clarkson and Anthony J. Robinson. Language model adaptation using mixtures and an exponentially decaying cache. In *IEEE Int. Conference on Acoustics, Speech, and Signal Processing*, pages 799–802, Munich, 1997.

[6] George Foster. Incorporating position information into a maximum entropy/minimum divergence translation model. In *Conference on Natural Language Learning (CoNLL)*, pages 37–42, Lisbon, Portugal, 2000.

[7] George Foster. A Maximum Entropy / Minimum Divergence translation model. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL)*, Hong Kong, October 2000.

[8] George Foster. *Text Prediction For Translators.* PhD thesis, Université de Montréal, May 2002.

[9] George Foster, Pierre Isabelle, and Pierre Plamondon. Target-text Mediated Interactive Machine Translation. *Machine Translation*, 12 :175–194, 1997.

[10] George Foster, Philippe Langlais, and Guy Lapalme. User-friendly text prediction for translators. In *2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Philadelphia, July 2002.

[11] Joshua Goodman. A bit of progress in language modeling. *Computer Speech and Language*, 15(4) :403–434, October 2001.

[12] Frederick Jelinek and Robert L. Mercer. Interpolated estimation of Markov source parameters from sparse data. In E. S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice*. North-Holland, Amsterdam, 1980.

[13] Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 12(6) :570–583, June 1990.

[14] Philippe Langlais. Notes de cours, ift6010 : Traitement statistique des langues naturelles., 2003.

[15] Philippe Langlais, George Foster, and Guy Lapalme. Unit completion for a computer-aided translation typing system. In *Applied Natural Language Processing (ANLP)*, pages 135–141, Seattle, Washington, May 2000.

[16] Philippe Langlais, Guy Lapalme, and Marie Loranger. TransType : Development-evaluation cycles to boost translator's productivity. *accepté à Machine Translation (Special Issue on Embedded Machine Translation Systems)*, page 24 pages, feb 2003.

[17] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[18] Sven C. Martin, Jorg Liermann, and Hermann Ney. Adaptative topic-dependent language modelling using word-based varigrams. In *Fifth European Conference on Speech Communication and Technology(Eurospeech)*, volume 3, pages 1447–1450, Rhodes, September 1997.

[19] Rada Mihalcea and Ted Pedersen. An evaluation exercise for word alignment. In Rada Mihalcea and Ted Pedersen, editors, *HLT-NAACL 2003 Workshop : Building and Using Parallel Texts : Data Driven Machine Translation and Beyond*, pages 1–10, Edmonton, Alberta, Canada, May 31 2003. Association for Computational Linguistics.

[20] Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modelling. *Computer Speech and Language*, 10 :187–228, 1996.

[21] Michel Simard, George Foster, and Pierre Isabelle. Using cognates to align sentences in bilingual corpora. In *Proceedings of the 4th Conference on Theoretical*

and Methodological Issues in Machine Translation (TMI), pages 67–81, Montréal, Québec, 1992.

[22] Dominique Vaufreydaz. *Modélisation statistique du langage à partir d'Internet pour la reconnaissance automatique de la parole continue.* Ph.d. thesis in computer sciences, University Joseph Fourier, Grenoble (France), January 2002.

[23] Jean Veronis and Philippe Langlais. Evaluation of parallel text alignment systems : The ARCADE project. In *Parallel Text Processing.* Kluwer Academic Publishers, Text, Speech and Language Technology Series, Jean Véronis editor, 2000.

Annexe A

Les corpora utilisés

A.1 Le Hansard

Le Hansard est le Journal des débats à la Chambre des communes du Parlement canadien. Le corpus utilisé dans ce mémoire contient la totalité du Hansard, pour la période allant de avril 1986 à février 2003. Il s'agit de traductions de l'anglais au français et vice versa. Le tout représente environ 235 millions de mots.

On utilise le corpus du Hansard canadien étant donné qu'il donne accès à une quantité énormes de documents traduits en deux langues. Outre des textes légaux ou journalistiques, il est très rare d'obtenir tant de textes traduits.

Certaines tranches de ce corpus ont donc été utilisées pour l'entraînement, alors que d'autres ont été utilisées pour des tests. Voici les tranches utilisées :

Le tranche d'entraînement **train.main** contient donc 1.6 millions de phrases alignées pour un total d'environ 30 millions de mots français et anglais. Les tranches **train.ho1** et **train.ho2** sont des tranches utilisées principalement pour l'entraînement de coefficients de mélanges de modèles de la forme $\lambda_1 \times p(x) + \lambda_2 \times q(x)$. Finalement, la tranche **test** est utilisée pour obtenir les résultats de perplexité. Ces trois dernières tranches comportent environ cinquante mille phrases alignées et un million de mots tant français qu'anglais.

tranche	fichiers	phrases	mots français	mots anglais
train.main	922	1.6M	31.8M	29.5M
train.ho1	30	55K	1M	1M
train.ho2	30	59K	1.2M	1M
test	30	54K	1M	1.1M

TAB. A.1 – Les différentes tranches du corpus Hansard utilisées.

tranche	fichiers	phrases	mots français	mots anglais
test	5	4K	61K	50K

TAB. A.2 – La tranche du corpus *sniper* utilisée.

Ces tranches sont toutes mutuellement exclusives.

A.2 *Sniper*

Le corpus *sniper* est un tout petit corpus constitué d'un manuel permettant l'apprentissage du métier de "tireur d'élite" (*sniper*). Le manuel est en deux langues, soit le français et l'anglais.

L'utilité d'un tel corpus se situe dans le fait qu'il est constitué d'un langage et d'un vocabulaire extrêmement différents que ceux que l'on retrouve dans le corpus *Hansard*.

Nous l'avons donc utilisé uniquement pour des tests. Aucun de nos modèles ne sont entraînés avec ce corpus. Ces tests nous permettent de quantifier l'utilité d'un modèle *adaptatif* lorsque l'on tente de traduire un texte différent des textes qui ont servi à l'entraînement des modèles.

Annexe B

Soutien mathématique

B.1 La perplexité

La théorie de l'information définit l'entropie d'une source d'information comme étant la quantité moyenne d'information véhiculée par un événement de la source. Cette quantité d'information est exprimée en *bits*. Intuitivement, l'entropie représente le nombre moyen de bits nécessaires pour représenter optimalement de façon binaire les événements d'une source d'information qui suit une certaine distribution de probabilités.

Pour une source S modélisée par une distribution p prenant des valeurs parmi un ensemble de k valeurs $[1, k]$, l'entropie est défini par cette espérance :

$$H(S) = -\sum_{i \in [1,k]} p_i \, log_2 \, p_i \tag{B.1}$$

On fait donc la moyenne des $-log_2 p_i$ qui représentent la quantité de bits nécessaire pour représenter le i^e symbole de la source.

En théorie langagière, on considère la source comme une variable aléatoire W. Ainsi, :

$$H(W) = -\sum_{W} p(W) \, log_2 \, p(W) \tag{B.2}$$

où chaque instance de W est une certaine séquence de mots w_1^n de longueur quelconque. L'équation B.2 peut être reformulée de la façon suivante :

$$H(W) = -\sum_{w_1^n} p(W = w_1^n)\, log_2\, p(w_1^n) \tag{B.3}$$

On exprime ensuite l'entropie par mot, définie de la façon suivante :

$$H_m(W) = -\frac{1}{n}\sum_{w_1^n} p(W = w_1^n)\, log_2\, p(w_1^n) \tag{B.4}$$

Ce qui nous mène à exprimer l'entropie d'un language :

$$H(L) = -\lim_{n\to\infty}\frac{1}{n}\sum_{w_1^n} p(W = w_1^n)\, log_2\, p(w_1^n) \tag{B.5}$$

Évidemment, on ne peut calculer une telle expression car il existe une infinité de w_1^n possibles. On considère alors la source comme ergodique et on estime l'entropie du language avec une seule instance de W, instance idéalement la plus longue possible. On définit alors le *logprob* qui estime l'entropie de la façon suivante :

$$LP(W) = -\frac{1}{n}\, log_2\, \hat{p}(w_1 ... w_n) \tag{B.6}$$

Ainsi, l'entropie du language est une borne inférieure au *logprob* que l'on exprime par rapport à la distribution estimée ($LP > H$).

La perplexité quant à elle est définie de la façon suivante :

$$PP(W) = 2^{H(W)} \tag{B.7}$$

De la même façon que l'on estime l'entropie d'un language par le *logprob*, on estime la perplexité du language par le *logprob*, ainsi :

$$\hat{PP}(W) = 2^{LP(W)} \tag{B.8}$$

De façon intuitive, la perplexité représente le nombre moyen de choix que fait un

modèle lors de la prédiction d'un mot. Ainsi, un modèle ayant une perplexité de 22 verrait le modèle faire en moyenne 22 choix de mots pour prédire le bon mot. [22, 14]