Apprendre par imitation : applications à quelques problèmes d'apprentissage structuré en traitement des langues

Elena Knyazeva

université PARIS-SACLAY

NNT : 2018SACLS134

Thèse de doctorat

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

Apprendre par imitation : application à quelques problèmes d'apprentissage structuré en traitement des langues

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

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

Thèse présentée et soutenue à Orsay, le 25/05/2018, par

Mme Elena Knyazeva

Composition du Jury :

Mme Sophie Rosset Directrice de recherche, CNRS (- LIMSI)	Présidente
M. Pascal Denis Chargé de recherche, INRIA Lille	Rapporteur
M. Thierry Artières Professeur, Université de Marseille (- LIS)	Rapporteur
M. Artem Sokolov Chercheur Senior, Amazon Berlin	Examinateur
M. Matthieu Geist Professeur, Université de Loraine (- LIEC)	Examinateur
M. François Yvon PR1, CNRS (- LIMSI)	Directeur de thèse
M. Guillaume Wisniewski Maître de conférences, Université Paris-Sud (-LIMSI)	Co-directeur de thèse

Table des matières

Remerciements

Tout d'abord, je tiens à remercier mes encadrants. Je remercie François Yvon pour toute l'attention qu'il a donné à notre travail malgré ses nombreuses obligations au laboratoire, et pour m'avoir guidé avec patience au travers de toutes les difficultés. Je remercie Guillaume Wisniewski pour m'avoir confié ce sujet passionnant et pour m'avoir transmis tout l'enthousiasme qu'il pouvait pour cette longue route.

Merci à mes rapporteurs Pascal Denis et Thierry Artières d'avoir étudié avec attention mon travail ainsi que pour leur remarques qui m'ont permit de l'améliorer. Merci également à mes examinateurs Artem Sokolov et Matthieu Geist pour les discussions passionnantes pendant ma soutenance. Merci à Sophie Rosset d'avoir accepté de présider mon jury et pour l'intérêt qu'elle a porté à mon travail.

Je remercie tous mes collègues qui ont partagé mon quotidien au LIMSI, en particulier : Li Gong, Benjamin Marie, Nicolas Pécheux, Quoc Khanh Do, Julia Ive, Arseny Gorin, Franck Burlot, Rachel Bawden, Matthieu Labeau, Lauriane Aufrant, Pierre Godard, Yong Xu, William Hartmann, Thiago Fraga de Silva, Natalia Segal et Cyril Grouin. Nous avons beaucoup de souvenirs à partager, j'espère que l'on aura des occasions de se revoir.

Je remercie également mes parents, mes grand-parents, mes cousins et cousines, oncles et tantes, ainsi que mon petit filleul Ivan qui est né pendant ce travail. Je suis heureuse de faire partie d'une telle famille grande et soudée (malgré les distances qui nous séparent tous). Merci également à mes beaux-parents, ma belle sœur et mon beau-frère et à mes deux petites nièces, qui font que je me sens toujours un peu plus chez moi en France. J'ai une pensée toute particulière pour mes deux grand-pères et mes deux belles grand-mères qui nous ont quitté pendant cette thèse. Je remercie aussi père Andrei pour son aide inestimable et ses conseils pleins de sagesse.

Un grand merci à mes amis russes Dimitri, Marina, Lena, Katia, Ania, Alyona, Artem, Olia, et particulièrement à mes meilleures amies Tania et Natacha, pour leur existence et la fidélité de leur amitié. Vous êtes des amis incroyables.

Je remercie ma petite fille Anastasia d'avoir rendu la dernière année de mon travail pleine de bonheur et de joie, ainsi que d'avoir bien dormi pendant les premiers mois pour me laisser un peu de temps pour finir mon manuscrit.

Enfin, je ne remercierai jamais assez mon mari Thomas pour tout le soutien qu'il m'a accordé pendant ces années pas simples. C'est grâce à ma présence au LIMSI que j'ai pu le rencontrer, et c'est à lui que je dédie ce travail.

Notations

X	l'ensemble des entrées
Y	l'ensemble des sorties
$G(x, y)$	l'espace de recherche associée à l'entrée x et la référence y
S	l'ensemble fini des états possibles dans un espace de recherche
A	l'ensemble fini des actions possibles dans un espace de recherche
S_f	l'ensemble des états finaux
s_i	l'état initial
$r(s, a)$	la récompense associée à l'action a dans l'état s
A_s	l'ensemble des actions accessibles dans l'état s
$s \circ a$	l'état d'arrivé après avoir effectué l'action a dans l'état s *déterministe dans notre configuration*
T	la longueur minimale telle que chaque chemin de cette longueur ou plus est complet
$\hat{y}_\alpha$	la sortie structurée correspondante au chemin α
$R(\alpha)$	la récompense cumulée du chemin α *moyenne, si la politique est stochastique*
$L(\hat{y})$	la perte associée à la sortie structurée $\hat{y}$
$L(\pi)$	la perte associée à la sortie structurée produite par la politique π *moyenne, si la politique est stochastique*
$\pi(s)$	l'action choisie par la politique π dans l'état s *stochastique, si la politique est stochastique*
$s \circ \pi$	l'état d'arrivé après avoir effectué l'action choisie par la politique π dans l'état s *stochastique, si la politique est stochastique*

$\widetilde{s} \circ \pi$	de même pour la distribution $\widetilde{s}$ *stochastique*
$s \circ \pi^k$	l'état d'arrivé après avoir effectué k actions choisies par la politique π dans l'état s *stochastique, si la politique est stochastique*
$\widetilde{s} \circ \pi^k$	de même pour la distribution $\widetilde{s}$ *stochastique*
$s \circ \pi^*$	l'état d'arrivé après avoir effectué un nombre aléatoire (entre 0 et T) d'actions choisies par la politique π dans l'état s *stochastique*
$\widetilde{s} \circ \pi^*$	de même pour la distribution $\widetilde{s}$ *stochastique*
$s \xrightarrow{\pi,\,\mathrm{k}} \cdot$	le chemin de k actions et $k+1$ états construit à l'aide de la politique π à partir de l'état s *stochastique, si la politique est stochastique*
$\widetilde{s} \xrightarrow{\pi,\,\mathrm{k}} \cdot$	de même pour la distribution $\widetilde{s}$ *stochastique*
$s \xrightarrow{\pi,\,\cdot} \mathcal{S}_f$	le chemin de construit à l'aide de la politique π à partir de l'état s jusqu'à un état final *stochastique, si la politique est stochastique*
$\widetilde{s} \xrightarrow{\pi,\,\cdot} \mathcal{S}_f$	de même pour la distribution $\widetilde{s}$ *stochastique*
$V(s;\pi)$	$R(s \xrightarrow{\pi,\,\cdot} \mathcal{S}_f)$ fonction de valeur de l'état s vis-à-vis de la politique π *moyenne, si la politique est stochastique*
$V(\widetilde{s};\pi)$	$R(s \xrightarrow{\pi,\,\cdot} \mathcal{S}_f)$ de même pour la distribution $\widetilde{s}$ *moyenne*
$Q(s;a,\pi)$	$R(s \xrightarrow{\cdot} s \oplus a \xrightarrow{\pi,\,\cdot} \mathcal{S}_f)$ fonction de valeur associée à l'état s et l'action a vis-à-vis de la politique π *moyenne, si la politique est stochastique*
$Q(\widetilde{s};\pi_1,\pi_2)$	$R(\widetilde{s} \xrightarrow{\pi_1,1} \cdot \xrightarrow{\pi_2,\,\cdot} \mathcal{S}_f)$ fonction de valeur associée à l'état s et l'action choisie par la politique π_1 vis-à-vis de la politique π_2 *moyenne*
$l(s,a;\pi)$	perte de potentiel associée à la prise de l'action a dans l'état s vis-à-vis de la politique π *moyenne si* π *est stochastique*
$l(\widetilde{s},\pi_1;\pi_2)$	perte de potentiel associée à la prise de l'action choisie par la politique π_1 dans l'état distribué comme $\widetilde{s}$ vis-à-vis de la politique π_2 *moyenne*

$\text{ExpProb}(\pi)$	problème d'apprentissage de l'expert engendré par la politique π
$\text{ExpProb}^t(\pi)$	problème d'apprentissage de l'expert à la t-ème action engendré par la politique π
$\text{TutProb}(\pi)$	problème d'apprentissage de tuteur engendré par la politique π
$\text{TutProb}^t(\pi)$	problème d'apprentissage de tuteur à la t-ème action engendré par la politique π

Introduction

L'apprentissage automatique a permis de développer de nombreuses applications du traitement automatique de l'information, et a permis leur diffusion à un large public. Par exemple, les techniques de traitement d'image permettent aujourd'hui de reconnaitre des objets dans des scènes animées complexes, d'identifier des personnes etc. L'apprentissage est également au cœur des applications de recherche d'information multimédia, de traitement automatique des langues naturelles, etc. On peut considérer que pour l'essentiel, les progrès récents de ces applications reposent sur des méthodes d'apprentissage.

Le travail présenté dans ce manuscrit porte sur l'apprentissage pour le traitement automatique des langues naturelles dont les applications sont très variées : la traduction automatique, la synthèse vocale, les assistants personnels tels que Siri ou Cortana...

Ces applications ont pour point commun que les objets manipulés ont en général une composition complexe. De nombreuses applications en traitement automatique de la langue donnent lieu à des problèmes dits d'*apprentissage structuré* : c'est-à-dire des problèmes où les objets considérés sont composés d'un ensemble d'objets plus simples mais interdépendants et où l'apprentissage et l'inférence sont réalisés en exploitant la structure formée par ces dépendances. Ces structures peuvent être plus ou moins complexes, d'une simple séquence à des graphes arbitraires en passant par les arbres. L'inférence et l'apprentissage exacts sur ces structures sont généralement problématiques : les nombreuses dépendances impliquent que dans beaucoup de cas ces problèmes sont NP-difficiles. Deux grandes familles d'approches proposant des solutions approximatives pour réduire cette complexité existent dans la littérature :

— La première approche : l'« approximation de la structure » consiste à ne retenir qu'une partie des dépendances composant la structure afin de rendre l'apprentissage et l'inférence plus simples. Il est par exemple possible de se limiter aux dépendances locales entre éléments voisins respectant la propriété de Markov. Cette hypothèse simplificatrice

est à la base de nombreux modèles (modèle de Markov caché (*Hidden Markov Model* ou HMM) et champ aléatoire conditionnel (*Conditional Random Field* ou CRF) par exemple) et permet de réaliser aussi bien l'apprentissage que l'inférence de manière exacte. Toutefois, la perte d'information est parfois trop importante comme dans le cas de la traduction automatique ou de l'analyse syntaxique où deux mots fortement dépendants peuvent être très éloignés dans la phrase à analyser ou à traduire.

— La deuxième approche : l'« approximation de la recherche », consiste quant-à-elle à approximer les procédures d'apprentissage et d'inférence afin de pouvoir conserver toutes (ou presque toutes) les dépendances présentes dans la structure. Ces nombreuses dépendances peuvent, par exemple, amener la fonction objectif à être non-convexe et impliquent aussi généralement qu'une *inférence approchée* doit être utilisée afin de trouver une *bonne* solution en un temps raisonnable. Cette stratégie n'offre pas de garanties d'optimalité de la solution obtenue, mais les méthodes connues dans la littérature sous le nom d'« apprendre à chercher » [Daumé et al., 2009] montrent qu'il est possible, en adaptant la procédure d'apprentissage, de contrôler la qualité de cette recherche.

La dénomination « apprendre à chercher » [1] signifie que le modèle est appris de manière à construire la solution pas-à-pas de façon heuristique. Il est toutefois possible d'apprendre à bien chercher et donc de minimiser l'impact de cette approximation. L'étude de ces méthodes est actuellement un sujet très actif notamment avec l'arrivée des réseaux neuronaux récurent (RNN) qui sont dans un cadre non-markovien et donc concernés par ces problèmes de recherche inexacte.

Un premier objectif de ce travail est de donner une présentation unifiée de cette famille de méthodes, qui est encore relativement mal documentée, en identifiant précisément les fondements théoriques qui justifient les deux grands types d'algorithmes : la famille AGGRAVATE [2] [Ross and Bagnell, 2014] et la famille SEARN [Daumé et al., 2009]. Nous nous sommes ensuite intéressés à l'étude des questions suivantes :

1. Comment résoudre le compromis entre l'approximation de la structure et celle de la recherche sur des tâches concrètes ? Est-ce que ce

1. Notons que la signification de ce terme reste encore vague dans la littérature ; certaines méthodes d'apprentissage par imitation ou par renforcement sont parfois présentées sous ce nom.

2. Cette famille inclut notamment l'algorithme DAGGER [Ross et al., 2011], probablement le mieux connu dans la communauté de traitement automatique des langues.

compromis est dépendant de la tâche et de sa structure et si oui, quelles sont les caractéristiques favorisant une approche ou l'autre ? Afin d'apporter des éléments de réponses à ces questions, nous avons considéré plusieurs tâches structurées du traitement automatique des langues de nature différente, et comparé des méthodes réalisant des compromis différents sur ces tâches.

2. Est-il possible d'améliorer les performances des approches utilisant une recherche approchée en considérant différentes manières de construire la structure recherchée ? Nous avons étudié cette question en observant si l'attribution des étiquettes par le système dans un ordre libre permet d'obtenir de meilleures performances que l'ordre gauche-droite classiquement utilisé.
3. Comment adapter ces techniques à des tâches de plus grande complexité ? Nous nous sommes ici concentrés sur l'adaptation de ces méthodes au cas de la traduction automatique qui présente notamment de nombreuses dépendances à longue distance et où les hypothèses sont de taille variable.

Ce manuscrit est organisé de manière suivante. Dans le premier chapitre, nous présentons un état de l'art de l'apprentissage supervisé et en particulier les bases nécessaires à la présentation de la théorie des méthodes « apprendre à chercher ». Avant de présenter ces dernières, nous faisons un détour au chapitre 2 afin de présenter les éléments du traitement automatique des langues naturelles et les différentes tâches de ce domaine qui feront l'objet des expérimentations.

Le chapitre 3 est consacré aux méthodes de type « apprendre à chercher » : les principaux algorithmes de l'état de l'art y sont d'abord présentés de manière unifiée, ce qui permet d'identifier les fondements théoriques de ces approches et de pointer les différences entre les familles SEARN et AGGRAVATE. Nous y présentons également une première contribution sous la forme d'une variante de SEARN disposant de meilleurs garanties théoriques.

Le chapitre 4 est consacré à l'étude expérimentale de différentes tâches d'étiquetage de séquences. Cette étude porte principalement sur une comparaison entre l'algorithme SEARN comme représentant de l'approche « apprendre à chercher » et les modèles CRF comme représentants des approches approximant la structure. Cette étude nous permet d'apporter des éléments de réponses aux deux premières questions présentées.

Dans le chapitre 5, nous présentons une adaptation de SEARN pour la tâche de traduction automatique. Cette tâche étant très complexe, ce

chapitre a pour principal objectif de détailler les différentes étapes de fonctionnement de l'algorithme glouton. Pour chacune d'entre elles nous présentons les difficultés de mise en œuvre ainsi qu'un protocole permettant de l'évaluer. Cette démarche nous a permis de détecter et de corriger les principaux problèmes rencontrés lors de cette mise en œuvre.

Chapitre 1

Apprentissage supervisé

Sommaire

L'objectif de l'apprentissage automatique est de définir des méthodes permettant de résoudre automatiquement des tâches génériques sans les programmer de manière explicite. Les algorithmes d'apprentissage peuvent être regroupés en grandes familles parmi lesquelles les principales sont :

- l'*apprentissage supervisé* pour lequel la tâche est prédéfinie et ou un expert est disponible afin d'indiquer la sortie attendue pour chaque donnée lors de la phase d'apprentissage du modèle [Duda et al., 2000] ;
- l'*apprentissage non supervisé* où seules des données sont disponibles et ou le modèle va mettre en évidence une structure plus ou moins cachée présente dans ces données [Duda et al., 2000] ;
- l'*apprentissage semi-supervisé* applicable lorsque l'expert ne donne une information que partielle sur les données [Chapelle et al., 2010] ;
- l'*apprentissage par renforcement* où l'algorithme apprend un comportement en fonction d'une évaluation des actions qu'il choisit [Sutton and Barto, 1998].

Avec la complexification et l'élargissement des problèmes d'apprentissage étudiés, de nouvelles familles d'algorithmes apparaissent telles que l'*apprentissage par imitation* dont nous reparlerons dans le chapitre 3.

Ce chapitre est dédié à la présentation des problèmes d'apprentissage supervisé, c'est-à-dire des problèmes pour lesquels un ensemble d'exemples

indépendants et disposant d'un étiquetage de référence est fourni. Nous nous intéresserons en particulier au cas ou la sortie recherchée possède une structure complexe.

Nous commençons par introduire le concept de prédiction dans la première section et le développons pour le cas structuré dans la suivante. Les questions d'apprentissage automatique sont ensuite développées dans la troisième section.

1.1 Prédiction

1.1.1 Problématique

On suppose donnés deux ensembles X et Y de nature quelconque que l'on appellera respectivement ensemble d'entrée et de sortie, ainsi qu'une *boite noire* qui à chaque entrée $x \in X$ associe la sortie $y \in Y$ qui lui correspond. Cette boite noire représente généralement un expert humain capable de réaliser ces associations parfaitement.

Par exemple, dans le cas du problème de l'identification d'un objet à partir de sa photo, la photo est une entrée x et le nom de l'objet représenté est une sortie y. On peut imaginer de nombreuses tâches qui entrent dans ce modèle. Si beaucoup de ces tâches sont naturelles pour un humain, elles peuvent être très complexes à réaliser de manière automatique.

La *prédiction automatique* [Duda et al., 2000] consiste à reproduire automatiquement le comportement de cette boite noire, c'est-à-dire d'être capable de trouver la sortie y pour de nouvelles entrées x inconnues. Pour cela, on se basera sur un ensemble limité d'exemples de son fonctionnement :

$$E = (x_m, y_m)_{m=1}^{M} \tag{1.1}$$

Autrement dit, on cherche une *fonction de prédiction* $f : X \to Y$ qui, pour chaque entrée x renvoie la sortie y correcte. On note $f(x)$ le résultat de l'application de cette fonction. Lorsque l'entrée x et la fonction de prédiction f sont claires à partir du contexte, on utilisera la notation $\hat{y}$.

Idéalement, cette fonction de prédiction devrait toujours reproduire les résultats de la boite noire. En pratique, cela est rarement possible et il est nécessaire d'évaluer la qualité des prédictions réalisées. Cette évaluation s'effectue à l'aide d'une *fonction de perte* $L : Y \times Y \to \mathbb{R}^+$ permettant de mesurer la perte réelle $L(y, \hat{y})$ obtenue en prédisant la sortie $\hat{y}$ à la place de la sortie y donnée par la boite noire. Par définition, $L(y, y) = 0$.

Afin de formaliser le problème, en suivant [Vapnik, 1995], on introduit la distribution $\tilde{D}$ (inconnue) sous laquelle les couples d'exemple sont en-

gendrés $x \in X$, $y \in Y$: $(x, y) \sim \tilde{D}$. L'ensemble (1.1) est en fait un tirage selon cette distribution. La quantité reflétant l'espérance de la perte selon la distribution $\tilde{D}$ s'appelle la *fonction de risque* :

$$\mathbb{E}_{(x,y)\sim\tilde{D}}[L(f(x), y)] \tag{1.2}$$

Trouver la fonction de prédiction qui permette de minimiser (1.2) est l'objectif de l'apprentissage.

1.1.2 Classification

Dans ce travail, nous nous intéresserons principalement aux problèmes de prédiction où l'ensemble de sortie Y est fini, autrement dit aux problèmes de *classification*. La classification a pour objectif de choisir une sortie parmi un ensemble fini d'alternatives $y' \in Y$ ayant chacune une perte $L(y, y')$ associée. Une fonction de perte très utilisée dans les problèmes de classification est la *fonction* $0-1$ qui attribue une perte identique à toutes les réponses à l'exception de la réponse correcte :

$$L(y, y') = \begin{cases} 0 & \text{si } y' = y \\ 1 & \text{si } y' \neq y \end{cases} \tag{1.3}$$

Cette fonction de perte peut être utilisée par exemple pour la tâche de reconnaissance d'une personne à partir de l'enregistrement de sa voix, et de manière générale pour toutes les tâches où aucune erreur n'est plus grave qu'une autre.

Dans les situations où différentes erreurs amènent à des conséquences différentes, d'autres fonctions de perte doivent être utilisées. Par exemple, dans le cas de la reconnaissance de maladie grave à partir d'un dossier médical, les conséquences associées à un faux positif sont généralement bien plus faibles que celles associées à une non-détection. En effet, le premier type d'erreur implique une inquiétude inutile et des examens supplémentaires, alors que pour le deuxième les conséquences peuvent aller jusqu'au décès du malade. Dans cette situation, on utilisera plutôt une fonction de perte telle que :

$$L(y, y') = \begin{cases} 0 & \text{si } y' = y \\ 1 & \text{si } y = \text{pas de maladie et } y' = \text{maladie} \\ 1000 & \text{si } y = \text{maladie et } y' = \text{pas de maladie} \end{cases} \tag{1.4}$$

Nous utiliserons par la suite le terme *étiquette* pour désigner la sortie $y' \in Y$, et le terme *classer* pour la procédure de choix d'une étiquette à

associer à une entrée donnée. Afin de classer une entrée, il est nécessaire d'estimer la perte de chaque étiquette possible en lui attribuant un score, puis de choisir la sortie ayant le score le plus bas [1]. La fonction de prédiction a donc la forme suivante :

$$f(x) = \arg\min_{y' \in Y} F(x, y') \tag{1.5}$$

où $F(x, y')$ est une fonction de score, qui idéalement associe le plus petit score à l'étiquette impliquant la perte minimale. La qualité de la fonction de prédiction dépend alors directement de la qualité de la fonction de score utilisée.

1.1.3 Représentation vectorielle

Afin de pouvoir définir la fonction de score, il est nécessaire de transformer chaque couple abstrait (x, y') sous une forme mathématiquement utilisable.

Pour réaliser une tâche donnée, un expert humain va le plus souvent se concentrer sur un ensemble limité d'informations discriminantes qui lui permettent de déterminer la bonne étiquette. Pour offrir les mêmes possibilités au système automatique, chaque couple (x, y') est représenté par un vecteur de caractéristiques $\phi(x, y') \in \mathbb{R}^L$. Ce vecteur contient des valeurs réelles qui représentent différentes informations pertinentes sur le couple (x, y'), par exemple la forme et la couleur des différentes parties d'une image que l'on souhaite identifier.

Le choix d'un bon ensemble de caractéristiques est crucial pour que la tâche soit réalisable. En effet, toute l'information présente dans les données d'origine qui n'est pas mise sous la forme d'une caractéristique ne pourra pas être utilisée par le système. Reconnaître des fruits à partir de photos est ainsi quasiment impossible si aucune information de couleur n'est donnée.

En utilisant une représentation sous forme d'un vecteur de caractéristiques, la fonction de prédiction prend la forme suivante :

$$f_w(x; w) = \arg\min_{y' \in Y} F(\phi(x, y); w) \tag{1.6}$$

où $F : \mathbb{R}^N \to \mathbb{R}$ est une fonction de score pour le vecteur de paramètres w, dont le choix sera explicité à la section 1.3.1.

1. Les scores représentant ici des pertes, il est normal de chercher à les minimiser.

1.2 Prédiction structurée

Deux facteurs principaux influencent la complexité du calcul de (1.6) : la difficulté de calcul des scores $F(\phi(x, y))$ ainsi que la taille de l'ensemble des sorties Y. Si la complexité due au calcul des scores peut être mitigée par un choix approprié des fonctions F et ϕ, le deuxième facteur est plus difficile à maitriser : lorsque la taille de l'ensemble de sortie Y est telle que la recherche de l'élément qui minimise le score ne peut être réalisée en un temps raisonnable, une méthode approchée doit être utilisée.

L'introduction de ce type d'approximations a un coût qui se traduit généralement par une baisse de qualité des prédictions et il peut être difficile, voir impossible, d'obtenir des prédictions satisfaisantes. Dans cette section nous allons nous intéresser à un type particulier de problèmes où la taille de Y rend les calculs exacts impossibles, mais où des méthodes approchées de bonne qualité existent.

1.2.1 Structures de dépendances

Lorsque l'ensemble de sortie Y est de très grande taille, les différentes étiquettes qui le composent ont généralement une structure interne que l'on peut exploiter afin de simplifier les calculs. Ces étiquettes peuvent être découpées en plus petites unités ayant un nombre limité de dépendances entre elles. On peut alors profiter de ce petit nombre de dépendances et du fait qu'un grand nombre de ces unités vont être partagées par plusieurs étiquettes, afin de réduire la quantité de calcul en utilisant par exemple des techniques de programmation dynamique.

La *prédiction structurée* est donc un cas particulier de prédiction où l'on fait l'hypothèse que la sortie y se compose de plusieurs variables aléatoires liées entre elles. Chaque variable représente une unité de prédiction et les éventuelles dépendances entre les variables relient les différentes unités. Il est important que les unités, ainsi que les liens entre elles, soient choisis en fonction de la nature de la tâche mais aussi de manière à permettre une factorisation efficace des calculs.

Nous appellerons *structure de dépendances* le graphe dont les nœuds représentent les variables aléatoires et dont les arcs représentent les dépendances entre les variables. Ce graphe de dépendances peut avoir une structure arbitraire, adaptée à une tâche donnée ; dans le cas de la prédiction structurée on retrouve le plus souvent des structures telles que des séquences ou des arbres.

1.2.2 Espace de recherche

L'*espace de recherche* est un ensemble d'instances [2] de la structure de dépendances représentant l'ensemble de sortie Y organisé de manière à rendre les calculs efficaces. *L'espace de recherche sur les séquences* permet une présentation simple ; ce type d'espace de recherche est le plus courant en pratique, et le seul que nous utiliserons dans ce travail [3], c'est pourquoi nous allons concentrer notre présentation sur celui-ci. L'extension à des structures telles que les arbres ou graphes se fait naturellement et nous les évoquerons lorsque des points particuliers seront à noter.

En supposant donc que la structure de dépendances soit une séquence, l'espace de recherche est un graphe comportant un nœud initial et un nœud final qui sont reliés par des chemins représentant les différentes prédictions possibles. Les arcs de ce graphe sont pondérés de manière à ce que la somme des poids des arcs d'un chemin représente la perte totale associée à la prédiction correspondante. La résolution du problème de minimisation (1.6) est équivalente à la recherche du plus court chemin dans cet espace de recherche.

Sous sa forme la plus naïve, l'espace de recherche est un graphe dans lequel chaque chemin est indépendant des autres (n'a aucun nœud commun avec un autre chemin). Il comporte donc un nombre de nœuds proportionnel à la taille de l'ensemble Y et le calcul de son plus court chemin n'est donc pas plus efficace qu'une prédiction non-structurée. Mais si les dépendances le permettent, il est possible de factoriser ce graphe en exploitant la structure commune des chemins de manière à obtenir un treillis de recherche comme illustré dans la figure 1.1. Sous cette forme, chaque nœud est partagé par un grand nombre de chemins et la recherche du plus court chemin peut être réalisée de manière plus efficace.

1.2.3 Inférence dans un espace de recherche

La recherche du plus court chemin dans cet espace de recherche factorisé peut se faire de manière exacte à l'aide par exemple de l'algorithme de Viterbi [Viterbi, 1967]. Une première passe sur le graphe (appelée passe *forward*) remplace les scores locaux par des scores globaux ; le score global sur un arc représente le score minimal de tous les chemins partant du nœud initial jusqu'au nœud cible de l'arc et passant par cet arc. Cette trans-

2. Une instance de la structure de dépendance est un graphe de même structure où chaque nœud prend une valeur concrète de la variable aléatoire associée.

3. Nous verrons au chapitre 3 que cela ne nous limite pas à des tâches de prédiction sur les séquences.

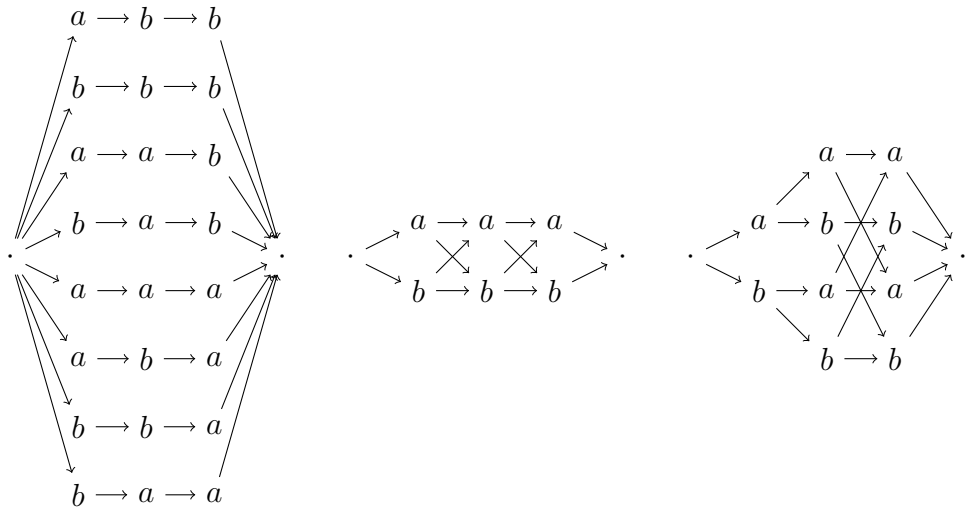


FIGURE 1.1 – Exemple de factorisation d'un espace de recherche sous la forme d'un treillis.

formation des scores se fait par programmation dynamique en parcourant le graphe depuis l'état initial. Le score de chaque arc est remplacé par la somme de son score local et du score minimal des arcs entrant de son nœud source. Par construction, une fois cette passe terminée, l'arc ayant le poids minimal amenant au nœud final nous indique le score du plus court chemin dans le graphe. Une deuxième passe (appelée passe *backward*) partant cette fois-ci du nœud final permet de retrouver le chemin correspondant en suivant les arcs de poids minimal.

La programmation dynamique permet souvent d'exploiter la factorisation du graphe et de rendre le plus court chemin calculable en un temps raisonnable mais ce n'est pas toujours le cas. Si le graphe de dépendance est plus complexe qu'une séquence et contient des cycles par exemple, il n'existe pas dans le cas général d'algorithme exact et il est souvent nécessaire d'utiliser des algorithmes approchés tels que la propagation de confiance [Pearl, 1988] ou des méthodes variationnelles [Beal, 2003]. Lorsque la structure est sans cycle, une recherche exacte telle que présentée nécessite de parcourir complètement le graphe ce qui peut être difficile même lorsque celui-ci est factorisé. Il est dans ce cas aussi nécessaire d'utiliser des approximations telle que la recherche gloutonne ou la recherche en faisceau [Cormen et al., 1990].

La recherche gloutonne est similaire à la passe arrière de l'algorithme de

Viterbi effectuée dans le sens inverse : la solution est construite étape par étape, en partant de l'état initial et en choisissant les arcs ayant un score minimal. Dans le cas général, cette méthode est sous-optimale et ne permet pas de trouver le meilleur chemin de manière sûre. Néanmoins il existe des problèmes pour lesquels la recherche gloutonne permet d'obtenir la solution exacte. C'est notamment le cas de la passe arrière de l'algorithme de Viterbi où les scores représentent une information globale sur toute la suite du chemin. Notons que la passe forward présentée est un algorithme qui permet de mettre sous cette forme un graphe avec des scores quelconques et ainsi de rendre la recherche gloutonne exacte. Dans le chapitre 3 nous allons considérer des problèmes similaires où les scores sur les arcs représentent une information sur la perte globale associée à un arc et sont obtenus en normalisant les scores de la première passe de Viterbi [4].

1.3 Apprentissage simple et structuré

1.3.1 Fonctions paramétriques

Pour formuler le problème de recherche de la fonction de prédiction comme un problème d'optimisation, nous allons restreindre la recherche à une classe paramétrique de fonctions. Cette classe est de nature arbitraire, notamment il n'est pas obligatoire qu'elle contienne la fonction de la boite noire (et c'est rarement le cas), mais plus elle est riche et diverse, plus elle permet de trouver une bonne approximation pour la fonction recherchée. Par exemple, la classe des fonctions polynomiales permet d'approcher une fonction quelconque avec une précision d'autant plus grande que le dégré des polynômes considérés est élevé.

En pratique, deux facteurs font que la boite noire ne peut souvent pas être imitée parfaitement dans la classe considérée. Ceci est dû d'une part au manque d'information dans les arguments de la fonction (les caractéristiques) et d'autre part à la famille de fonctions considérées. Si la fonction que réalise la boite noire n'est pas dans la classe considérée, il est alors bien sûr impossible de l'obtenir en faisant des recherches dans cette classe, mais si les informations utilisées par la boite noire ne sont pas disponibles, aucune classe de fonction ne pourra imiter son comportement.

L'ensemble des fonctions de la classe considérée est décrit par un vecteur de paramètres w. En faisant varier les valeurs de ces paramètres dans l'en-

4. Plus précisément, ces scores que l'on appellera dans la section 3.1.3 les pertes de potentiel, représentent la différence entre le meilleur score qui peut être obtenu en passant par l'arc considéré et le meilleur score atteignable depuis l'état source de cet arc.

semble de paramètres W correspondant à la classe de fonctions considérée, il est possible de chercher des fonctions imitant plus ou moins bien le fonctionnement de la boite noire selon le critère (1.2). On obtient donc le problème d'optimisation suivant :

$$w = \arg\min_{w \in W} \mathbb{E}_{(x,y)\sim\tilde{D}}[L(f_w(x), y)] \tag{1.7}$$

1.3.2 Risque empirique

L'optimisation directe du problème (1.7) est impossible car la distribution $\tilde{D}$ est inconnue. L'ensemble d'apprentissage (1.1) étant obtenu par une série de tirages indépendants selon cette distribution, l'espérance mathématique dans (1.7) peut être approchée par l'espérance empirique :

$$w = \text{argmin} \frac{1}{M} \left(\sum_{m=1}^{M} [L(f_w(x_m), y_m)] \right) \tag{1.8}$$

Plus l'ensemble d'apprentissage est grand, plus l'espérance de la perte sous la distribution $\tilde{D}$ pourra être approximée fidèlement. Une bonne approximation de la fonction de risque améliore les possibilités de bien choisir la fonction de prédiction.

On appelle *modèle idéal pour l'ensemble d'apprentissage* la fonction de prédiction qui donne les bonnes étiquettes pour tous les éléments de l'ensemble d'apprentissage. Autrement dit, $L(f(x_m), y_m) = 0$ pour tout m. Ce modèle idéal n'existe pas toujours. Un exemple trivial où le modèle idéal n'existe pas est le cas où il existe deux exemples d'apprentissage (x_1, y_1) et (x_2, y_2) avec $y_1 \neq y_2$ tels que

$$\phi(x_1, y_1) = \phi(x_2, y_1) \quad \text{et} \quad \phi(x_2, y_1) = \phi(x_2, y_2)$$

Dans cet exemple les caractéristiques utilisées ne permettent pas de faire la distinction entre les exemples d'apprentissage x_1 et x_2. C'est le cas par exemple si, pour reconnaitre un objet sur une photo, les seules caractéristiques utilisées sont sa couleur et sa forme. Il ne sera pas possible dans ce cas de faire la distinction entre une orange et un abricot.

1.3.3 Recherche du modèle idéal

Lorsque le modèle idéal pour l'ensemble d'apprentissage existe, il peut être trouvé avec la méthode du *perceptron* [Rosenblatt, 1958], présentée dans l'algorithme 1. Le principe du perceptron est de traiter les exemples

d'apprentissage un par un et de mettre à jour le modèle à chaque fois qu'un exemple est mal classé par le modèle courant. La mise-à-jour s'effectue de manière à ce que le modèle ait plus de chances de bien classer cet exemple la prochaine fois. On peut montrer que si le modèle idéal existe, cette procédure converge en un nombre fini d'itérations.

Algorithme 1 : Algorithme de perceptron

Données : $E = (x_m, y_m)_{m=1}^{M}$, nombre d'itérations N
Initialisation : $w = 0$; **pour** $n \in (1, N)$ **faire**
  Réordonner E aléatoirement;
  pour $i \in (1, M)$ **faire**
    $\hat{y} = \arg\max_{y' \in Y} w \cdot \phi(x_i, y')$;
    si $\hat{y} \neq y_i$ **alors**
      $w = w + \phi(x_i, y_i) - \phi(x_i, \hat{y})$
    fin
  fin
fin
retourner w;

La méthode du perceptron peut aussi être appliquée dans le cas où le modèle idéal pour l'ensemble d'apprentissage n'existe pas, mais dans le cas général, on ne dispose pas de garantie ni sur la convergence, ni sur la performance du modèle obtenu. En pratique, même sans garantie cette méthode donne des résultats souvent satisfaisants, notamment si elle est renforcée par des astuces telles que moyenner les poids, ou utiliser un vote [Freund and Schapire, 1999]. Dans la section suivante nous allons étudier plus en détail ce cas où le modèle idéal n'existe pas dans la classe des fonctions considérée.

1.3.4 Risque empirique régularisé

En pratique, même si le modèle idéal pour un ensemble d'apprentissage existe, il est rarement souhaitable de l'obtenir, car en général des performances parfaites sur les données d'apprentissage sont un signe de sur-apprentissage. C'est en général le cas dans les problèmes d'apprentissage où la quantité d'informations utilisées est significativement plus grande que ce qui est nécessaire pour construire un modèle de bonne qualité. Par exemple, pour un problème de reconnaissance d'objets à partir de photos, disposer d'informations très précises sur les nuances de couleurs permettra, s'il y a autant de nuances différentes que d'exemple d'apprentissage,

d'associer parfaitement chaque exemple à sa nuance. Un modèle appris de cette manière n'aura aucune capacité de généralisation et donc de mauvaises performances en pratique.

Afin de contourner cet effet, on évite de chercher le modèle qui à tout prix classe parfaitement les exemples d'apprentissage. Pour cela, on ajoute un terme supplémentaire de régularisation dans la fonction objectif ; ce terme pénalise les paramètres ayant de trop grandes valeurs et donc favorise les modèles les plus simples.

Nous allons utiliser la régularisation ℓ_2 qui se traduit par l'ajout d'un terme $\lambda||w||_2^2/2$ dans la fonction objectif [Ng, 2004] (la norme au carré et le facteur $1/2$ sont juste des formalités qui s'utilisent par convention pour simplifier certains calculs) :

$$w = \text{argmin}\, \frac{1}{M}\left(\sum_{m=1}^{M}[L(f_w(x_m), y_m)] + \frac{\lambda}{2}||w||^2\right) \tag{1.9}$$

Le paramètre λ permet de choisir la force de la régularisation ; la valeur de ce méta-paramètre est en général choisie en utilisant un jeu de données de validation supplémentaire. Ces données permettent de faire des tests intermédiaires pour différentes valeurs de ce paramètre afin de choisir la meilleure. D'autres formes de régularisation existent et peuvent être préférés dans différentes situations [Bishop, 2006].

1.3.5 Optimisation et garanties

Le problème (1.9) peut être résolu par des techniques d'optimisation numérique classiques. Par exemple, si la fonction de perte L est convexe, alors la fonction objectif est également convexe, et le problème peut être résolu par une méthode de descente de gradient [Boyd and Vandenberghe, 2004] (à condition que l'on sache le calculer) ; si ce n'est pas le cas, d'autres techniques d'optimisation doivent être utilisées. Notons que le cas d'une fonction objectif non-convexe est le cas le plus fréquent en pratique ; notamment la fonction de perte $0-1$ n'est pas convexe.

Afin de résoudre (1.9) de manière approchée, il est possible d'approximer la fonction objectif par une fonction convexe. Pour cela, la fonction de perte $L(y, \hat{y})$ est remplacée par une approximation convexe $\breve{L}(y, x, w)$ [5], ce qui

5. La fonction de perte peut être approximée par une fonction qui dépend éventuellement d'autres arguments que y et $\hat{y}$

transforme le problème de la manière suivante :

$$w = \arg\min \frac{1}{M}\left(\sum_{m=1}^{M}[\breve{L}(y_m, x_m, w)] + \frac{\lambda}{2}||w||^2\right) \tag{1.10}$$

Remplacer la fonction de perte $\breve{L}$ par la *Hinge loss* permet de retrouver la méthode des séparateurs à vaste marge [Cortes and Vapnik, 1995] :

$$\breve{L}(y, x, w) = \max_{y' \in Y}(w \cdot \phi(x, y') - w \cdot \phi(x, y) + L(y, y')) \tag{1.11}$$

Cette méthode n'optimise pas uniquement la perte obtenue $L(y, y')$, mais aussi la marge $w \cdot \phi(x, y') - w \cdot \phi(x, y)$. Cela assure que la bonne réponse y est suffisamment bien séparée des autres étiquettes c'est-à-dire que le score du modèle est significativement plus petit pour y que pour $y' \neq y$. Ceci peut être vu comme une manière d'améliorer la capacité de généralisation du modèle : on souhaite que non seulement il classe correctement le maximum d'exemples, mais aussi qu'il le fasse avec une marge pour le cas où les exemples de test sont légèrement différents. Cette fonction objectif n'est pas différentiable, mais elle peut être optimisée par une descente de sous-gradient , ce qui garantit de trouver le meilleur modèle selon le critère (1.10) avec la Hinge loss comme fonction de perte [Shalev-Shwartz et al., 2007].

Un autre exemple de convexification est le remplacement de $\breve{L}$ par une perte probabiliste, ce qui nous amène à la méthode de maximum d'entropie [Berger et al., 1996] :

$$\begin{aligned} \breve{L}(y, x, w) &= w \cdot \phi(x, y) - \log Z(w, x) \\ Z(w, x) &= \sum_{y' \in Y} \exp(w \cdot \phi(x, y')) \end{aligned} \tag{1.12}$$

Cette méthode cherche à donner la plus grande probabilité à la bonne étiquette. La fonction objectif est convexe, et son gradient est classiquement donné par l'expression suivante :

$$\begin{aligned} \Delta_w(\breve{L}(y, x, w)) &= \frac{1}{M}\left(\sum_{m=1}^{M} -\phi(x_m, y_m) + \mathbb{E}_w(\phi(x_m, Y)) + \lambda w\right) \\ \mathbb{E}_w(\phi(x, Y)) &= \textstyle\sum_{y' \in Y} \mathbb{P}_w(y'|x)\phi(x, y') \\ \mathbb{P}_w(y'|x) &= \frac{\exp(w \cdot \phi(x, y'))}{Z(w, x)} \end{aligned} \tag{1.13}$$

La descente de gradient *garantit de trouver le meilleur modèle selon le critère* (1.10) *en utilisant la fonction de perte* (1.12).

Une variante de la fonction de perte (1.12) permettant de prendre en compte les différentes pertes associées aux étiquettes (ce qui correspond au problème de classification sensible aux coûts [Domingos, 1999]) est introduite dans [Gimpel and Smith, 2010] :

$$\begin{aligned} \breve{L}_{\text{cs}}(y, x, w) &= w \cdot \phi(x, y) - \log Z_{\text{cs}}(w, x) \\ Z_{\text{cs}}(w, x) &= \sum_{y' \in Y} \exp(w \cdot \phi(x, y') + L(y, y')) \end{aligned} \tag{1.14}$$

La fonction objectif reste convexe et le gradient est calculable de manière analogue à (1.13) ; l'optimisation par descente de gradient garantit alors de trouver le meilleur modèle selon le critère (1.10) avec la fonction de perte (1.14) [6].

Pour résumer, l'algorithme 2 décrit l'apprentissage avec une fonction de perte convexe arbitraire par la méthode de descente de gradient basique. Le pas η doit être choisi par exemple en utilisant les conditions de Wolfe [7].

Algorithme 2 : Apprentissage avec descente de gradient

Données : $E = (x_m, y_m)_{m=1}^{M}$, λ, nombre d'itérations N

Initialisation : $w = 0$; **pour** $n \in (1, N)$ **faire**
  pour $i \in (1, M)$ **faire**
    calculer $\Delta_w \breve{L}(x_m, y_m, w)$;
  fin
  actualiser le modèle $w = w + \eta \left(\frac{1}{M} \sum_{m=1}^{M} \Delta_w \breve{L}(x_m, y_m, w) + \lambda w \right)$;
fin
retourner w;

Il faut noter une ressemblance entre les algorithmes 1 et 2, même si leurs idées de base sont différentes. Notamment, dans le cas où la descente

6. En pratique le problème de classification sensible aux coûts est souvent résolu par des moyens empiriques tels que la réduction à une régression [Tu and Lin, 2010] ou à une classification binaire avec pondération des exemples d'apprentissage [Zadrozny et al., 2003]. Des heuristiques similaires seront utilisées dans les chapitres 4 et 5 de cette thèse.

7. En pratique on utilise généralement des variantes plus avancées de la descente de gradient, telles que BFGS ou L-BFGS [Liu and Nocedal, 1989].

de gradient telle qu'elle est présentée ci-dessus ne peut pas être réalisée en temps raisonnable (le calcul du gradient complet sur toutes les données d'apprentissage peut être couteux), sa variante stochastique [Bottou, 1991] (algorithme 3) peut être utilisée. Cette dernière peut être vue comme un intermédiaire entre les deux premières méthodes. Notons que dans ce cas aucune des garanties sur les performances vues précédemment n'est applicable : la mise-à-jour, qui est différente de celle du perceptron, fait que la séparation des données séparables n'est plus garantie, et l'approximation du calcul du gradient ne garantit plus d'atteindre l'optimum global.

Algorithme 3 : Apprentissage avec descente de gradient stochastique

Données : $E = (x_m, y_m)_{m=1}^{M}$, λ, nombre d'itérations T
Initialisation : $w = 0$; **pour** $t \in (1, T)$ **faire**
  pour $i \in (1, M)$ **faire**
    calculer $\Delta_w \breve{L}(x_m, y_m, w)$;
    actualiser le modèle $w = w + \eta \left(\Delta_w \breve{L}(x_m, y_m, w) - \frac{\lambda w}{M} \right)$;
  fin
fin
retourner w;

La convexification n'est pas la seule méthode d'optimisation d'une fonction non-convexe ; le domaine de l'optimisation numérique propose de nombreuses méthodes de recherche d'optimum, qui sont plus ou moins adaptées suivant les cas [Nocedal and Wright, 2006]. La méthode du recuit simulé [Kirkpatrick et al., 1983, Cerny, 1985] propose d'approcher une fonction non-convexe par une fonction convexe en plusieurs étapes : chaque nouvelle approximation est construite de manière à mieux approximer la zone où, d'après l'approximation précédente, l'optimum se trouve. L'optimisation peut aussi être réalisée coordonnée par coordonnée [Wright, 2015], au lieu de faire la descente de gradient sur l'ensemble des variables. Cette méthode est souvent appliquée quand le nombre de coordonnées est relativement petit. L'optimisation sur une grille (ou exhaustive) utilise de multiples points pour explorer différentes zones de la fonction et trouver les meilleurs points expérimentalement. Ce type d'optimisation est souvent utilisé afin de trouver les bonnes valeurs des hyperparamètres tels que la valeur de λ dans (1.9) [Bergstra and Bengio, 2012]. Dans le cas général ces méthodes n'ont pas de garanties théoriques de trouver l'optimum global comme dans le cas convexe ; en pratique leurs performances dépendent de nombreux facteurs. Par exemple, la qualité du modèle trouvé par l'optimisation sur une

grille dépend directement du nombre de points explorés ; les performances peuvent être améliorées en utilisant une recherche raffinée dans les zones qui ont montrées les meilleurs résultats lors de la recherche sur une grille à gros grain [Hsu et al., 2010].

1.3.6 Apprentissage structuré

Les méthodes d'apprentissage présentées dans les sections précédentes nécessitent toutes de réaliser un calcul sur l'ensemble complet des étiquettes possibles. Par exemple, la prédiction selon le modèle (1.6) demande de trouver l'étiquette qui minimise le score et le calcul du gradient (1.13) nécessite une somme sur l'ensemble des étiquettes. Dans le cadre de l'apprentissage structuré, ces calculs doivent être réalisés sur le graphe factorisé tel que nous l'avons vu dans la section 1.2 [8].

La possibilité de réaliser les calculs de manière exacte permet de construire une généralisation naturelle pour ces méthodes d'apprentissage. Ainsi, l'algorithme du perceptron devient le *perceptron structuré* [Collins, 2002, Roark et al., 2004] ; la seule différence apportée par la structure se trouve au niveau du calcul de la prédiction (1.6) qui s'effectue à l'aide de l'algorithme du plus court chemin à la place d'une simple énumération. De la même manière, SVM se généralise dans sa version structurée SVM struct [Tsochantaridis et al., 2005] : comme pour le perceptron, la prédiction est faite à l'aide de l'algorithme du plus court chemin. La méthode du maximum d'entropie se transforme en CRF (eng. Conditionnal random field [Lafferty et al., 2001]) ; en plus du calcul du maximum, la somme est aussi calculée par l'algorithme de plus court chemin généralisé. Bien évidemment, les garanties théoriques restent les mêmes : le perceptron converge vers le modèle idéal s'il existe dans la classe considérée, les fonctions objectifs (1.11) et (1.12) restent convexes avec un gradient calculable de manière exacte, ce qui permet de trouver l'optimum global.

Dans le cas où les calculs ne peuvent pas être réalisés de manière exacte, différentes manières de réaliser l'apprentissage et l'inférence sont possibles. Notamment, une approche simple consiste à agir comme si les calculs étaient exacts. Cette approche permet souvent d'obtenir de bons résultats en pratique ([Smith and Eisner, 2008], [Vishwanathan et al., 2006], [Sutton et al., 2007]), bien que dans le cas général les garanties théoriques ne soient plus valables ([Huang et al., 2012],

8. Notre présentation s'est limitée au calcul du plus court chemin car aussi bien le minimum que la somme sur des chemins d'un graphe peut être calculé de cette manière lorsque le bon semi-anneau est utilisé [Mohri, 2002].

[Kulesza and Pereira, 2008]). Mais il existe aussi de nombreux travaux où le processus d'apprentissage se fait de manière différente et adaptée aux nouvelles conditions (par exemple, [Collins and Roark, 2004], [Huang et al., 2012], [Stoyanov and Eisner, 2012], [Domke, 2011]). Notamment, dans le chapitre 3 nous étudierons en détails les techniques d'adaptation pour une famille de méthodes d'apprentissage structuré basées sur l'inférence approchée gloutonne qui s'appelle « apprendre à chercher ».

1.4 Conclusion

Dans ce chapitre, nous avons présenté les bases de l'apprentissage supervisé nécessaires à la bonne compréhension de ce travail, en particulier le cas où la sortie recherchée possède une structure complexe.

Dans le cas où la structure choisie est suffisamment simple pour qu'il soit possible de réaliser les calculs nécessaires en un temps raisonnable, l'apprentissage structuré ne diffère que peu de l'apprentissage simple. La suite de ce travail est dédiée à l'étude des cas où ce n'est pas possible et où il est nécessaire de réaliser les calculs de manière approchée.

Dans le chapitre suivant, nous présentons différentes taches du traitement automatique de la langue qui présentent des structures complexes pour lesquelles cette approche est nécessaire.

Chapitre 2

Problèmes structurés en traitement automatique des langues

Sommaire

2.1 Introduction

Dans ce travail nous concentrons nos études expérimentales autours d'applications du traitement automatique des langues naturelles. Dans ce chapitre nous faisons une brève revue de ce domaine, en faisant particulièrement attention aux aspects structurés des tâches pratiques considérées dans les chapitres 4 et 5.

En linguistique, de nombreuses théories existent pour décrire la structure des langues naturelles [1]. Sans entrer dans les détails, remarquons qu'un point commun de ces théories est le fait que la langue naturelle possède une

1. Il existe de nombreuses définitions des langues naturelles dans la littérature et en choisir une est hors du cadre de ce travail. Nous nous contenterons ici d'une définition simple nous permettant de présenter les objets linguistiques avec lesquels nous allons travailler : les *langues naturelles* sont les langues écrites ou parlées (ou qui l'étaient

structure hiérarchique constituée de plusieurs niveaux, les éléments d'un niveau se combinant pour former les éléments du niveau supérieur. De manière générale, on trouve notamment les niveaux suivants : les phonèmes, les morphèmes, les mots, les phrases et les textes. Ces niveaux nous permettent d'introduire la notion suivante : on appel *structure linguistique* un élément d'un niveau composé d'éléments de niveau inférieur.

Tous les niveaux de la langue naturelle partagent, de manière plus ou moins importante, une propriété commune : ils sont incomplets. Cela signifie que toutes les combinaisons d'éléments de niveau inférieur ne forment pas une structure linguistique correcte. Par exemple, seule une petite partie des combinaisons de lettres forment un mot, de même qu'une faible proportion des combinaisons de mots forment une phrase. Il existe donc des contraintes entre les éléments pouvant être combinés afin de former une structure linguistique. Par conséquent, on observe une certaine redondance d'information dans la langue naturelle [Schneier, 1995]. Ces contraintes, que nous appellerons *dépendances*, sont d'une aide précieuse pour des applications en traitement des langues lorsque l'on souhaite résoudre des problèmes d'apprentissage et d'inférence.

Dans ce travail, nous avons choisi de nous concentrer sur les trois niveaux principaux communs à la majorité des langues : le mot, la phrase et le texte. Ces trois niveaux nous permettront de présenter des dépendances de natures différentes au travers d'exemples de tâches réelles, étudiées dans les prochains chapitres.

2.2 Les fondations : les mots

Le mot est l'élément de base de la langue, en tant qu'objet linguistique il possède différentes caractéristiques telles qu'une sonorité fixe, une représentation écrite, une structure syllabique et dans certaines langues un accent tonique. Suivant la tâche à réaliser, un mot peut être vu comme un ensemble de lettres, de syllabes ou de phonèmes formant une séquence.

De fortes dépendances locales entre les éléments sont présentes et limitent les combinaisons possibles, certains enchainement de phonèmes sont par exemple impronon̂cables, la grammaire nécessite d'accorder les mots entre eux, etc. Chaque langue naturelle a bien sûr sa propre sonorité et écriture ; les différentes dépendances sont alors particulières à la langue. Le russe et le français sont par exemple très différents au niveau des com-

comme par exemple le *latin*) par les humains, qui ont été créées spontanément au fil du temps avec pour objectif de communiquer [Wikipédia, article *Natural language*, visité le 17 décembre 2017].

binaisons voyelle-consonne. Il est habituel en russe d'avoir quatre ou plus consonnes consécutives, et parfois jusqu'à sept consonnes prononçables de suite (par exemple, le mot ко**нтрвзгля**д (que l'on pas traduire par *avis opposé*) contient sept consonnes consécutives) là où en français on retrouve rarement plus de trois consonnes prononçables de suite. Dans les paragraphes qui suivent nous présentons deux tâches qui portent sur le niveau lexical : la reconnaissance de mots isolés et la transcription graphème-phonème.

2.2.1 Reconnaissance de l'écriture manuscrite

La reconnaissance de l'écriture manuscrite consiste à retrouver un mot à partir d'une image de sa forme écrite [Kassel, 1995]. Ses applications sont très variées mais l'on peut citer notamment la numérisation de documents qui est un enjeu important actuellement.

L'exemple illustré par la figure 2.2.1 met en évidence l'importance des dépendances entre les lettres voisines (appelées aussi dépendances à courte distance). Le mot à reconnaitre ici est le mot « braque » mais certaines lettres sont ambigües, par exemple le « r » peut facilement être confondu avec un « v ». La structure linguistique est exploitée pour lever ces ambigüités, la séquence de lettre « bv » est très peu observée en français et même absente en début de mot contrairement à la séquence « br ».

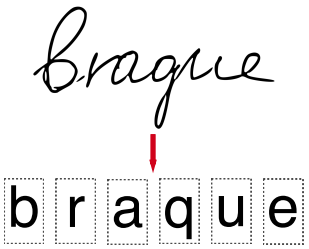


FIGURE 2.1 – Exemple de reconnaissance de l'écriture manuscrite.

De manière générale, les dépendances à longue distance ont une importance moins claire pour les tâches lexicales : on peut par exemple noter des cas relativement rares tels que la présence d'un préfixes « re » et « de » en français qui augmentent la probabilité que le mot considéré soit un verbe et donc la probabilité que ses dernières lettres forment une terminaison de verbe conjugué.

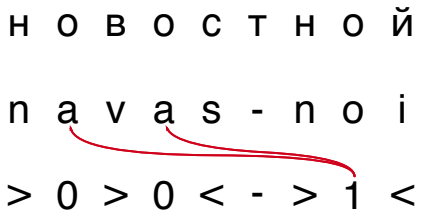


FIGURE 2.2 – Cet exemple met en évidence l'utilité de dépendances à longue distance afin de prédire correctement la prononciation de certaines lettres. Le fait que l'accent tonique (noté *1*) se place sur la dernière syllabe fait que les deux premières lettres o se prononcent comme a, car elles sont non-accentuées.

2.2.2 Prédiction de la prononciation

L'objectif de cette tâche est de déterminer la prononciation d'un mot à partir de son écriture [Allen et al., 1987]. Son application principale est la constitution de dictionnaires de prononciation pour la reconnaissance et la synthèse vocale notamment pour les mots inconnus et les noms propres, mais aussi pour les mots plus courants dans le cas des langues peu dotées.

La tâche de prononciation comporte deux composants essentiels : la prédiction de la suite de phonèmes à prononcer et la prédiction de la structure syllabique associée à ces phonèmes (les accents toniques primaires et secondaires ainsi que les rattachements entre les voyelles et les consonnes) [Sejnowski and Rosenberg, 1987]. Ces deux composants permettent d'obtenir une représentation canonique de la prononciation mais il est possible d'aller plus loin en proposant aussi des variantes de prononciation [Allen et al., 1987].

Les dépendances locales sont particulièrement importantes dans cette tâche, notamment parce que plusieurs lettres vont parfois se combiner pour former un seul son comme pour les séquences *au* et *eau* en français. Mais aussi parce que certaines lettres changent de prononciation en fonction du contexte tel que le *m* en russe qui devient muet dans la séquence *стн*).

Les dépendances à longue distance apportent aussi beaucoup d'information. Par exemple, un mot ne peut avoir plus d'un accent tonique principal, ce qui crée une dépendance entre toutes les voyelles du mot. Cet accent tonique peut modifier la séquence phonémique ce qui ajoute des dépendances entre les deux composants de la tâche. Ce phénomène est illustré par la figure 2.2.2 pour le mot russe *новостной*. L'accent principal est placé sur le troisième *o* qui se prononce alors *o*. Les deux premiers *o* n'étant pas accentués sont réduits et donc prononcés *a*.

2.3 Les murs : les phrases

Dans cette section nous allons nous intéresser au niveau intermédiaire que sont les phrases [2]. Ce niveau est particulièrement important en traitement automatique des langues car c'est à ce niveau que sont réalisées une grande partie des tâches classiques.

En effet, les phrases forment des objets linguistiques consistants, c'est-à-dire qu'il est possible d'étudier par exemple la grammaire, la syntaxe ou la sémantique d'une phrase sans nécessairement connaitre les phrases qui la précèdent ou la suivent. Elles représentent donc une portion de texte à la fois suffisamment grande pour contenir une information intéressante à extraire, tout en étant suffisamment petite pour que l'on puisse appliquer de manière efficace des méthodes de prédiction structurée.

Tout comme au niveau des mots, les dépendances locales entre éléments voisins sont très utiles et les plus exploitées. Nous verrons dans les tâches que nous allons présenter que des dépendances entre des mots plus éloignés sont toutefois souvent utiles voire indispensables pour les réaliser correctement.

2.3.1 Étiquetage morpho-syntaxique

L'étiquetage morpho-syntaxique a pour objectif d'assigner à chaque mot sa catégorie grammaticale, c'est-à-dire une étiquette indiquant son rôle au sein de la phrase, ainsi que d'éventuelles informations sur sa morphologie telles que le genre, le nombre, le cas ou encore l'aspect [DeRose, 1988], [Toutanova et al., 2003].

Cette tâche, tout comme l'analyse syntaxique de surface qui regroupe les mots en syntagmes [Abney, 1991], n'a que peu d'utilité directe. Son principal intérêt est d'extraire une information linguistique générale des mots et des groupes des mots qui sera exploitée pour des tâches plus complexes telles que la reconnaissance d'entités nommées [Ehrmann, 2008], l'analyse syntaxique en dépendances [Nivre, 2005] ou encore la traduction que nous présenterons au chapitre 5.

En pratique, un simple dictionnaire permet d'attribuer correctement un nombre important d'étiquettes, la difficulté de cette tâche provient des mots dont la fonction est ambigüe ainsi que des mots inconnus. La catégorie grammaticale de ces mots peut être déterminée le plus souvent en exploitant des dépendances locales liant les mots proches. Ainsi, en français entre un

2. Dans ce travail, nous considérerons les phrases comme de simples suites de mots. Il est parfois nécessaire de les étudier à niveau plus fin dans le cas des langues agglutinantes où pour les langues n'ayant pas de notion de mot.

déterminant et un adjectif on trouvera généralement un nom ; de même, un adjectif adjacent à un nom féminin sera lui aussi probablement féminin.

De manière générale, cette tâche est considérée comme bien maîtrisée lorsque suffisamment de données d'apprentissage sont disponibles. Pour les langues moins bien dotées elle reste toutefois une étape qui peut être difficile et pourtant cruciale pour la réalisation de tâches plus complexes.

2.3.2 Traduction automatique

La tâche de traduction automatique [Koehn, 2010] consiste à retranscrire dans une langue cible, une phrase d'une langue source, en préservant sa sémantique. La traduction automatique est d'une grande importance et très demandée par le grand public. Il est en effet de plus en plus important de pouvoir dialoguer ou comprendre des documents dans différentes langues, tandis que les services offerts par des traducteurs humains sont souvent inadaptés (trop chers ou trop lents par exemple).

L'approche la plus simple – la traduction mot à mot – produit des phrases de mauvaise qualité et souvent sémantiquement incohérentes. Afin de produire de bonne traductions, il est nécessaire de prendre en compte les dépendances entre les mots générés. Notamment, une même phrase peut souvent être traduite de différentes manières et il est donc nécessaire de s'assurer que les différentes portions de la traduction sont cohérentes. Par exemple, pour traduire en français la phrase anglaise suivante :

« *The way you choose will be long and thorny.* »

la traduction du mot *way*, qui peut être par exemple *chemin* ou *voie*, va déterminer le genre des adjectifs *long(-ue)* et *épineux(-se)*.

L'utilisation de dépendances sémantiques permet de construire une phrase non seulement grammaticalement correcte, mais aussi fluide et naturelle, en prenant en compte les subtilités de la langue. Dans l'exemple précédent, si le mot *way* est traduit plutôt par *manière* ou *façon*, alors l'adjectif *thorny* ne doit pas être traduit littéralement comme *épineuse* et doit plutôt être remplacé par exemple par *ardue*.

La traduction automatique est une tâche très complexe qui se heurte à de nombreuses difficultés telles que l'ambiguïté des mots qui ne peut être résolue sans prendre en compte les dépendances longues ; mais aussi d'autres difficultés telles que les expressions idiomatiques ou encore les références culturelles différentes.

2.3.3 Réinflexion

La tâche de réinflexion consiste à réintroduire des traits morphologiques telles que le genre, le nombre où le cas dans une phrase où ils sont absents [Fraser et al., 2012]. La réinflexion est utilisée en post-traitement de la tâche de traduction automatique (ou plus généralement, d'une tâche de génération de texte) qui peut dans ce cas être rendue plus simple.

Dans le cas de la traduction automatique par exemple, certaines information morphologiques n'ont pas de correspondances entre les deux langues. Comme nous l'avons vu dans la section précédente, le genre est souvent absent en anglais mais doit être réintroduit lors de la traduction vers le français. Il peut être plus simple de l'inférer de manière séparée du processus de traduction.

Les données prises en entrée sont donc des phrases dont les mots ont été privés de certains traits morphologiques. Suivant le contexte, il peut s'agir de tous les mots ou de seulement une partie d'entre eux. De même, tous les traits peuvent être manquants ou seulement certains d'entre eux. Dans l'exemple de la section précédente, un système de traduction produirait la sortie suivante (les mots en gras sont les mots privés de leur genre) :

« ***Le***+*G voie que tu as* ***choisi***+*G est* ***long***+*G et* ***épineux***+*G.*

Dans cet exemple simple, l'objectif est de prédire le genre des mots marqués ce qui permet ensuite de retrouver la forme fléchie correspondante.

Pour cette tâche, il est important de considérer les dépendances locales, beaucoup d'informations morphologiques telles que le genre et le nombre se propageant naturellement aux mots voisins. Des dépendances plus longues sont néanmoins aussi importantes car l'inférence du cas par exemple se repose principalement sur les dépendances syntaxiques.

2.4 Le toit : le texte

Le texte est la structure linguistique la plus riche, on peut y trouver les dépendances les plus diverses. Une partie de ces dépendances est héritée des niveaux précédents, telles que certaines dépendances sémantiques ou même sonores dans le cas de la poésie par exemple. D'autres dépendances sont spécifiques à ce niveau supérieur, par exemple les liens causaux entre les évènements décrits, les liens de contraste [Braud, 2015]. Pour détecter de telles dépendances il peut être nécessaire de considérer des portions de texte assez éloignées, par exemple une conséquence peut être séparée de sa cause avec du texte supplémentaire.

Parmi les tâches de traitement automatique de la langue utilisant les dépendances à ce niveau on trouve par exemple la résolution de co-références [Hobbs, 1986], [Rahman and Ng, 2009], le résumé automatique [Mani, 1999], la création des systèmes de réponse automatique aux questions [Hirschman and Gaizauskas, 2001] et plus généralement de dialogue [Jurafsky and Martin, 2000], [Paek and Pieraccini, 2008].

2.4.1 Identification structurée du locuteur

Dans ce travail, une seule tâche dont les dépendances principales se manifestent au niveau du texte sera considérée : l'identification du locuteur dont la principale application est l'indexation des contenus audio et vidéo [Anguera Miro et al., 2012]. Cette tâche consiste à retrouver les tours de parole d'un dialogue et de leur attribuer un locuteur. Lorsque le texte est sous forme écrite, cette annotation peut être réalisée au niveau du mot ou de la phrase, tandis que pour un texte audio elle se fera au niveau des tours de parole.

Les systèmes état de l'art ont longtemps considéré cette tâche comme non-structurée, mais dans [Knyazeva et al., 2015a] nous avons montré que l'utilisation de l'information structurée permet d'améliorer significativement les performances. En effet, la parole multi-locuteur a une structure naturelle due au fait que les participants d'une conversation sont souvent stables. L'ensemble des locuteurs va évoluer au fil du temps lorsque de nouveaux participants entrent dans la conversation et que d'autres en sortent, mais généralement chaque groupe de locuteurs échangera au moins quelques répliques. Cette stabilité relative permet de faire l'hypothèse qu'un locuteur identifié à un moment donné à de forte chances d'être aussi présent dans les instants proches.

L'exemple suivant met en évidence l'utilité de ces dépendances. Supposons que nous ayons détecté plusieurs échanges de répliques entre deux personnages, Anna et Boris ; dans la suite de la conversation, deux répliques peuvent être attribuées à Anna encadrant une réplique très courte et donc difficile à attribuer à un personnage en particulier : *A B A B A ? A ...* On peut raisonnablement supposer que Anna et Boris sont en train de dialoguer et que cette réplique peut être attribuée à Boris.

2.5 Conclusion

Dans les sections précédentes nous avons présenté l'aspect structuré des problèmes de traitement automatique de la langue sur différentes tâches

étudiées dans ce travail. Cette présentation est très sélective ; pour une présentation plus complète et détaillée des tâches structurées on peut se référer par exemple à [Smith, 2011]. Pour conclure, nous allons discuter deux aspects généraux des problèmes structurés.

Tout d'abord, une tâche peut être « plus ou moins » structurée. Par exemple, un cas particulier de la reconnaissance de l'écriture manuscrite consiste à ne reconnaître que des nombres, une tâche utile pour reconnaître les numéros de bâtiments sur une photo. Les dépendances locales sont ici inutiles puisque les chiffres composant un numéro sont en général indépendants les uns des autres.

À l'opposé, dans le cas de la reconnaissance de mots complets mais sur un vocabulaire très réduit, chaque position décodée apporte beaucoup d'informations sur les autres positions. Plus le vocabulaire est limité et plus grand sera cet effet, une seule lettre reconnue permettant de réduire énormément le nombre de mots à considérer.

Ensuite, dans ce chapitre nous avons placé chaque tâche présentée à un niveau particulier : mot, phrase ou texte ; mais dans les faits il n'y a pas de séparation stricte entre eux. Les tâches ont été ici réparties en fonction du niveau où se trouvent les principales dépendances exploitées. En pratique, des dépendances provenant des autres niveaux peuvent être aussi exploitées.

Ainsi, dans l'exemple 2.2.1 de reconnaissance de l'écriture, une confusion entre les mots « braque » et « brague » est possible. Dans le contexte d'une phrase parlant par exemple des races de chiens, cette ambiguïté est levée et le premier deviendra nettement plus probable que le deuxième. De même, la prononciation d'un mot peut changer en fonction des mots qui l'entourent et donc nécessiter un contexte pour être prédite correctement. Par exemple, le mot russe *замок* peut signifier à la fois *chateau* et *serrure* ; l'accent tonique se plaçant sur la première ou deuxième syllabe respectivement.

Chapitre 3

Apprendre à chercher

Sommaire

Dans ce chapitre nous présentons les méthodes de la famille « apprendre à chercher », qui ont pour objectif de résoudre des problèmes de prédiction structurée. La prédiction s'effectue de manière incrémentale, en réalisant une suite de décisions ; chacune d'entre elles apporte des informations qui serviront ensuite à construire la sortie structurée. La recherche de la bonne structure (séquence, arbre, graphe ou autre) se réduit alors à la recherche d'une *séquence* de décisions la construisant.

Dans l'approche « apprendre à chercher », nous disposons d'un expert capable de réaliser le processus de construction de la sortie. L'expert résout plusieurs problèmes simples, l'un après l'autre, afin de la reconstruire. Ce sont ces problèmes qui engendrent les exemples d'apprentissage. Les problèmes rencontrés sont dépendants de la manière dont les précédents ont été résolus ; les exemples d'apprentissage ne sont donc pas supposés être indépendants, contrairement au cas de l'apprentissage supervisé.

Apprendre à imiter le comportement de l'expert, tout en prenant en compte l'interdépendance des problèmes qu'il rencontre, constitue l'objectif des méthodes « apprendre à chercher ». Le concept d'imitation du comportement de l'expert fait immédiatement appel au domaine sou-

vent identifié dans la littérature sous le nom d'« apprentissage par imitation » [Ross and Bagnell, 2010]. Néanmoins, les méthodes « apprendre à chercher » représentent un cadre légèrement différent ; nous reviendrons sur ces particularités dans la conclusion de ce chapitre.

Les débuts de cette approche peuvent être trouvés par exemple dans les travaux [Collins and Roark, 2004, Daumé III and Marcu, 2005]. Les algorithmes présentés dans ces articles sont basés sur le perceptron ; l'idée principale étant de mettre-à-jour les poids au moment où une erreur se produit dans la séquence des décisions. Différentes améliorations de ces idées ont donné lieu à de nombreux algorithmes et publications, parmi lesquelles deux que nous avons retenus pour donner une présentation détaillée de deux grandes approches existantes, [Daumé et al., 2009] et [Ross and Bagnell, 2014].

Nous commencerons par introduire dans la section 1 les notions essentielles et expliciter de manière formelle le problème de recherche de la meilleure politique imitant l'expert. La quasi totalité de ces notions est propre au domaine de l'apprentissage par renforcement [Sutton and Barto, 1998], néanmoins, pour des raisons de simplicité, nous faisons le choix d'expliciter ces notions en recourant uniquement aux bases de l'apprentissage introduites dans le chapitre 1. Les liens profonds entre les algorithmes présentés et l'apprentissage par renforcement seront établis en conclusion de ce chapitre.

Dans les sections 2 et 3 nous traiterons des approches naïves pour l'apprentissage par imitation ; cela nous permettra d'illustrer les problèmes majeurs et de motiver les approches étudiées dans les sections suivantes. Les sections 4 et 5 seront dédiées chacune à l'étude détaillée d'une des familles d'approches avancées, la famille AGGRAVATE [Ross and Bagnell, 2014] et la famille SEARN [Daumé et al., 2009]. Enfin, dans la section 6 nous ferons une brève revue de la littérature.

3.1 Définitions de base

3.1.1 Espace de recherche

Pour une entrée $x \in X$ et sa référence $y \in Y$, on définit l'espace de recherche $G(x, y) = \langle S, A, S_f, s_i, r \rangle$ [1], où

— S est l'ensemble (fini) des états possibles ;

1. Ceci est une simplification du concept de processus de décision markovien (MDP [Puterman, 1994]) dans laquelle l'état d'arrivée est une fonction déterministe de l'état de départ et de l'action effectuée.

— A est l'ensemble (fini) des actions possibles ;
— $S_f \subset S$ est l'ensemble des états finaux ;
— $s_i \in S$ est un état initial (unique pour chaque espace de recherche) ;
— $r(s, a)$ la *récompense* associée à l'action a dans l'état s.

On notera A_s l'ensemble des actions réalisables dans l'état s et $s' = s \circ a$ l'opération d'extension qui sélectionne l'action $a \in A_s$ dans l'état s afin d'atteindre l'état s'. Dans le cadre de ce travail cette fonction est supposée déterministe, ce qui signifie que la connaissance de l'état et de l'action sont suffisantes pour connaitre sans ambiguïté l'état d'arrivée. L'espace de recherche forme un graphe sans cycle à l'exception d'une boucle sur chaque état final dont la fonction sera clarifiée dans la section 3.1.4 [2].

On appelle chemin de longueur t une suite de couples (état, action) $(s_1, a_1), ..., (s_t, a_t)$ telle que $s_{t'+1} = s_{t'} \circ a_{t'}$ pour $1 \leq t' \leq t - 1$. On appelle *chemin complet*, un chemin tel que $s_1 = s_i$ et $s_t \circ a_t \in S_f$, c'est-à-dire un chemin allant de l'état initial à un état final. Dans le cas contraire, on parlera d'un *chemin partiel*.

La séquence d'actions composant un chemin complet α peut être vue comme une liste d'instructions permettant de construire la sortie structurée correspondante $\hat{y}_\alpha$. Le passage de α à $\hat{y}_\alpha$ peut être direct, chaque action correspondant au choix d'une partie de la sortie, comme dans le cas de l'étiquetage morpho-syntaxique où une action détermine généralement une étiquette, ou bien indirect comme pour l'analyse en dépendances où les actions représentent des opérations *shift* et *reduce* d'un analyseur par transition et où le lien entre chaque action et la sortie n'est pas explicite [Goldberg and Nivre, 2012].

La récompense cumulée d'un chemin α est la somme des récompenses associées aux actions effectuées le long de ce chemin :

$$R(\alpha) = \sum_{(s,a)\in\alpha} r(s, a)$$

Cette récompense cumulée correspond à la qualité de la solution obtenue en utilisant ce chemin. On a définit dans la section 1.1.1 la qualité de la sortie $\hat{y}$ à l'aide de la fonction de perte $L(y, \hat{y})$. Le lien entre la fonction de récompense et la qualité de la solution est donc le suivant : la différence entre les récompenses cumulées de deux chemins est égale à l'opposé de la différence de pertes associées aux solutions correspondantes. Plus précisément, pour deux chemins arbitraires α_1 et α_2 l'égalité suivante est vérifiée :

$$L(y, \hat{y}_{\alpha_1}) - L(y, \hat{y}_{\alpha_2}) = -(R(\alpha_1) - R(\alpha_2))$$

2. Un tel graphe est généralement appelé treillis, bien que contrairement à la définition standard il contienne plusieurs état finaux.

Il est important de noter que les récompenses locales ne peuvent pas toujours être associées directement à un fragment de la structure en construction, c'est notamment le cas lorsque le lien entre la séquence d'actions et la sortie structurée est indirect.

Dans le chapitre 1 nous avons supposé donnée la distribution $\widetilde{D}$ des couples (x, y) d'entrées et de sorties. Maintenant nous supposons donnée la distribution $\widetilde{D^+}$, qui contient, en plus des entrées et sorties, les espaces de recherche qui leur sont associés :

$$(x, y, G(x, y)) \sim \widetilde{D^+}$$

On notera T la longueur minimale telle que pour chaque espace de recherche de $\widetilde{D^+}$, tous les chemins de cette longueur ou plus soient complets.

3.1.2 Politique et notions associées

Une *politique* est une fonction déterminant l'action devant être choisie dans un état donné. Cette fonction peut être déterministe ou stochastique. Dans le premier cas, on parle de la politique déterministe $\tau : S \mapsto A$ qui est une fonction associant à chaque état $s \in S$ une action $a \in A_s$ selon une règle quelconque. Dans ce cas, l'application de la politique τ de l'état initial jusqu'à l'état final produit un chemin déterministe α_τ, qui correspond à la sortie structurée $\hat{y}_{\alpha_\tau}$.

Une politique stochastique est une fonction stochastique $\pi : S \mapsto A$ qui à chaque état $s \in S$ associe une action selon une certaine distribution de probabilité sur A_s. Dans ce cas le chemin produit ainsi que la sortie structurée sont aussi stochastiques. Nous noterons la distribution correspondante de chemins par $\widetilde{\alpha}_\pi$.

Les politiques déterministes sont des cas particuliers de politiques stochastiques.

Qualité de la politique La qualité de la politique π se détermine par la qualité de la sortie structurée qu'elle produit. Afin de mettre en évidence cette connexion, nous utiliserons la même notation pour exprimer ces deux notions. Ainsi, la qualité d'une politique déterministe est simplement la perte associée à la sortie structurée produite par cette politique :

$$L(\pi) = L(y, \hat{y}_{\alpha_\pi})$$

Pour une politique stochastique, la perte associée est la perte moyenne sur les sorties pouvant être produites :

$$L(\pi) = \mathbb{E}_{\alpha \sim \widetilde{\alpha}_\pi}[y, L(\hat{y}_\alpha)]$$

L'objectif de l'apprentissage est de trouver, à partir des données structurées étiquetées, la politique permettant d'obtenir des solutions de bonne qualité qui minimisent cette perte.

Application de politique, notions Nous avons besoin de quelques notations supplémentaires afin de manipuler les politiques. Dans ce chapitre, pour plus de généralité, on supposera que la politique π considérée est stochastique. Si la politique doit être déterministe, cela sera explicitement précisé.

On introduit les notions suivantes concernant les distributions sur les états pouvant être obtenues à l'aide d'une politique π :

- $s \circ \pi^t$: le résultat de l'application d'une suite de t actions choisies par la politique π dans l'état s ; en particulier, $s \circ \pi = s \circ \pi^1$ pour une seule application de la politique. La politique π étant supposée stochastique, ce résultat l'est aussi (c'est une distribution sur les états pouvant être obtenus). *C'est le cas de tous les objets définis dans cette section lorsqu'une politique stochastique est utilisée.*
- $s \circ \pi^*$: le résultat de l'application d'un nombre aléatoire (entre 0 et T) d'actions choisies par la politique π dans l'état s.

Ces trois distributions peuvent être généralisées au cas d'une distribution sur les états de départ. Ainsi, $\widetilde{s} \circ \pi$ est la distribution sur les états obtenue en appliquant l'action choisie par la politique π sur la distribution $\widetilde{s}$ d'états de départ, $\widetilde{s} \circ \pi^t$ la distribution sur les états obtenue en appliquant t fois la politique π sur la distribution $\widetilde{s}$ d'états de départ et $\widetilde{s} \circ \pi^*$ la distribution sur les états obtenue en appliquant un nombre aléatoire de fois la politique π sur la distribution $\widetilde{s}$ d'états de départ.

On introduit maintenant les notions suivantes concernant les chemins qui peuvent être produits à l'aide d'une politique π :

- $s \xrightarrow{\pi, t} \cdot$ le chemin de t actions construit à l'aide de la politique π à partir de l'état s.
- $s \xrightarrow{\pi, \cdot} S_f$ le chemin construit à l'aide de la politique π à partir de l'état s jusqu'à un état final. Notons que dans ce cas la longueur du chemin n'est pas fixe.

Ces deux notions peuvent elles aussi être généralisées au cas d'une distribution d'états de départ. Ainsi, $\widetilde{s} \xrightarrow{\pi, t} \cdot$ est le chemin obtenu en appliquant t fois la politique π sur la distribution $\widetilde{s}$ d'états de départ et $\widetilde{s} \xrightarrow{\pi, \cdot} S_f$ est le chemin obtenu en appliquant la politique π sur la distribution $\widetilde{s}$ d'états de départ jusqu'à un état final.

On peut également combiner différentes politiques pour construire un

chemin complexe. Par exemple,

$$s \xrightarrow{\pi_1, t_1} \cdot \xrightarrow{\pi_2, t_2} \cdot \xrightarrow{\pi_3, \cdot} S_f$$

est un chemin où les t_1 premières actions sont choisies par la politique π_1, les t_2 actions suivantes sont choisies par la politique π_2 et la suite du chemin jusqu'à l'état final est construite par la politique π_3.

Récompense cumulée et fonctions de valeur Les notions suivantes nous seront également nécessaires pour la suite :

— Nous avons introduit la notion de récompense cumulée auparavant pour un chemin déterministe. Maintenant nous introduisons la notion de récompense cumulée $R(\widetilde{\alpha})$ d'un chemin $\widetilde{\alpha}$ stochastique, qui se définit de la manière suivante :

$$R(\widetilde{\alpha}) = \mathbb{E}_{\alpha \sim \widetilde{\alpha}}[R(\alpha)]$$

Ainsi, la récompense cumulée d'un chemin stochastique, par analogie avec la perte associée à une politique stochastique, se calcule en prenant la moyenne du résultat de tous les chemins déterministes.

— La *fonction de valeur* d'un état s associée à une politique π est définie de la manière suivante :

$$V(s; \pi) = R(s \xrightarrow{\pi, \cdot} S_f)$$

Elle représente la récompense cumulée depuis l'état s jusqu'à l'état final, sachant que toutes les actions sont choisies selon la politique π. Notons que si la politique est stochastique, alors le chemin pris en argument sera lui aussi stochastique. L'extension de cette notion à une distribution sur les états nous sera également utile par la suite :

$$V(\widetilde{s}; \pi) = R(\widetilde{s} \xrightarrow{\pi, \cdot} S_f)$$

— La fonction de valeur d'un couple (état s, action $a \in A_s$) vis-à-vis de la politique π est définie de la manière suivante :

$$Q(s, a; \pi) = R(s \xrightarrow{\cdot} s \circ a \xrightarrow{\pi, \cdot} S_f)$$

Elle représente la récompense cumulée de l'état s à l'état final sachant que la première action de la séquence est a et que les actions suivantes sont choisies selon la politique π.
Il est possible d'étendre cette fonction au cas où l'état et l'action sont stochastiques. Le cas important pour nous est celui où l'action a est

donnée par une politique stochastique. Soient π_1 et π_2 deux politiques quelconques, alors la fonction de valeur d'un état s et de la politique π_1 vis-à-vis de la politique π_2 est définie de manière suivante :

$$Q(s, \pi_1; \pi_2) = R(s \xrightarrow{\pi_1, 1} \cdot \xrightarrow{\pi_2, \cdot} S_f)$$

Cette fonction peut également être étendue pour une distribution d'états :

$$Q(\widetilde{s}, \pi_1; \pi_2) = R(\widetilde{s} \xrightarrow{\pi_1, 1} \cdot \xrightarrow{\pi_2, \cdot} S_f)$$

3.1.3 Présentation de l'expert et du tuteur

Au sens large du terme, l'expert π_{exp} est une politique qui permet, lorsqu'elle est exécutée de bout en bout, de construire la meilleure solution présente dans l'espace de recherche. Notons que la référence n'est pas toujours présente dans l'espace de recherche [3]. La séquence complète d'actions de l'expert amène donc à la solution la moins éloignée de la référence selon la fonction de perte choisie. Soit $\alpha^*(G)$ l'ensemble des chemins de l'espace de recherche G, alors pour la politique de l'expert la propriété suivante est vérifiée :

$$\hat{y}_{\pi_{\text{exp}}} = \arg\min_{\alpha \in \alpha^*(G)} L(y, \hat{y}_\alpha)$$

Dans ce travail on suppose avoir accès à un expert plus puissant qui, pour une solution partielle quelconque, permet de construire la meilleure solution complète possible [4]. C'est-à-dire, le début du chemin étant construit à l'aide d'une autre politique, l'expert permet de trouver la meilleure manière de le poursuivre. Dans le cas d'une partie d'échecs, par exemple, cet expert pourra amener à la victoire, quand c'est encore possible, une partie qui a été mal commencée par un joueur non-expert.

Avec les notations introduites dans la section précédente, cet expert peut être exprimé de la manière suivante [5]

$$\pi_{\text{exp}}(s) = \arg\max_{a \in A_s} Q(s, a, \pi_{\text{exp}})$$

Ainsi, l'expert est la politique choisissant l'action impliquant la meilleure récompense cumulée sachant que toutes les actions suivantes jusqu'à l'état

3. Ce problème qui a été documenté dans [Liang et al., 2006] sera étudié de manière plus détaillée au chapitre 5.

4. Cette idée rejoint la notion d'« oracle complet » présentée dans [Goldberg and Nivre, 2012].

5. Si plusieurs actions permettent d'obtenir la valeur Q maximale, alors la politique de l'expert est non-déterministe. Dans ce chapitre nous supposons la politique de l'expert déterministe ; le cas non-déterministe sera considéré dans le chapitre suivant.

final seront aussi choisies par la politique expert. Notons que cette politique correspond à la mise en œuvre d'une politique gloutonne dont les scores sont définis par la fonction de valeur. Si l'on a accès à la fonction de valeur pour chaque action, la recherche gloutonne permet d'obtenir le meilleur chemin de manière exacte.

Certains algorithmes présentés dans ce chapitre nécessitent un expert qui soit capable d'agir de la meilleure manière possible étant donné non seulement le début du chemin, mais aussi en sachant que la suite du chemin après son intervention sera construite par une autre politique. On appelle un tel expert un *tuteur*. Pour une partie d'échecs, le tuteur serait capable de conseiller un joueur pour son prochain coup, en prenant en compte son niveau. Il évitera de conseiller une action qui serait gagnante, mais seulement pour un joueur d'un meilleur niveau, et conseillera plutôt une stratégie que le joueur pourra poursuivre seul.

Formellement, la politique du tuteur peut être exprimée de la manière suivante :

$$\pi_{\text{tut}}(s, \pi') = \arg\max_{a \in A_s} Q(s, a, \pi')$$

Ainsi, le tuteur est la politique choisissant l'action impliquant la meilleure récompense cumulée sachant que toutes les actions suivantes jusqu'à l'état final seront choisies par la politique π'. Autrement dit, le tuteur permet d'apporter la meilleure amélioration *locale* à une politique donnée.

Notons que la politique expert est un cas particulier simple de la politique tuteur ; en pratique il est généralement beaucoup plus difficile d'avoir accès à la politique tuteur qu'à la politique expert. La politique expert dépend uniquement de l'espace de recherche et peut être calculée une seule fois pour chaque état. Par exemple, dans le cas de l'étiquetage morpho-syntaxique, cette politique correspond simplement au choix de l'étiquette de référence pour chaque position indépendamment des autres positions ; son calcul est donc immédiat. Dans d'autres cas, tels que pour l'analyse en dépendances par exemple, ce calcul peut être plus astucieux [Goldberg and Nivre, 2012]. La politique tuteur demande en plus la prise en compte d'une l'autre politique π'. Cela nécessite de réaliser les calculs pour chaque politique π', et peut de plus être problématique en pratique dans le cas où la politique π' est stochastique [Daumé et al., 2009] par exemple.

Dans ce travail, nous faisons une autre supposition concernant les politiques expert et tuteur que simplement avoir accès à leur choix : l'expert ou le tuteur peuvent indiquer quelle est la perte immédiate apportée par une action par rapport à l'action proposée. Cela signifie qu'il est possible

de calculer *la perte de potentiel* :

$$l(s, a, \pi') = \max_{a' \in A_s} Q(s, a', \pi') - Q(s, a, \pi')$$

pour tout état $s \in S$ et pour toute action $a \in A_s$, avec $\pi' = \pi_{\text{exp}}$ pour l'expert et π' une politique arbitraire pour le tuteur.

Comme pour les fonctions V et Q, nous introduisons l'extension de la perte de potentiel au cas où l'état et l'action sont stochastiques. Nous supposons que l'action stochastique est donnée par une politique stochastique π_1. Dans ce cas :

$$l(s, \pi_1, \pi_2) = \max_{a' \in A_s} Q(s, a', \pi_2) - Q(s, \pi_1, \pi_2)$$

Dans le cas où l'état de départ est représenté par une distribution $\widetilde{s}$ on obtient :

$$l(\widetilde{s}, \pi_1, \pi_2) = \mathbb{E}_{s \sim \widetilde{s}} \left[\max_{a' \in A_s} Q(s, a', \pi_2) - Q(s, \pi_1, \pi_2) \right]$$

La perte de potentiel vis-à-vis d'une politique définit la perte immédiate par rapport à la meilleure action accessible. Par la suite il nous sera utile de définir la perte non seulement par rapport à la meilleure action possible, mais aussi par rapport à l'action choisi par une autre politique. Nous aurons notamment besoin du cas où le chemin est obtenu en utilisant la politique π_1 à l'exception d'une position où la politique π_2 est utilisée, par rapport au cas où la politique π_1 construit le chemin entier. Cela représente la perte locale liée à l'utilisation de la politique π_2 une seule fois par rapport à l'utilisation de π_1 :

$$l_{\text{loc}}(\pi_2; \pi_1) = V(\widetilde{s}_i; \pi_1) - Q(\widetilde{s}_i \circ \pi_1^*, \pi_2; \pi_1)$$

Lorsque la position d'application de la politique π_2 est fixe, nous utilisons la notation suivante :

$$l_{\text{loc}}^k(\pi_2; \pi_1) = V(\widetilde{s}_i; \pi_1) - Q(\widetilde{s}_i \circ \pi_1^{k-1}, \pi_2; \pi_1)$$

3.1.4 Fonction objectif

Comme nous l'avons vu dans la section précédente, suivre la politique expert sur la trajectoire complète permet d'obtenir la solution impliquant la perte minimale parmi celles qui sont présentes dans l'espace de recherche. Néanmoins, dans les applications pratiques que l'on va considérer dans les

chapitres 4 et 5 il est rarement possible d'imiter parfaitement la politique expert. Notre objectif est alors de trouver la politique π permettant de construire une solution qui aurait la perte $L(\pi)$ la plus petite possible.

Il est possible de représenter la fonction objectif de la manière suivante :

$$\begin{aligned}
L(\pi) &= L(\pi_{\text{exp}}) + \big[V(\widetilde{s}_i, \pi) - V(\widetilde{s}_i, \pi_{\text{exp}})\big] \\
&= L(\pi_{\text{exp}}) + \sum_{t=0}^{T-1} \big[R(\widetilde{s}_i \xrightarrow{\pi, t+1} \cdot \xrightarrow{\pi_{\text{exp}}, \cdot} S_f) - R(\widetilde{s}_i \xrightarrow{\pi, t} \cdot \xrightarrow{\pi_{\text{exp}}, \cdot} S_f)\big] \\
&= L(\pi_{\text{exp}}) + \sum_{t=0}^{T-1} \big[R(\widetilde{s}_i \circ \pi^t \xrightarrow{\pi} \cdot \xrightarrow{\pi_{\text{exp}}, \cdot} S_f) - R(\widetilde{s}_i \circ \pi^t \xrightarrow{\pi_{\text{exp}}, \cdot} S_f)\big] \\
&= L(\pi_{\text{exp}}) + \sum_{t=0}^{T-1} \big[Q(\widetilde{s}_i \circ \pi^t, \pi; \pi_{\text{exp}}) - V(\widetilde{s}_i \circ \pi^t; \pi_{\text{exp}})\big] \\
&= L(\pi_{\text{exp}}) + \sum_{t=0}^{T} l(\widetilde{s}_i \circ \pi^t, \pi; \pi_{\text{exp}}) \\
&= L(\pi_{\text{exp}}) + T \cdot l(\widetilde{s}_i \circ \pi^*, \pi; \pi_{\text{exp}})
\end{aligned} \tag{3.1}$$

Notons que grâce à la boucle présente sur chaque état final nous assurons la possibilité de prolonger chaque chemin jusqu'à la longueur T même si l'état final est atteint plus tôt ; cela nous permet de considérer uniquement les chemins de cette longueur. Pour des raisons de simplicité nous utiliserons cette technique de manière systématique dans la suite de ce chapitre.

Cette représentation montre que la perte totale se décompose en une perte associée à la politique expert π_{exp}, la référence n'étant pas toujours atteignable dans l'espace de recherche, et en des pertes de potentiel par rapport à la politique expert associée à chaque action réalisée par la politique π. La perte associée à la politique expert est fixe, elle ne dépend pas de la politique π. Pour minimiser la perte totale il suffit donc de minimiser la perte moyenne de potentiel par rapport à la politique expert

$$\arg\min_{\pi \in \Pi} L(\pi) = \arg\min_{\pi \in \Pi} l(\widetilde{s}_i \circ \pi^*, \pi; \pi_{\text{exp}}) \tag{3.2}$$

Dans les sections suivantes nous allons étudier les difficultés de l'optimisation de (3.2), ainsi que les méthodes approximatives proposées dans la littérature.

3.2 Méthodes non-itératives

Le problème (3.2) est difficile à résoudre en raison de l'aspect cyclique de la fonction considérée : pour trouver la politique qui aura une perte de potentiel moyenne minimale, il est nécessaire de connaitre la distribution des états pour laquelle la perte moyenne sera calculée, et pour trouver cette distribution il est nécessaire de connaitre la politique.

Il est possible de retrouver une fonction facilement optimisable en sélectionnant une distribution fixe (c'est-à-dire indépendante de la politique inconnue) d'états $\widetilde{s}_{\text{fix}}$ pour remplacer la distribution $\widetilde{s_i} \circ \pi^*$. Cela signifie minimiser l'objectif suivant par rapport à la politique π :

$$l(\widetilde{s}_{\text{fix}}, \pi; \pi_{\text{exp}}) \tag{3.3}$$

Avec un choix raisonnable pour la distribution $\widetilde{s}_{\text{fix}}$ cette fonction, on l'espère, approxime correctement l'objectif $l(\widetilde{s_i} \circ \pi^*, \pi; \pi_{\text{exp}})$.

L'optimisation de (3.3) est un problème standard de classification multi-classe sensible aux coûts. Il nécessite un ensemble d'apprentissage de la forme suivante :

$$\cup_{k=0}^{K} (o_k, P_k, C_k)$$

où K est la taille de l'ensemble d'apprentissage, P_k est un ensemble de prédictions possibles pour l'observation o_k, et C_k est un ensemble de coûts associés aux prédictions. Dans notre cas, pour un état s_k représenté par l'observation o_k, $P_k = A_s(s_k)$ est l'ensemble d'actions accessibles dans cet état et les coûts C_k correspondent aux pertes de potentiel $l(s_k, a; \pi_{\text{exp}})$ de ces actions $a \in A_s(s_k)$ vis-à-vis de l'expert. Moins formellement, nous appelons les triplets (état, actions accessibles, pertes associées) des *situations* auxquelles une politique peut être confrontée, dans le sens où ces triplets représentent l'ensemble des informations à partir desquelles la politique doit faire le choix d'une action.

La construction de cet ensemble nécessite un simulateur de la distribution $\widetilde{s}_{\text{fix}}$, ainsi qu'un calculateur des coûts $l(s, a; \pi_{\text{exp}})$. Cet ensemble d'apprentissage permet de trouver la politique π minimisant (3.3) [6].

Nous commençons par observer ce qui se passe pour deux distributions $\widetilde{s}_{\text{fix}}$ simples à définir :

- apprendre sur la trajectoire expert (seuls les états de la trajectoire expert sont utilisés pour l'apprentissage)
- apprendre sur l'espace complet (tous les états de l'espace de recherche sont considérés)

6. Nous supposons dans ce chapitre théorique que l'erreur liée à la taille limitée de l'ensemble d'apprentissage, ainsi qu'à l'estimation non-exacte des coûts, est négligeable.

Ces approximations nous permettront de mettre en évidence les deux difficultés importantes que nous rencontrerons aussi dans les autres approches : l'*effet d'avalanche* et le *gaspillage de ressources*.

3.2.1 Apprendre sur la trajectoire de l'expert : l'effet d'avalanche

L'apprentissage sur la trajectoire expert a pour objectif d'apprendre à résoudre les problèmes comme l'expert en traitant les situations qu'il rencontre. Cela correspond à optimiser la fonction objectif suivante :

$$l(\widetilde{s}_i \circ \pi^*_{\text{exp}}, \pi; \pi_{\text{exp}}) \tag{3.4}$$

De manière informelle, l'effet d'avalanche est un problème qui se manifeste au moment de l'inférence et qui est lié au fait que le modèle n'étant pas parfait, il finira par commettre une erreur. Lorsque la séquence prédite s'écarte de la trajectoire de l'expert, l'action pour laquelle une erreur est commise va nous amener dans un état qui n'a pas été observé lors de l'apprentissage. Le manque d'expérience dans ce nouvel état va fortement augmenter le risque qu'une nouvelle erreur soit commise ce qui déclenche une réaction en chaine.

Plus précisément, cette situation correspond à une situation où la distribution des états visités lors de l'inférence diffère de celle qui a été vue lors de l'apprentissage ($\widetilde{s}_i \circ \pi_{\text{exp}}$). Ce problème est aussi parfois appelé « exposure bias » [Ranzato et al., 2016] ou encore « propagation d'erreur » [Le and Fokkens, 2017] dans la littérature. La perte moyenne de potentiel (3.4) obtenue en apprentissage donne alors une mauvaise estimation de la perte totale $L(\pi)$. Une borne possible pour la perte totale est la suivante [Ross and Bagnell, 2010] :

$$L(\pi) \leq L(\pi_{\text{exp}}) + p_{\text{err}} \cdot T \cdot L_{\max} \tag{3.5}$$

où p_{err} est la probabilité de commettre une erreur (c'est à dire de choisir une action différente de celle proposée par l'expert) dans un état *sur la trajectoire optimale*. Cette estimation signifie que la perte obtenue dans le cas d'une erreur reste bornée par la perte maximale même si cette erreur implique une perte de potentiel très petite, car l'état d'arrivée de l'action choisie représente une situation inconnue pour la politique apprise. Cette garantie est bien sûr insatisfaisante.

Notons que l'apprentissage sur n'importe quel ensemble qui ne contient pas tous les états visités lors de l'inférence risque, dans les faits, de donner lieu à cet effet d'avalanche.

3.2.2 Apprendre sur l'espace complet : le gaspillage de ressources

Dans cette deuxième stratégie naïve, l'ensemble d'apprentissage est composé de l'intégralité des états de l'espace de recherche. Soit $\widetilde{s}_{\text{uni}}$ la distribution qui attribue la même probabilité à tous les états d'un même espace de recherche. Supposons que l'on possède un générateur de cette distribution et que l'on soit capable de calculer le coût (la perte de potentiel vis-à-vis de l'expert) de chaque action de l'espace de recherche. On peut alors trouver la politique qui minimise la perte de potentiel correspondante :

$$l(\widetilde{s}_{\text{uni}}, \pi; \pi_{\text{exp}}) \tag{3.6}$$

Au premier abord, cette stratégie consistant à inclure des exemples de toutes les situations pouvant se présenter lors de l'inférence peut paraître raisonnable mais elle soulève deux problèmes. Tout d'abord un problème calculatoire, car le nombre d'exemples nécessaire pour approximer correctement la distribution $\widetilde{s}_{\text{uni}}$ peut être gigantesque.

Ensuite, même dans le cas où cette stratégie est techniquement réalisable, son efficacité n'est pas absolue car, comme pour la méthode précédente, la perte (3.6) ne reflète pas correctement la performance moyenne sur une trajectoire choisie par la politique π. Pour obtenir une garantie théorique, une possibilité est d'utiliser sa borne supérieure $\sup_{s\sim\widetilde{s}_{\text{uni}}} [l(s, \pi; \pi_{\text{exp}}]$:

$$L(\pi) \leq L(\pi_{\text{exp}}) + \sup_{s\sim\widetilde{s}_{\text{uni}}} [l(s, \pi; \pi_{\text{exp}}] \cdot T \tag{3.7}$$

Intuitivement, le deuxième problème est lié au fait que dans un cadre d'apprentissage automatique tel que le notre, les capacités du système sont limitées. C'est le cœur de la problématique d'optimisation : utiliser les ressources limitées dont on dispose afin d'apprendre correctement les phénomènes les plus pertinents plutôt que d'apprendre de manière peu fiable un peu tout ce qui est possible. Si tous les états possibles sont considérés au moment de l'apprentissage, une partie des ressources limitées va être consacrée à modéliser des phénomènes très éloignés du chemin suivi par l'expert et qui donc, au moment de l'inférence, ne seront utiles que dans des situations déjà désespérées.

Notons que l'apprentissage sur n'importe quel ensemble contenant des états autres que ceux rencontrés en pratique va dans le cas général donner lieu aux difficultés liées à ce phénomène que nous appelons dans ce travail le gaspillage de ressources.

3.3 Itération de politique

La notion d'itération de politique est à la base de toutes les méthodes d'apprentissage que nous allons présenter dans le reste de ce chapitre. Dans cette section, après avoir présenté les notions et algorithmes de base nécessaires, nous détaillerons l'implémentation la plus naïve de la méthode d'itération de politique.

3.3.1 Notions de base

Dans la section précédente nous avons posé le principe général de l'apprentissage d'une bonne politique : collecter un ensemble de situations où la politique doit faire un choix (décrites par l'état et les actions accessibles) et indiquer si le comportement est désiré ou non à l'aide des coûts.

Nous avons ensuite remarqué que la composition de cet ensemble de situations a une grande importance : il est inutile de gaspiller des ressources en apprenant les situations que l'on ne rencontrera pas en pratique tout comme il est dangereux de se retrouver dans une situation que l'on n'a pas étudié pendant l'apprentissage. Dans cette section nous allons définir d'autres manières de construire l'ensemble de situations à apprendre en utilisant une politique autre que celle de l'expert [7].

Supposons donnée une politique π de l'ensemble paramétrique Π dans lequel on cherche la meilleure politique. Notre premier problème d'apprentissage est le suivant :

Définition Le *problème d'apprentissage avec l'expert* engendré par la politique π noté $\mathrm{ExpProb}(\pi)$ est composé des états s distribués selon $\widetilde{s_i} \circ \pi^*$ et des actions $a \in A_s$ associées aux coûts $l(s, a; \pi_{\mathrm{exp}})$.

Ce problème d'apprentissage permet donc d'apprendre à imiter l'expert dans les situations rencontrées par la politique π. La politique π_{res} obtenue en résolvant le problème d'apprentissage $\mathrm{ExpProb}(\pi)$ a la propriété suivante :

$$\pi_{\mathrm{res}} = \arg\min_{\pi' \in \Pi} l(\widetilde{s_i} \circ \pi^*,\ \pi';\ \pi_{\mathrm{exp}})$$

7. Les concepts introduits dans cette section ne sont pas nouveaux ; les différentes manières de réduire l'apprentissage de la politique à l'apprentissage supervisé existent dans la littérature (par exemple, [Ross and Bagnell, 2010]). Ici nous donnons juste un nom à des concepts qui n'étaient pas, à notre connaissance, explicitement nommés. L'introduction de ces définitions nous permettra de mettre en avant les grandes lignes communes des différents algorithmes.

Pour certains algorithmes de ce chapitre nous aurons besoin d'une variante de ce problème :

Définition Le *problème d'apprentissage avec l'expert à la position t* engendré par la politique π noté $\text{ExpProb}^t(\pi)$ est composé des états s distribués selon $\widetilde{s_i} \circ \pi^t$ et des actions $a \in A_s$ associées aux coûts $l(s, a; \pi_{\text{exp}})$.

La différence par rapport à la définition précédente est donc la distribution des états : dans le cas de $\text{ExpProb}(\pi)$ ce sont tous les états atteignables avec la politique π, et dans le cas de $\text{ExpProb}^t(\pi)$ les états considérés se trouvent à distance t de l'état initial. Par analogie avec le cas précédent, la politique π_{res} obtenue en résolvant le problème d'apprentissage $\text{ExpProb}^t(\pi)$ a la propriété suivante :

$$\pi_{\text{res}} = \arg\min_{\pi' \in \Pi} l(\widetilde{s_i} \circ \pi^t,\ \pi';\ \pi_{\text{exp}})$$

L'utilisation de l'expert pour déterminer les coûts n'est généralement pas la solution optimale. En effet, celui-ci va privilégier les actions qui amènent à la perte globale minimum en supposant que toutes les actions suivantes seront optimales. En pratique, suivre les choix de l'expert peut amener dans des états particulièrement difficiles pour la politique courante qui sera utilisée pour le reste du décodage.

Les coûts optimaux que l'on souhaiterait pouvoir utiliser sont ceux définis par la politique qui sera obtenue en résolvant le problème d'apprentissage. En effet, ceux-ci représentent parfaitement le potentiel de chaque action. Si à chaque itération la nouvelle politique change peu, la politique courante en est une bonne approximation et peut être utilisée pour définir les scores des actions des exemples :

Définition Le *problème d'apprentissage avec le tuteur* engendré par la politique π noté $\text{TutProb}(\pi)$ est composé des états s distribués selon $\widetilde{s_i} \circ \pi^t$ et des actions $a \in A_s$ associées aux coûts $l(s, a; \pi)$.

La politique π_{res} obtenue en résolvant le problème d'apprentissage $\text{TutProb}(\pi)$ a la propriété suivante :

$$\pi_{\text{res}} = \arg\min_{\pi' \in \Pi} l(\widetilde{s_i} \circ \pi^*, \pi', \pi)$$

Cette fois-ci, la politique π_{opt} minimise la perte de potentiel vis-à-vis de la politique π sur la distribution d'états engendrée par cette politique.

Remarquons que l'estimation de la perte vis-à-vis de la politique π est généralement beaucoup plus compliquée à réaliser d'un point de vue technique. Les facteurs qui en font une tâche compliquée sont par exemple le fait

que la politique est stochastique ou la longueur importante de la séquence d'actions.

Par analogie avec le problème d'apprentissage avec l'expert, le problème d'apprentissage avec le tuteur peut être défini pour une position en particulier plutôt que pour le chemin complet de la politique :

Définition Le *problème d'apprentissage avec le tuteur à la position t* engendré par la politique π noté $\text{TutProb}^t(\pi)$ est composé des états s distribués selon $\widetilde{s_i} \circ \pi^t$ et des actions $a \in A_s$ associées aux coûts $l(s, a; \pi)$.

Les conditions pratiques permettant de résoudre ce problème restent les mêmes que celle du problème précédent.

3.3.2 Algorithme ITERATE

Dans la section 3.2 nous avons vu que la réalisation de l'apprentissage sur une distribution d'états différente de celle obtenue lors de l'inférence implique certaines difficultés. L'objectif est de créer des conditions d'apprentissage aussi proches que possible de celles de l'inférence.

La première approche naïve que nous proposons à titre d'illustration se base sur un processus itératif utilisant la politique apprise lors de l'itération précédente pour produire les exemples d'apprentissage pour l'itération courante. Il est naturel de supposer que l'utilisation de la politique apprise permettra de créer un ensemble de situations d'apprentissage plus proche des conditions de l'inférence que si cela était réalisé à l'aide d'une politique expert non-atteignable.

La politique peut être utilisée afin de construire un problème d'apprentissage avec l'expert ou avec le tuteur. Ces deux approches sont présentées dans l'algorithme 4.

Algorithme 4 : ITERATE : algorithme illustratif. *Prob représente ExpProb ou TutProb.

Données : données d'apprentissage représentant $\widetilde{D^+}$, ensemble paramétrique Π

Initialisation : $\pi_1 \leftarrow \pi_{\text{exp}}$;
pour $n = 1..N$ **faire**
    Construire le nouveau problème d'apprentissage $^*\text{Prob}(\pi_n)$;
    Apprendre $\pi_{i+1} \in \Pi$ résolvant $^*\text{Prob}(\pi_n)$;
fin
retourner π_N;

Ce premier algorithme illustre les principes généraux des approches étudiées dans ce chapitre ; il n'offre pas de garantie sur le niveau des performances de la politique obtenue. En réalité, les performances de la politique peuvent évoluer de manière imprévisible d'une itération à l'autre. En effet, rien ne garantit que l'ensemble d'apprentissage formé à l'aide d'une politique reste pertinent à l'itération suivante.

Intuitivement, l'absence de garantie s'explique par le fait qu'à chaque itération, une politique est apprise de manière à ne plus faire les erreurs de la politique précédente. Les exemples d'erreurs vont donc disparaitre et les politiques suivantes sont donc condamnées à refaire ces erreurs à nouveau.

Dans les deux sections suivantes nous allons présenter deux familles d'algorithmes plus avancés reposant sur ce processus itératif : la famille AGGRAVATE qui repose sur le problème d'apprentissage ExpProb, et la famille SEARN reposant sur le problème d'apprentissage TutProb.

3.4 Famille AGGRAVATE

Nous avons vu que l'algorithme ITERATE présente le problème que la performance de la politique obtenue d'une itération à l'autre peut varier de manière imprévisible. Les algorithmes que nous réunissons dans la famille AGGRAVATE[Ross and Bagnell, 2014] tentent de résoudre ce problème en utilisant une stratégie différente de génération de problème d'apprentissage avec l'expert. Chaque nouvelle politique est apprise non pas uniquement sur le nouvel ensemble d'apprentissage mais sur l'accumulation des exemples collectés depuis le début de la procédure d'entrainement.

Au contraire d'ITERATE qui construit un nouveau problème d'apprentissage à chaque itération, la philosophie de cette approche est d'apprendre à agir dans de nouvelles situations tout en préservant les connaissances accumulées lors des itérations précédentes. Asymptotiquement, cela permet de trouver une politique qui est préparée à l'ensemble des situations qu'elle peut potentiellement rencontrer. Par contre, l'ensemble d'apprentissage contient plus de situations à apprendre que celles qui seront observées en pratique ; on retrouve donc le problème du gaspillage de ressources ici. Mais, contrairement à l'approche consistant à imiter l'expert sur l'ensemble des états possibles, les exemples qui sont accumulés ici sont produits par de vraies politiques apprises sur les données d'apprentissage et contiennent, on l'espère, des situations pertinentes.

3.4.1 Description de l'algorithme

Nous commençons par présenter une version simple de l'algorithme AGGRAVATE [Ross and Bagnell, 2014] dont le principe est le suivant : en partant d'une politique arbitraire, à chaque nouvelle itération un nouveau problème d'apprentissage expert engendré par cette politique est construit puis réuni avec celui de l'itération précédente, une nouvelle politique est ensuite apprise. La politique finale est choisie à l'aide des données de validation parmi les politiques obtenues à chaque itération.

Notre simplification par rapport à la version originale de AGGRAVATE consiste à enlever l'option de mélange de la politique apprise avec celle de l'expert au cours des premières itérations. Nous le faisons afin de mettre en avant le mécanisme de recours à l'optimisation convexe en ligne [Shalev-Shwartz, 2012] ; la démonstration présentée dans la section suivante diffère de celle de [Ross and Bagnell, 2014] uniquement par quelques détails techniques qui ne sont pas à notre avis essentiels dans la compréhension du fonctionnement de l'algorithme. La version originelle de l'algorithme [Ross and Bagnell, 2014] sera présentée dans la section 3.4.3.

Algorithme 5 : Agrégation des données d'apprentissage

Données : données d'apprentissage et de validation représentant $\widetilde{D^+}$, ensemble paramétrique Π

Initialisation : $\pi_0 \leftarrow$ politique arbitraire de Π, $\mathcal{E} \leftarrow \emptyset$;
pour $n = 1..N$ **faire**
    Construire le problème d'apprentissage ExpProb(π_{n-1});
    Réunir les exemples d'apprentissage $\mathcal{E} = \mathcal{E} \cup$ ExpProb(π_{n-1});
    Apprendre π_n résolvant $\mathcal{E}$;
fin
En utilisant les données de validation, parmi $\{\pi_n\}_{n=1}^N$ choisir π_{res};
retourner π_{res};

3.4.2 Analyse

Théorème 3.4.1. *AGGRAVATE construit une suite de politiques* $\{\pi_n\}_{n=1}^N$ *telle que, pour au moins une politiques* π_{res} *on ait :*

$$L(\pi_{\text{res}}) \leq Ł_{\text{exp}} + T \cdot \min_{\pi \in \Pi} \frac{1}{N} \sum_{n=1}^{N} l(\widetilde{s_i} \circ \pi_n, \pi, \pi_{\text{exp}}) + O\left(\frac{1}{N}\right) \tag{3.8}$$

La meilleure politique π_{res} *peut ensuite être sélectionnée à l'aide de l'ensemble de validation.*

Démonstration. (Cette démonstration est une simplification de la démonstration donnée dans [Ross and Bagnell, 2014].) La justification de ce résultat repose sur la théorie de l'optimisation convexe en ligne [Shalev-Shwartz, 2012], dont le principe est le suivant : l'apprentissage se déroule en une suite d'itérations ; lors de l'itération t la politique actuelle π_t est transmise à l'environnement, qui répond avec une fonction de perte convexe $\breve{f}_t$ (potentiellement différente à chaque itération). Chaque fonction de perte $\breve{f}_t : \Pi \to \mathbb{R}$ associe une perte à chaque politique de l'ensemble considéré. Le système obtient donc la perte $\breve{f}_t(\pi_t)$ correspondante à la valeur de cette fonction pour sa politique actuelle, et met à jour la politique en prenant en compte la nouvelle fonction de perte. L'objectif de l'apprentissage est de proposer une *suite de politiques* (contrairement à l'apprentissage hors-ligne qui recherche une seule politique) qui minimise la perte cumulée lors des itérations $\sum_{n=1}^{N} \breve{f}_n(\pi_n)$.

Les algorithmes de la famille *sans regret* garantissent construire dans ces conditions une suite de politiques π_1, π_2, ..., π_N ayant la propriété suivante :

$$\frac{1}{N}\sum_{n=1}^{N} \breve{f}_n(\pi_n) \leq \min_{\pi\in\Pi} \frac{1}{N}\sum_{n=1}^{N} \breve{f}_n(\pi) + O\left(\frac{1}{N}\right) \tag{3.9}$$

La quantité $\frac{1}{N}\sum_{n=1}^{N} \breve{f}_n(\pi_n) - \min_{\pi\in\Pi} \frac{1}{N}\sum_{n=1}^{N} \breve{f}_n(\pi)$ représente le regret associé à la suite de politiques choisie par rapport à la meilleure politique de Π. Notamment, l'algorithme *suivre le leader* (follow the leader) [Shalev-Shwartz, 2012] utilise la règle suivante de mise-à-jour de la politique :

$$\pi_{n+1} = \arg\min_{\pi\in\Pi} \sum_{i=1}^{n} \breve{f}_i$$

qui correspond à la recherche de la politique minimisant chaque fonction de perte reçue depuis la première itération.

Nous avons vu dans la section précédente que le problème d'apprentissage ExpProb(π) permet d'obtenir la politique π' minimisant $l(\widetilde{s}_i \circ \pi^*, \pi'; \pi_{\text{exp}})$. Nous pouvons maintenant remarquer que AGGRAVATE est un cas particulier de l'algorithme *suivre le leader* utilisant la fonction de perte suivante :

$$\breve{f}_n(\pi) = l(\widetilde{s}_i \circ \pi_n, \pi, \pi_{\text{exp}}) \tag{3.10}$$

Avec l'utilisation de (3.10) l'inégalité (3.9) se transforme de la manière suivante :

$$\frac{1}{N}\sum_{n=1}^{N} l(\widetilde{s}_i \circ \pi_n^*, \pi_n, \pi_{\text{exp}}) \leq \min_{\pi\in\Pi} \frac{1}{N}\sum_{n=1}^{N} l(\widetilde{s}_i \circ \pi_n^*, \pi, \pi_{\text{exp}}) + O\left(\frac{1}{N}\right) \tag{3.11}$$

La moyenne étant supérieure au minimum, on en déduit qu'il existe une politique π_k telle que

$$l(\widetilde{s}_i \circ \pi_k^*, \pi_k, \pi_{\text{exp}}) \leq \min_{\pi \in \Pi} \frac{1}{N} \sum_{n=1}^{N} l(\widetilde{s}_i \circ \pi_n^*, \pi, \pi_{\text{exp}}) + O\left(\frac{1}{N}\right) \tag{3.12}$$

Le résultat (3.8) s'obtient en utilisant la représentation (3.1) de la perte totale. □

Les performances de la politique obtenue sont donc asymptotiquement égales aux performances (en termes de minimisation de perte de potentiel) de la *meilleure* politique pour un ensemble de trajectoires accumulées au cours de l'apprentissage. Par contre, le fait de minimiser $\sum_{n=1}^{N} l(\widetilde{s}_i \circ \pi_n^*, \pi, \pi_{\text{exp}})$ plutôt que $l(\widetilde{s}_i \circ \pi^*, \pi, \pi_{\text{exp}})$ soulève le problème du gaspillage de ressources. Autrement dit, rien ne garantit la qualité de l'ensemble de trajectoires considéré

$$\cup_{n=1}^{N} \widetilde{s}_i \circ \pi_n^* \tag{3.13}$$

ni la bonne répartition des ressources d'apprentissage entre les trajectoires plus et moins pertinentes. Ce n'est donc pas une garantie d'optimalité.

En pratique, on peut raisonnablement supposer que l'apprentissage sur l'ensemble de trajectoires (3.13) est plus pertinent que l'utilisation de l'ensemble de tous les états possibles décrite la section 3.2.2, car il s'agit d'un ensemble de vraies trajectoires sélectionnées par des politiques de Π.

Par contre, la politique initiale peut être de très mauvaise qualité, ce qui entraine un apprentissage dans des zones désespérées, et rien ne garantit qu'il soit possible de sortir de cette zone. Dans la section suivante nous allons présenter une heuristique qui permet de modifier l'ensemble (3.13) de manière à contenir de meilleures trajectoires.

3.4.3 S'éloigner progressivement de l'expert

Un moyen d'assurer que l'ensemble d'apprentissage soit constitué, au moins en partie, de trajectoires que l'on souhaite pouvoir reproduire est d'ajouter une opération supplémentaire à l'algorithme 5. Cette étape réalise un mélange stochastique entre la solution du problème d'apprentissage $\pi'_n = \arg\min_{\pi \in \Pi} \text{ExpProb}(\pi_{n-1})$ et l'expert π_{exp} afin de constituer la politique π_n :

$$\pi_n(s) = \begin{cases} \pi_{\text{exp}}(s) & \text{avec une probabilité } \beta_n \\ \pi'_n(s) & \text{avec une probabilité } 1 - \beta_n \end{cases} \tag{3.14}$$

L'algorithme 6 représente la méthode AGGRAVATE incluant la modification (3.14). Notons que dans [Ross and Bagnell, 2014] l'algorithme AGGRAVATE est présenté directement de cette manière.

Algorithme 6 : AGGRAVATE

Données : données d'apprentissage et de validation représentant $\widetilde{D^+}$, ensemble paramétrique Π, nombre d'itérations N, suite $\{\beta_n\}_{n=1}^N$

Initialisation : $\pi_0 \leftarrow$ politique arbitraire, $\mathcal{E} \leftarrow \emptyset$;
pour $n = 1..N$ **faire**
  Construire le problème d'apprentissage ExpProb(π_{n-1});
  Réunir les exemples d'apprentissage $\mathcal{E} = \mathcal{E} \cup \text{ExpProb}(\pi_{n-1})$;
  Apprendre τ_n résolvant $\mathcal{E}$;
  $\pi_n = \beta_n \pi_{\text{exp}} + (1 - \beta_n)\tau_n$;
fin
En utilisant les données de validation, parmi $\{\tau_n\}_{n=1}^N$ choisir π_{res};
retourner π_{res};

Suite à l'introduction de (3.14) il est toujours possible de garantir le respect du théorème 3.4.1. Pour cela, la suite $\{\beta_n\}_{n=1}^N$ doit remplir la condition suivante : $\sum_{n=1}^N \beta_n < \infty$ (convergence de la série). Si en théorie les garanties sont les mêmes que pour la version de base d'AGGRAVATE, en pratique on espère obtenir un ensemble d'exemples accumulés de meilleure qualité, et donc une borne (3.8) plus intéressante.

Ceci est dû au fait que l'on cherche à rester plus longtemps avec une politique de bonne qualité (où une partie d'actions est choisie par l'expert) le temps de pouvoir apprendre suffisamment. Le choix habituel est de créer le premier problème d'apprentissage entièrement en utilisant l'expert (ce qui correspond au cas $\beta_1 = 1$). Le passage à $\beta_n = 0$ est réalisé progressivement à mesure que la qualité de la politique s'améliore.

3.4.4 Conclusion

Le principe de réduction du problème de recherche d'une bonne politique vers l'apprentissage en ligne a été initialement proposé dans [Ross et al., 2011] avec DAGGER, l'ancêtre d'AGGRAVATE. DAGGER repose uniquement sur un retour binaire de l'expert plutôt que sur des coûts réels, ce qui est plus simple à obtenir en pratique mais apporte une borne sur la performance de la politique moins précise. Notons également le récent algorithme LOLS [Chang et al., 2015] qui utilise le problème d'apprentissage

avec un tuteur (ou un mélange stochastique de l'expert et du tuteur) dans le cas où l'expert est difficilement accessible (ou la politique de référence qui l'approxime est de mauvaise qualité).

3.5 Famille SEARN

L'idée générale des algorithmes présentés dans cette section est d'utiliser le problème d'apprentissage avec le tuteur de manière à effectuer des changements locaux dans la politique expert en s'en éloignant à une vitesse contrôlée. Cela permet de garantir que la qualité de la politique obtenue est raisonnablement dégradée par rapport à l'expert. Nous avons appelé cette famille SEARN du nom de l'algorithme de cette famille probablement le mieux connu de la communauté du TAL. Avant de présenter SEARN et ses garanties, nous allons commencer par introduire d'autres algorithmes plus simples qui partagent des nombreuses idées communes avec SEARN.

3.5.1 Politique non-stationnaire

Les algorithmes réunis dans cette section sont très similaires. Chacun d'eux sera présenté avec ses garanties théoriques de manière à présenter au lecteur le principe des changements locaux étape par étape. Contrairement à SEARN qui sera présenté dans la section suivante, les algorithmes présentés ici utilisent des politiques telles que la règle de choix de l'action change en fonction de numéro d'ordre de l'action, autrement dit des politiques que nous pouvons qualifier de « non-stationnaires ».

Algorithme FORWARD L'algorithme FORWARD original, tel qu'il est présenté dans [Ross and Bagnell, 2010], repose sur le problème d'apprentissage expert. Dans ce travail, nous le présentons en utilisant le problème d'apprentissage tuteur afin de mettre en avant la structure commune qu'il partage avec les algorithmes BACKWARD et RANDOMWARD, que nous définissons dans la suite de la section.

La politique $\pi_n = \text{PushForward}(\pi_{n-1}, \tau_n)$ est une politique qui est différente de la politique π_{n-1} juste pour la n-ème décision : elle utilise le classifieur τ_n alors que la politique π_{n-1} fait appel à l'expert pour la n-ème décision. Les décisions de 1 à $n-1$ sont réalisées avec les classifieurs $\tau_1, ...\tau_{n-1}$ appris lors des itérations précédentes, puis avec l'expert pour les décisions à partir de $n+1$. La politique peut donc être définie par la formule

Algorithme 7 : FORWARD

Données : données d'apprentissage représentant $\widetilde{D^+}$, ensemble paramétrique Π, longueur minimale assurant le chemin complet T, politique de l'expert déterministe π_{exp}

Initialisation : $\pi_0 \leftarrow \pi_{\text{exp}}$;
pour $n = 1..T$ **faire**
  Construire le problème d'apprentissage TutProb$^{n-1}(\pi_{n-1})$;
  Apprendre τ_n résolvant TutProb$^{n-1}(\pi_{n-1})$;
  $\pi_n = \text{PushForward}(\pi_{n-1}, \tau_n)$;
fin
retourner π_N;

récursive suivante :

$$\pi_n = \text{PushForward}(\pi_{n-1}, \tau_n) = \begin{cases} \pi_{n-1} & \text{pour } 1 \leq t \leq n-1 \\ \tau_n & \text{pour } t = n \\ \pi^* & \text{pour } t \geq n+1 \end{cases} \tag{3.15}$$

Par la suite, nous utiliserons une formulation différente de cette définition, ne marquant pas explicitement l'appel à l'expert :

$$\pi_n = \text{PushForward}(\pi_{n-1}, \tau_n) = \begin{cases} \pi_{n-1} & \text{pour } t \neq n \\ \tau_n & \text{pour } t = n \end{cases} \tag{3.16}$$

L'algorithme FORWARD possède la garantie théorique suivante :

Lemme 3.5.1. *Soit π_T la politique obtenue à l'aide de l'algorithme forward. L'égalité suivante est vérifiée :*

$$L(\pi_T) = L(\pi_{\text{exp}}) + \sum_{n=1}^{T} l_{\text{loc}}(\tau_n; \pi_{n-1}) \tag{3.17}$$

Ce lemme a été démontré dans [Ross and Bagnell, 2010] en s'appuyant sur la formule (3.15). Étant donnée son importance dans notre travail, nous allons donner une démonstration très détaillée de ce premier lemme et un peu plus générale que la preuve originale, sans utiliser explicitement le fait que la politique π_n fait appel à l'expert pour les positions $(n+1, ..., T)$ et en nous appuyant pour cela sur la formule (3.16).

Démonstration. (Cette démonstration est une généralisation de la démonstration présentée dans [Ross and Bagnell, 2010].) Considérons la

perte associée à la politique π_n. Soit $\widetilde{s}_{\text{div}}$ la distribution des états où le nouveau classifieur τ_n est appliqué. Autrement dit, $\widetilde{s}_{\text{div}}$ est la distribution des états d'arrivée après $n-1$ actions faites par la politique π_n :

$$\widetilde{s}_{\text{div}} = \widetilde{s}_i \circ \pi_n^{n-1}$$

Comme π_{n-1} est égale à π_{n-1} pour les $n-1$ premières actions, il est également vrai que :

$$\widetilde{s}_{\text{div}} = \widetilde{s}_i \circ \pi_{n-1}^{n-1}$$

$V(\widetilde{s}_i; \pi_{n-1})$ s'exprime donc de la manière suivante :

$$\begin{aligned} V(\widetilde{s}_i; \pi_{n-1}) &= R(\widetilde{s}_i \xrightarrow{\pi_{n-1}, n-1} \widetilde{s}_{\text{div}}) + R(\widetilde{s}_{\text{div}} \xrightarrow{\pi_{n-1}, \cdot} S_f) \\ &= R(\widetilde{s}_i \xrightarrow{\pi_{n-1}, n-1} \widetilde{s}_{\text{div}}) + V(\widetilde{s}_{\text{div}}; \pi_{n-1}) \end{aligned} \tag{3.18}$$

La première égalité vient de la fonction de valeur décomposée sur deux morceaux de trajectoire de la politique π_{n-1} : $(s_i, s_{\text{div}}) \cup (s_{\text{div}}, s_{\text{fin}} \in S_f)$. La deuxième égalité provient directement de la définition de V. Nous avons une décomposition similaire de $V(\widetilde{s}_i; \pi_n)$:

$$\begin{aligned} V(\widetilde{s}_i; \pi_n) &= R(\widetilde{s}_i \xrightarrow{\pi_n, n-1} \widetilde{s}_{\text{div}}) + R(\widetilde{s}_{\text{div}} \xrightarrow{\pi_n, 1} \cdot \xrightarrow{\pi_n, \cdot} S_f) \\ &= R(\widetilde{s}_i \xrightarrow{\pi_{n-1}, n-1} \widetilde{s}_{\text{div}}) + R(\widetilde{s}_{\text{div}} \xrightarrow{\tau_n, 1} \cdot \xrightarrow{\pi_{n-1}, \cdot} S_f) \\ &= R(\widetilde{s}_i \xrightarrow{\pi_{n-1}, n-1} \widetilde{s}_{\text{div}}) + Q(\widetilde{s}_{\text{div}}, \tau_n(\widetilde{s}_{\text{div}}); \pi_{n-1}) \end{aligned} \tag{3.19}$$

La première égalité vient à nouveau de la fonction de valeur décomposée cette fois-ci sur trois morceaux de trajectoire de la politique π_n : de l'état initial jusqu'à s_{div}, de s_{div} jusqu'à $\pi_n(s_{\text{div}})$ et de $\pi_n(s_{\text{div}})$ jusqu'à l'état final. La deuxième égalité utilise la formule de récurrence (3.15). La troisième égalité utilise la définition de Q.

En utilisant les deux représentations (3.18) et (3.19) on obtient l'expression suivante pour la fonction de valeur $V(\widetilde{s}_i; \pi_n)$:

$$\begin{aligned} V(\widetilde{s}_i; \pi_n) &= V(\widetilde{s}_i; \pi_{n-1}) - V(\widetilde{s}_{\text{div}}; \pi_{n-1}) + Q(\widetilde{s}_{\text{div}}, \tau_n(\widetilde{s}_{\text{div}}); \pi_{n-1}) \\ &= V(\widetilde{s}_i; \pi_{n-1}) + l^n_{\text{loc}}(\tau_n; \pi_{n-1}) \end{aligned} \tag{3.20}$$

ce qui nous amène à :

$$\begin{aligned} L(\pi_n) &= V(\widetilde{s}_i; \pi_n) - V(\widetilde{s}_i; \pi_{\text{exp}}) \\ &= V(\widetilde{s}_i; \pi_{n-1}) + l^n_{\text{loc}}(\tau_n; \pi_{n-1}) - V(\widetilde{s}_i, \pi_{\text{exp}}) \\ &= L(\pi_{n-1}) + l^n_{\text{loc}}(\tau_n, \pi_{n-1}) \end{aligned} \tag{3.21}$$

La perte de la politique π_n par rapport à la politique π_{n-1} se trouve donc être simplement la *perte locale* associée à l'utilisation du nouveau classifieur τ_n. Remarquons que certaines suites d'actions peuvent avoir moins de n actions. Dans ce cas, la perte correspondante $l^n_{\text{loc}}(\tau_n; \pi_{n-1})$ est considérée égale à 0, ce qui permet de ne pas traiter spécialement ce cas.

Il suffit maintenant d'appliquer la procédure présentée N fois afin d'obtenir le résultat recherché (3.17). □

Cette garantie signifie que la qualité de chaque nouvelle politique apprise est déterminée par la qualité de la politique précédente et par la perte *locale* du nouveau classifieur. Autrement dit, la politique apprise est garantie de ne pas être sujette aux effets d'avalanche. Il est intuitif dans ce cas de comprendre le mécanisme qui permet d'éviter l'effet d'avalanche. Chaque nouveau classifieur est entrainé à faire ses choix connaissant la politique qui va engendrer l'état où il sera appliqué. Il va donc rencontrer lors de l'inférence des situation similaires à celles qu'il a pu observer lors de l'apprentissage.

Algorithme BACKWARD Nous proposons l'algorithme suivant afin de montrer un mécanisme différent de gestion de l'effet d'avalanche et faire une transition plus lisse avec les sections suivantes de ce chapitre. Comme son nom le suggère, l'algorithme BACKWARD est le symétrique de l'algorithme FORWARD. Les classifieurs qui composent la politique non-stationnaire sont entrainés dans l'ordre inverse : du dernier au premier. Pour chaque classifieur, les exemples d'apprentissage sont construits de la manière suivante : les états sont issus de l'application de la politique expert, alors que les coûts des actions sont estimés à l'aide des classifieurs appris lors des itérations précédentes.

La politique $\pi_n = \text{PushBackward}(\pi_{n-1}, \tau_n)$ diffère de la politique π_{n-1} uniquement pour la $T - n + 1$-ème décision pour laquelle elle utilise le classifieur τ_n au lieu de faire appel à l'expert. Les autres décisions sont réalisées à l'aide de l'expert pour les décisions de 1 à $T - n$, et par des classifieurs $\tau_1, ..., \tau_{n-1}$ pour les décisions suivantes. Ce qui donne la formule récursive suivante :

$$\pi_n = \text{PushBackward}(\pi_{n-1}, \tau_n) = \begin{cases} \pi^* & \text{pour } 1 \leq t \leq T - n \\ \tau_n & \text{pour } t = T - n + 1 \\ \pi_{n-1} & \text{pour } t > T - n + 1 \end{cases} \tag{3.22}$$

Comme pour le cas de FORWARD, nous utiliserons une forme différente de

Algorithme 8 : BACKWARD

Données : données d'apprentissage représentant $\widetilde{D^+}$, ensemble paramétrique Π, longueur minimale assurant un chemin complet T, politique de l'expert déterministe π_{exp}

Initialisation : $\pi_0 \leftarrow \pi_{\text{exp}}$;
pour $n = 1..T$ **faire**
  Construire le problème d'apprentissage $\text{TutProb}^{T-n}(\pi_{n-1})$;
  Apprendre τ_n résolvant $\text{TutProb}^{T-n}(\pi_{n-1})$;
  $\pi_n = \text{PushBackward}(\pi_{n-1}, \tau_n)$;
fin
retourner π_T;

cette formule :

$$\pi_n = \text{PushBackward}(\pi_{n-1}, \tau_n) = \begin{cases} \tau_n & \text{pour } t = T - n + 1 \\ \pi_{n-1} & \text{pour } t \neq T - n + 1 \end{cases} \tag{3.23}$$

Les garanties théoriques et la démonstration correspondante sont très proches de celles de l'algorithme FORWARD.

Lemme 3.5.2. *Soit π_T la politique obtenue à l'aide de l'algorithme BACKWARD. L'égalité suivante est respectée :*

$$L(\pi_T) = L(\pi_{\text{exp}}) + \sum_{n=1}^{T} l_{\text{loc}}^{T-n+1}(\tau_n; \pi_{n-1}) \tag{3.24}$$

Démonstration. La démonstration se déroule comme pour l'algorithme FORWARD, à la différence que $s_{\text{div}} = \widetilde{s_i} \circ \pi_n^{T-n}$; le reste de la démonstration est inchangée. □

Comme dans le cas de FORWARD, la qualité de chaque nouvelle politique apprise est déterminée par la qualité de la politique précédente et par la perte *locale* du nouveau classifieur. L'effet d'avalanche n'est donc à nouveau pas présent, mais le mécanisme qui permet de l'éviter est différent de celui mis en œuvre pour l'algorithme FORWARD. Dans ce cas, le nouveau classifieur ne possède pas de connaissances sur la politique qui le précède dans la construction du chemin, il a donc une mauvaise connaissance des états dans lesquels il devra prendre des décisions. Par contre, cette fois-ci, le classifieur possède une connaissance parfaite de la politique qui sera appliquée après sa décision. Cela lui permet d'estimer exactement le coût de

chaque action et donc d'éviter les chemins que la politique suivante ne sait pas gérer.

Notons que, contrairement au cas de FORWARD, BACKWARD nécessite une estimation des coûts des actions sachant que la politique qui agira ensuite n'est pas celle de l'expert ; le problème d'apprentissage expert ne peut donc pas être appliqué ici et il est donc nécessaire d'avoir recours à un tuteur.

Algorithme RANDOMWARD Nous avons donc vu deux mécanismes qui permettent, chacun à sa manière, d'éviter l'effet d'avalanche : une connaissance parfaite du passé dans le cas de FORWARD et du futur dans le cas de BACKWARD. Il est aussi possible d'utiliser des solutions intermédiaires, c'est-à-dire des connaissances partielles sur le passé et sur le futur. Pour cela, on apprend les classifieurs dans un ordre quelconque, ce que fait l'algorithme RANDOMWARD, qui est, comme son prédécesseur, proposé dans ce travail à titre d'illustration.

RANDOMWARD est un algorithme très similaire aux deux précédents à ceci près que les classifieurs qui composent les politiques non-stationnaires sont entrainés dans un ordre arbitraire. Les algorithmes FORWARD et BACKWARD ne sont donc que des cas particuliers de celui-ci.

Algorithme 9 : RANDOMWARD

Données : données d'apprentissage représentant $\widetilde{D^+}$, ensemble paramétrique Π, suite de positions $\mathcal{O}$, nombre d'itérations $N = |\mathcal{O}|$, politique de l'expert déterministe π_{exp}

1 Initialisation : $\pi_0 \leftarrow \pi_{\text{exp}}$;
2 **pour** $n \in 1..N$ **faire**
3 | Construire le problème d'apprentissage $\text{TutProb}^{o_n-1}\pi_{n-1}$);
4 | Apprendre τ_n résolvant $\text{TutProb}^{o_n-1}(\pi_{n-1})$;
5 | $\pi_n = \text{PushRandomward}(\pi_{n-1}, \tau_n, o_n)$;
6 **fin**
7 **retourner** π_N;

La politique $\pi_n = \text{PushRandomward}(\pi_{n-1}, \tau_n, o)$ est une politique qui est différente de la politique π_{n-1} juste pour la o-ème décision pour laquelle elle utilise le classifieur τ_n. Les autres décisions sont réalisées à l'aide de la

politique π_{n-1}. Ce qui donne la formule récursive suivante :

$$\pi_n = \text{PushRandomward}(\pi_{n-1}, \tau_n, o) = \begin{cases} \tau_n & \text{pour } t = o \\ \pi_{n-1} & \text{pour } t \neq o \end{cases} \tag{3.25}$$

La suite de positions $\mathcal{O}$ est une suite d'indices compris entre 1 et T indiquant chacun la position pour laquelle un nouveau classifieur doit être entrainé. Cette suite peut contenir un nombre arbitraire d'indices et il est donc possible que certaines positions ne soient pas entrainées et que d'autre le soient au contraire plusieurs fois. On appellera *couvrante* une suite de positions contenant chaque position au moins une fois.

Les garanties théoriques de RANDOMWARD dans le cas d'une suite de positions couvrantes sont très proches de celles des algorithmes FORWARD et BACKWARD :

Lemme 3.5.3. *Soit π_N la politique obtenue à l'aide de l'algorithme RANDOMWARD avec la suite de positions $\mathcal{O}$ couvrante. L'égalité suivante est respectée :*

$$L(\pi_N) = L(\pi_{\text{exp}}) + \sum_{n=1}^{N} l_{\text{loc}}^{o_n}(\tau_n; \pi_{n-1}) \tag{3.26}$$

Démonstration. La démonstration est similaire à celle de l'algorithme FORWARD, avec pour différence que $s_{\text{div}} = \widetilde{s_i} \circ \pi_n^{o_n - 1}$; le reste de la démonstration est inchangée. □

Dans le cas où la suite $\mathcal{O}$ n'est pas couvrante, la politique apprise nécessite l'expert au moment du test alors que celui n'est plus disponible. Dans ce travail nous considérons ce cas comme exceptionnel et les stratégies de choix de l'ordre doivent chercher à l'éviter. Néanmoins, dans le cas où la politique apprise continuerait à faire appel à l'expert, celui-ci sera remplacé par une politique aléatoire et nous considérons que la perte ne peut être limitée pour ce cas.

Corollaire 3.5.1. *Soit π_N la politique obtenue à l'aide de l'algorithme RANDOMWARD, et $\bar{\pi}_N$ la politique obtenue depuis π_N en remplaçant les appels à l'expert par des décisions aléatoires. L'inégalité suivante est respectée :*

$$L(\bar{\pi}_N) = \begin{cases} L(\pi_{\text{exp}}) + \sum_{n=1}^{N} l_{\text{loc}}^{o_n}(\tau_n; \pi_{n-1}) & \text{si } \mathcal{O} \text{ couvrante} \\ L_{\max} & \text{si } \mathcal{O} \text{ non} - \text{couvrante} \end{cases} \tag{3.27}$$

où $L_{\max}$ est la perte maximale possible.

3.5.2 Politique stationnaire : SEARN

Dans la section précédente nous avons introduit trois algorithmes d'apprentissage d'une politique non-stationnaire, c'est-à-dire une politique utilisant différents classifieurs pour différentes positions. Pour diverses raisons, notamment dans le cas où la suite d'actions est très longue voire infinie, on préfère généralement obtenir une politique stationnaire, c'est-à-dire une politique utilisant un unique classifieur pour l'ensemble de la séquence [Ross et al., 2011].

Dans cette section nous commençons par introduire SEARN, un algorithme permettant d'apprendre une politique stochastique, tel qu'il est présenté originellement dans [Daumé et al., 2009]. Nous discutons ensuite de ses garanties théorique, et nous finirons en introduisant une version moins sensible à l'effet d'avalanche.

Algorithme 10 : SEARN

Données : données d'apprentissage représentant $\widetilde{D^+}$, ensemble paramétrique Π, vitesse d'apprentissage β, nombre d'itérations N, politique de l'expert déterministe π_{exp}

Initialisation : $\pi_0 \leftarrow \pi_{\text{exp}}$;
pour $n \in 1..N$ **faire**
    Construire un nouvel ensemble d'apprentissage TutProb(π_{n-1});
    Apprendre τ_n résolvant TutProb($\pi_{\text{n}-1}$);
    $\pi_n = \text{RandomMix}(\pi_{n-1}, \tau_n, \beta)$;
fin
retourner $\bar{\pi}_N$;

L'algorithme 10 présente le fonctionnement de SEARN. La politique $\pi_n = \text{RandomMix}(\pi_{n-1}, \tau_n)$ est une politique qui pour chaque décision utilise le classifieur τ_n avec une probabilité β, et la politique π_{n-1} de l'itération précédente avec une probabilité $1-\beta$. Ce qui nous donne la formule récursive suivante :

$$\pi_n = \text{RandomMix}(\pi_{n-1}, \tau_n, \beta) = \begin{cases} \tau_n & \text{avec } p = \beta \\ \pi_{n-1} & \text{avec } p = 1 - \beta \end{cases} \tag{3.28}$$

Cette formule récursive peut être également réécrite de manière explicite :

$$\text{RandomMix}(\pi_{n-1}, \tau_n, \beta)(s) = \begin{cases} \pi_{\text{exp}}(s) & \text{avec } p = (1-\beta)^n \\ \tau_1(s) & \text{avec } p = (1-\beta)^{n-1} \cdot \beta \\ \dots & \\ \tau_{n-1}(s) & \text{avec } p = (1-\beta) \cdot \beta \\ \tau_n(s) & \text{avec } p = \beta \end{cases} \tag{3.29}$$

On observe qu'en contrepartie d'être stationnaire, la politique obtenue est stochastique. L'utilisation d'une politique stochastique est une autre approche qui permet de s'assurer que la politique π_n diffère peu de la précédente. SEARN possède la garantie suivante [Daumé et al., 2009] :

Lemme 3.5.4. *Soit T la longueur maximale de la séquence d'actions et $\bar{\pi}_N$ la politique obtenue après N itérations de SEARN avec $\beta \leq \frac{1}{T}$ et après le remplacement des appels à l'expert par des décisions aléatoires. L'inégalité suivante est respectée :*

$$L(\bar{\pi}_N) \leq L(\pi_{\text{exp}}) + \beta T \sum_{n=1}^{N} l_{\text{loc}}(\tau_n; \pi_{n-1}) + \left(\frac{1}{2}\beta^2 T^2 N + T(1-\beta)^N\right) L_{\max} \tag{3.30}$$

où $L_{\max}$ est la perte maximale possible dans un treillis.

Démonstration. (Cette démonstration est une reprise de la démonstration présentée dans [Daumé et al., 2009].) Considérons la récompense cumulée $V(s_i, \pi_n)$ de la politique obtenue à l'itération n vis-à-vis de la récompense $V(s_i, \pi_{n-1})$ de la politique obtenue à l'itération $n-1$. Soit k le nombre de fois où le nouveau classifieur τ_n a été appliqué sur la séquence d'actions. On distingue trois cas :

1. $k = 0$ dont la probabilité p_0 est bornée de la manière suivante :

   $$p_0 = (1-\beta)^T \leq 1 - T\beta$$

   Dans ce cas $V_{k=0}(s_i, \pi_n) = V(s_i, \pi_{n-1})$, car la suite des actions choisies par π_n reste inchangée par rapport à π_{n-1}.

2. $k = 1$ dont la probabilité p_1 est bornée de la manière suivante :

   $$p_1 = T\beta(1-\beta)^{T-1} \leq T\beta$$

   Dans ce cas $V_{k=1}(s_i, \pi_n) = V(s_i, \pi_{n-1}) + l_{\text{loc}}(\tau_n; \pi_{n-1})$. (La preuve en a été donnée pour l'algorithme FORWARD)

3. $k \geq 2$ dont la probabilité p_2 est bornée de la manière suivante :

$$p_2 \leq \sum_{i=2}^{T} \beta^i (1-\beta)^{T-1} \binom{T}{i} \leq \beta^2 \binom{T}{2} \leq \beta^2 \frac{T^2}{2}$$

Si le nouveau classifieur τ_n a été appliqué deux fois où plus, on remarque simplement que $V_{k=0}(s_i, \pi_n) \leq L_{\max}$, car un effet d'avalanche incontrôlé peut se produire.

En appliquant la formule des probabilités totales, pour la perte moyenne $L(\pi_n)$ on obtient :

$$\begin{aligned}
L(\pi_n) &= L(\pi_{\exp}) + V(s_i, \pi_n) - V(s_i, \pi_{\exp}) \\
&= L(\pi_{\exp}) + V_{k=0}(s_i, \pi_n) \cdot p_0 + V_{k=1}(s_i, \pi_n) \cdot p_1 + V_{k=2}(s_i, \pi_n) \cdot p_2 - V(s_i, \pi_{\exp}) \\
&\leq L(\pi_{\exp}) + V(s_i, \pi_{n-1}) - V(s_i, \pi_{\exp}) + T\beta l_{\text{loc}}(\tau_n; \pi_{n-1}) + T^2\beta^2 L_{\max} \\
&= L(\pi_{n-1}) + \beta T l_{\text{loc}}(\tau_n; \pi_{n-1}) + T^2\beta^2 L_{\max}
\end{aligned}$$

En appliquant cette procédure N fois à partir de la politique expert $\pi_{\exp}$ on obtient le résultat suivant :

$$L(\pi_N) = L(\pi_{\exp}) + \beta T \sum_{n=1}^{N} l_{\text{loc}}(\tau_n; \pi_{n-1}) + \frac{1}{2}\beta^2 T^2 N L_{\max}$$

Afin d'obtenir le résultat pour la politique $\bar{\pi}_N$ qui ne fait plus appel à l'expert, nous avons besoin d'estimer la probabilité que π_N fasse appel à l'expert. Après N itérations, la probabilité de recourir à l'expert se calcule en appliquant la formule correspondante du tirage avec remise :

$$1 - (1 - (1-\beta)^N)^T \leq T(1-\beta)^N \tag{3.31}$$

Comme pour l'algorithme RANDOMWARD, dans ce cas on considère simplement que la perte associée est limitée par $L_{\max}$. L'ajout du terme correspondant $T(1-\beta)^N L_{\max}$ achève la démonstration. □

La présence du terme :

$$\left(\frac{1}{2}\beta^2 T^2 N + T(1-\beta)^N\right) L_{\max}$$

dans l'expression (3.34) signifie que la politique obtenue $\bar{\pi}_N$ est sensible à l'effet d'avalanche avec un facteur

$$p_{\text{av}} = \frac{1}{2}\beta^2 T^2 N + T(1-\beta)^N$$

Plus p_{av} est petit, meilleure est la borne correspondante. Une valeur $p_{\text{av}} \geq 1$ signifie que théoriquement la politique obtenue n'apporte aucune amélioration par rapport à la plus mauvaise politique possible.

Pour conclure sur les garanties de SEARN, il est nécessaire d'estimer le nombre d'itérations et la vitesse d'apprentissage permettant de rendre l'effet d'avalanche raisonnable. Le compromis suivant entre les valeurs de β et N a été proposé dans [Daumé et al., 2009] :

Théorème 3.5.1. *Soit π_{fin} la politique apprise après $N = 2T^3\ln(T)$ itérations de SEARN avec $\beta = \frac{1}{T^3}$. L'inégalité suivante est alors respectée :*

$$L(\pi_{fin}) \leq L(\pi^*) + 2T\ln T \frac{1}{N}\sum_{n=1}^{N} l_{\text{loc}}(\tau_n; \pi_{n-1}) + \frac{1+\ln T}{T} L_{\max} \tag{3.32}$$

Démonstration. [Daumé et al., 2009] Notons que $T(1-\beta)^N < Te^{-N\beta}$. Il suffit maintenant de remplacer β et N dans l'expression (3.34) par les valeurs correspondantes. □

Afin de donner un ordre de grandeur de ce compromis, pour une séquence d'actions de longueur $T = 10$, il est nécessaire de faire plus de 4500 itérations. Cela permet d'obtenir une sensibilité théorique à l'effet d'avalanche avec un facteur $p_{\text{av}} \approx 0.33$. Pour des séquences plus longues cette garantie devient de plus en plus intéressante mais demande un nombre encore plus grand d'itérations.

3.5.2.1 Réduction de l'effet d'avalanche

Le facteur p_{av} est complexe à minimiser car les deux termes qui le composent posent des contraintes contradictoires. Afin de minimiser le premier terme :

$$p^1_{\text{av}} = \frac{1}{2}\beta^2 T^2 N$$

la vitesse d'apprentissage β et le nombre d'itérations N doivent être petits, tandis que pour le deuxième terme :

$$p^2_{\text{av}} = T(1-\beta)^N$$

l'inverse est nécessaire.

En effet, pour réduire la probabilité d'un appel à l'expert (première source de l'effet d'avalanche), il doit être remplacé au plus vite par d'autres politiques, ce qui requiert une grande vitesse d'apprentissage. Par contre, une grande vitesse d'apprentissage a pour conséquence de changer de manière importante la politique d'une itération à l'autre, ce qui rend difficile l'estimation des conditions où le nouveau classifieur sera appliqué, d'où la deuxième source d'effet d'avalanche. Un raisonnement similaire s'applique au nombre d'itérations.

Il est possible de remplacer la procédure de mélange stochastique par la procédure suivante :

$$\pi_n = \text{RandomMix}(\pi_{n-1}, \tau_n) = \begin{cases} \tau_n & \text{pour } \hat{t} = \text{rand}(1, T) \\ \pi_{n-1} & \text{pour } t \neq \hat{t} \end{cases} \tag{3.33}$$

La modification proposée élimine une des deux sources d'effet d'avalanche, ce qui permet de le minimiser plus simplement. Il est facile de voir que ce mécanisme implique la garantie suivante :

Lemme 3.5.5. *Soit π_N la politique obtenue après N itérations de SEARN utilisant le mécanisme de mélange stochastique (3.33). L'inégalité suivante est alors respectée :*

$$L(\pi_N) \leq L(\pi_{\text{exp}}) + \sum_{n=1}^{N} l_{\text{loc}}(\tau_n; \pi_{n-1}) + T \left(\frac{T-1}{T} \right)^N L_{\max} \tag{3.34}$$

Démonstration. Il suffit de ré-appliquer toutes les étapes de la preuve du lemme 3.5.4, en prenant en compte le fait que le nouveau classifieur τ_n sera toujours appliqué une seule fois sur la trajectoire. Pour estimer la probabilité de faire appel à l'expert on utilise le fait que pour chaque nouveau classifieur chaque position a une chance sur T d'être décodé (le facteur $1/T$ remplace l'ancien facteur β dans l'expression (3.31)). □

Par exemple, pour $T = 10$ le nombre d'itérations $N = 100$ permet de réduire l'effet d'avalanche théorique à un facteur $p_{\text{av}} \approx 2 \cdot 10^{-4}$.

Le mécanisme (3.33) de mélange stochastique présente l'inconvénient que la politique ne peut plus être appliquée dans un état indépendamment des autres. Une manière de contourner cette difficulté est de supposer que pour chaque espace de recherche G, l'ordre d'application des classifieurs est fixe et peut être déterminé simplement. Autrement dit, le mélange (3.33) est considéré comme pseudo-aléatoire.

3.5.3 Conclusion

Dans cette section, nous avons tout d'abord décrit des algorithmes d'apprentissage de politique non-stationnaire utilisant des évolutions locales de la politique initiale et qui évitent entièrement l'effet d'avalanche. Puis, nous avons présenté une méthode d'utilisation de ces évolutions locales permettant d'apprendre une politique stationnaire. Ceci est possible en utilisant un mélange stochastique ; néanmoins dans ce cas l'effet d'avalanche ne peut être évité entièrement. Nous avons ensuite proposé une méthode de mélange stochastique permettant de réduire significativement l'effet d'avalanche par rapport à la méthode originale [Daumé et al., 2009].

Un point important concernant les garanties de tous ces algorithmes est qu'elles ne donnent aucune information sur la qualité de la politique obtenue par rapport aux autres politiques de la classe considérée. On ne sait notamment pas si elle est proche de la politique optimale de cette classe ou non. Ces garanties ne nous informent que sur l'écart par rapport à l'expert. C'est une importante différence avec AGGRAVATE qui garantit d'obtenir une politique presque optimale pour un certain nombre de problèmes mais qui n'évite pas un gaspillage de ressources.

D'autres travaux utilisant le mélange stochastique pour réduire l'effet d'avalanche existent dans la littérature. SMILE [Ross and Bagnell, 2010] est un algorithme proche de SEARN utilisant le problème d'apprentissage avec l'expert plutôt que celui avec le tuteur, ce qui le rend plus facilement réalisable en pratique. Dans [Goldberg and Nivre, 2012] la même idée est appliquée à l'apprentissage en ligne avec le perceptron. Les applications récentes utilisant des réseaux de neurones récurrents sont décrites par exemple dans [Ranzato et al., 2016] et [Bengio et al., 2015]. Par contre, les modifications utilisées dans ces travaux sont incompatibles avec les garanties telle qu'elles sont décrites dans cette section.

3.6 Conclusion et références supplémentaires

Dans ce chapitre, nous sommes face à un problème de poule et d'œuf : afin d'entrainer la politique optimale, il est nécessaire de construire un bon problème d'apprentissage mais pour construire celui-ci la politique est elle-même nécessaire. La problématique « apprendre à chercher » s'avère être un problème d'optimisation de la répartition des ressources de manière à obtenir la politique optimale, contrairement l'apprentissage par renforcement ou par imitation, qui font face à des difficultés supplémentaires d'exploration. Faisant partie de ce cadre d'apprentissage déjà étudié, ce problème peut

emprunter de nombreuses solutions dans les travaux plus anciens.

Le vocabulaire utilisé dans ce chapitre (politique, fonction de valeur, récompense...) est, comme indiqué en introduction, emprunté à l'apprentissage par renforcement, tandis que l'utilisation que nous faisons de l'expert provient de l'apprentissage par imitation ; néanmoins, la nature du problème considéré ici est relativement différente. Notamment, dans le cadre présenté, nous disposons d'un accès direct aux pertes de potentiel pour chaque action contrairement à l'apprentissage par renforcement où il n'est en général possible d'observer que des informations partielles sur la récompense cumulée finale [Sutton and Barto, 1998] et ou l'accès au choix de l'expert n'est jamais envisagé. De même, la disponibilité d'un simulateur puissant permettant d'obtenir les pertes de potentiel nous éloigne également de l'apprentissage par imitation où généralement seules des traces du comportement de l'expert sont disponibles [Geist, 2016].

De nombreuses méthodes présentées dans cette section sont déjà bien connues dans le cadre de l'apprentissage par renforcement. Par exemple, le problème de gaspillage de ressources se retrouve dans le théorème sur la qualité d'une politique gloutonne basée sur la fonction de valeur approximée dans [Bertsekas, 1987] (page 236, le théorème est présenté pour le cas d'un horizon infini). Ensuite, l'algorithme illustratif ITERATE ressemble au concept d'itération de politique approximée (approximated policy iteration, [Bertsekas and Tsitsiklis, 1996]). L'algorithme BACKWARD est très similaire à l'algorithme NAPI présenté dans [Kakade, 2003] pour le cas sans appel à l'expert. Dans ce même travail on trouve l'algorithme d'itération conservative de politique (Conservative Policy Iteration), élaboré pour le cas de l'horizon infini et également sans appel à l'expert, qui est très similaire à SEARN.

Dans les deux prochains chapitres nous allons détailler, sur l'exemple de SEARN [8], les différents aspects pratiques de l'application des méthodes « apprendre à chercher » à des tâches du traitement automatique de la langue.

8. Une comparaison pratique de SEARN et DAGGER (l'ancêtre d'AGGRAVATE, voir la section 3.4.4) à été réalisée par exemple dans [Vlachos, 2013]. La conclusion de ce travail est que pour la tâche considérée (extraction d'informations biomédicales) DAGGER est plus performant et converge plus rapidement. Néanmoins, dans nos expériences préliminaires nous n'avons pas constaté de différences significatives entre les deux méthodes en termes de performances et de vitesse de convergence(à condition d'un choix soigneux des hyper-paramètres). Nous avons choisi de mener les expérimentations suivantes sur l'exemple de SEARN, qui a, dans sa version présentée dans la section 3.5.2.1, l'avantage *théorique* de demander un nombre d'itérations relativement faible afin de limiter l'effet d'avalanche.

Chapitre 4

SEARN pour l'étiquetage de séquences

Sommaire

L'étiquetage de séquences a pour objectif d'attribuer des étiquettes à des observations en tenant compte du fait qu'elles ont une structure de séquence. Elle se distingue d'un étiquetage simple où chacune des observations serait traitée indépendamment des autres. La séquence reste toutefois une structure simple et permet donc d'étudier dans de bonnes conditions l'utilisation d'algorithmes d'apprentissage structuré.

Dans ce chapitre nous allons étudier comment SEARN peut être utilisé pour l'étiquetage de séquences. Nous discuterons en particulier de comment il est possible de prédire les étiquettes des différentes positions non pas dans un ordre fixe comme c'est habituellement le cas mais dans un ordre adapté à la tâche. L'idée de décoder dans un ordre adapté n'est pas nouvelle en soi (voir, par exemple, [Shen et al., 2007]). Néanmoins, dans le cadre de SEARN, la question de l'adaptation de l'algorithme au cas de l'expert non-déterministe (voir la définition en section 3.1.3) se posera. Après une étude théorique, nous validerons les résultats obtenus sur plusieurs tâches de traitement automatique de la langue.

4.1 Espace de recherche

Soient X et Y les ensembles d'entrées et de sorties respectivement. Dans la suite de ce chapitre, nous noterons $\mathbf{x} = (x_t)_{t=1}^{T}$ une séquence d'observations et $\mathbf{y} = (y_t)_{t=1}^{T}$ une séquence d'étiquettes avec $x_t \in X$ et $y_t \in Y$. On notera $\mathbf{y}^*$ la séquence d'étiquettes, dite de référence, que l'on voudrait obtenir en tant que résultat de l'étiquetage de la séquence $\mathbf{x}$.

Nous utiliserons également les notations $\mathbf{x}_\theta$ et $\mathbf{y}_\theta$, où θ est un ensemble d'entiers compris entre 1 et T, pour désigner les éléments de $\mathbf{x}$ et de $\mathbf{y}$ dont les indices sont dans θ. Notons que les indices dans θ ne sont pas nécessairement consécutifs. Nous noterons en particulier $i{:}j$ l'ensemble des entiers compris entre i et j (inclus), $\mathbf{x}_{i:j}$ et $\mathbf{y}_{i:j}$ désignant les ensembles d'observations et d'étiquettes correspondants.

Nous allons introduire, dans les prochaines sections, deux types d'espace de recherche permettant de réaliser l'étiquetage de séquences. Ils formalisent deux manières de réaliser l'étiquetage : en considérant les observations soit dans un ordre fixe, soit dans un ordre libre.

4.1.1 Décodage en ordre fixe

L'étiquetage en ordre fixe repose sur le fait que l'ordre dans lequel vont être attribuées les étiquettes aux différentes positions est décidé à l'avance et indépendant du processus d'étiquetage. Les choix les plus courants sont l'ordre gauche-droite — de la première à la dernière position — ainsi que l'ordre droite-gauche — de la dernière à la première position. D'autres alternatives peuvent être utilisées comme un décodage dans un ordre déterminé par un arbre de dépendance par exemple.

L'espace de recherche, tel que présenté dans la section 3.1.1, pour le décodage en ordre fixe d'une séquence est défini par (S, A, S_f, s_i, r). Sans perte de généralité, nous le présentons ici pour l'ordre gauche-droite, un simple renommage des positions permettant d'obtenir n'importe quel ordre fixe.

- $S = \left\{(\mathbf{x}_{1:T}, \mathbf{y}_{1:t}), \mathbf{x}_{1:T} \in X^T, \mathbf{y}_{1:t} \in Y^t, 1 \leq t \leq T\right\}$: chaque état est décrit par l'ensemble des observations ainsi qu'une sortie partielle dont seules les t premières étiquettes sont connues ;
- $A_s = \{y, y \in Y\}$: chaque action prolonge la séquence d'étiquettes déjà connues par une nouvelle étiquette et permet de passer de l'état $s = (\mathbf{x}_{1:T}, \mathbf{y}_{1:t})$ à l'état $s' = (\mathbf{x}_{1:T}, \mathbf{y}_{1:t+1})$ avec $\mathbf{y}_{1:t+1} = \mathbf{y}_{1:t} \oplus y$;
- $S_f = \left\{(\mathbf{x}_{1:T}, \mathbf{y}_{1:T}), \mathbf{x}_{1:T} \in X^T, \mathbf{y}_{1:T} \in Y^T\right\}$: on considère final chaque état correspondant à un étiquetage complet de la séquence ;

— $s_i = (\mathbf{x}_{1:T}, \emptyset)$: l'état initial est l'état correspondant à une séquence d'étiquettes vide ;
— la récompense est définie en fonction de la perte associée au choix de l'étiquette y' à la position t.

Comme nous le verrons dans la section 4.3.3, différentes fonctions objectif peuvent être utilisées pour définir la perte associée à une action. En pratique, une perte binaire indiquant simplement si l'action choisie est l'action de référence est souvent une bonne approximation des fonctions objectifs plus complexes.

Le décodage en ordre fixe permet d'avoir une structure d'espace de recherche simple comportant $\sum_{t=0}^{T} |Y|^t$ états et $\sum_{t=0}^{T} |Y|^t \times |Y|$ actions ($|Y|$ actions par état). Un tel espace de recherche est illustré par la figure 4.1.

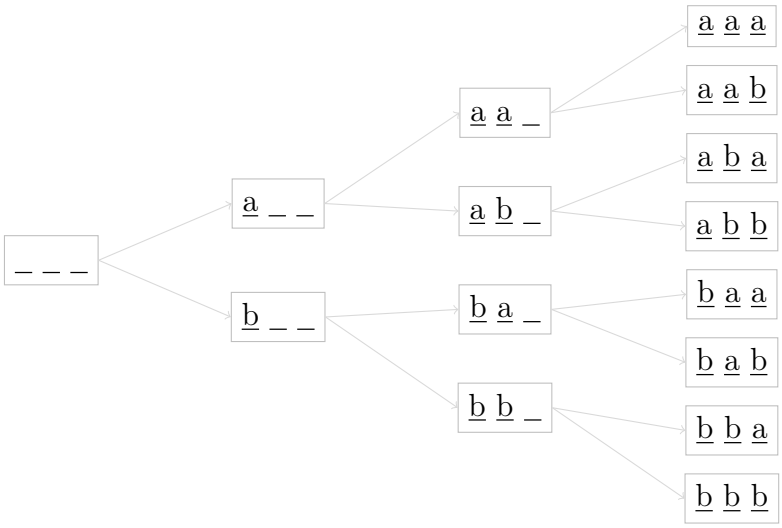


FIGURE 4.1 – Espace de recherche correspondant à l'étiquetage gauche-droite d'une séquence de trois observations avec deux étiquettes possibles (« a » et « b »). Pour simplifier, la représentation des états n'inclut que la séquence d'étiquettes attribuées, la séquence d'observations étant constante. Les positions dont l'étiquette n'est pas encore prédite sont indiquées par le symbole _.

Lors d'un décodage action par action, les premières décisions doivent être prises en disposant de très peu d'informations tandis que les dernières décisions disposent d'une information quasi complète. Le décodage en ordre fixe correspond à choisir de manière plus ou moins arbitraire quelles seront les décisions les moins bien informées.

Une autre approche consiste à chercher un ordre optimal permettant, dans la mesure du possible, de prendre chaque décision au moment ou l'information nécessaire est disponible. Dans la section suivante, nous présentons une telle approche : le décodage en ordre libre.

4.1.2 Espace de recherche pour décodage en ordre libre

L'étiquetage de séquences en ordre libre suppose l'attribution des étiquettes à des positions sans ordre prédéfini : chaque nouvelle étiquette peut être attribuée à n'importe quelle position parmi celles qui n'ont pas encore été étiquetées. L'espace de recherche correspondant est le suivant :

- $S = \{(\mathbf{x}_{1:T}, \mathbf{y}_\theta), \theta \in \wp([1:T])\}$ où $\wp(E)$ est l'ensemble des parties de E : chaque état correspond à une séquence dont seuls les éléments dont les indices sont dans θ ont été étiquetés ;
- $A_s = \{(y,t), y \in Y, t \notin \theta\}$ pour $s = (\mathbf{x}_{1:T}, \mathbf{y}_\theta)$: une action choisit *une des positions non étiquetées et son étiquette* et permet de passer de l'état $(\mathbf{x}_{1:T}, \mathbf{y}_\theta)$ à l'état $(\mathbf{x}_{1:T}, \mathbf{y}_{\theta'})$ avec $\theta' = \theta \cup \{t\}$;

L'ensemble d'états finaux S_f, l'état initial s_i ainsi que les récompenses restent définis de la même manière que pour le décodage en ordre fixe.

La figure 4.2 donne un exemple de l'espace de recherche considéré lors d'un étiquetage en ordre libre. De manière générale, celui-ci comporte $\sum_{t=0}^{T} \binom{t}{T} |Y|^t$ états et $\sum_{t=0}^{T} \binom{t}{T} |Y|^t \times |Y|$ actions. On notera que, lorsque la contrainte sur l'ordre des étiquettes est relâchée, plusieurs séquences d'actions peuvent conduire à un même état.

Le décodage en ordre libre apporte un degré de liberté supplémentaire par rapport au décodage en ordre fixe. Il est par exemple possible de commencer le décodage par les positions les plus simples pour lesquelles peu d'information est nécessaire afin de trouver la bonne étiquette et de terminer par les plus complexes qui pourront alors profiter du maximum d'information. Cette plus grande liberté se fait par contre au prix d'un espace de recherche significativement plus grand que celui défini pour l'ordre fixe, ce qui à un impact sur le coût du décodage.

4.2 SEARN et l'expert non-déterministe

Lorsque l'espace de recherche considéré est l'espace des décodages en ordre fixe, la définition d'un chemin de l'expert est triviale puisque un seul chemin permet de reconstruire l'étiquetage de référence.

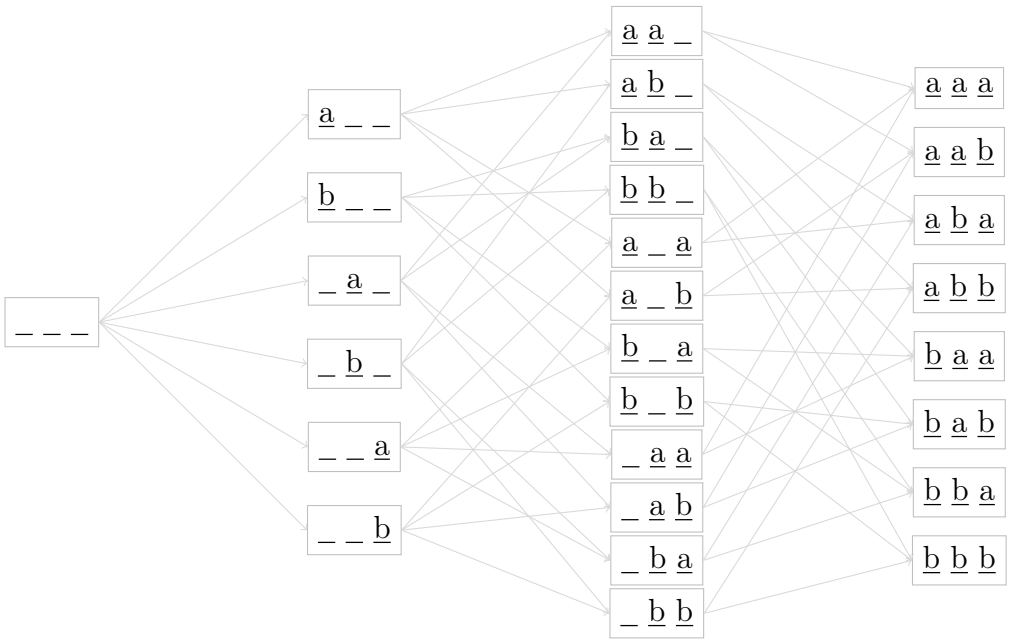


FIGURE 4.2 – Espace de recherche d'un étiquetage en ordre libre d'une séquence de trois observations lorsqu'il y a deux étiquettes possibles (« a » et « b »). Les positions dont l'étiquette n'est pas encore prédite sont indiquées par le symbole _.

Le cas de l'espace de recherche des décodages en ordre libre est plus complexe. Il n'existe pas un seul chemin conduisant à la référence mais un grand nombre qui ne se différencient que par l'ordre dans lequel les étiquettes sont placées. Il y a donc dans chaque état potentiellement plusieurs actions de l'expert et l'on parle dans ce cas d'*expert non-déterministe*.

On notera A_s^* l'ensemble des actions de l'expert accessibles à partir de l'état s et $\pi_{\text{exp}}(s)$, conformément au chapitre précédent, une politique expert qui comporte une procédure de choix d'une de ces actions de l'expert A_s^*. Cette procédure de choix peut être aléatoire, déterministe en choisissant par exemple toujours l'action agissant sur la position la plus à gauche, ou se baser sur d'autres critères dépendant par exemple de la tâche ou de la politique que l'on souhaite apprendre. Dans la section suivante nous expliquons notre choix pour la procédure d'adaptation de SEARN au cas de l'expert non-déterministe.

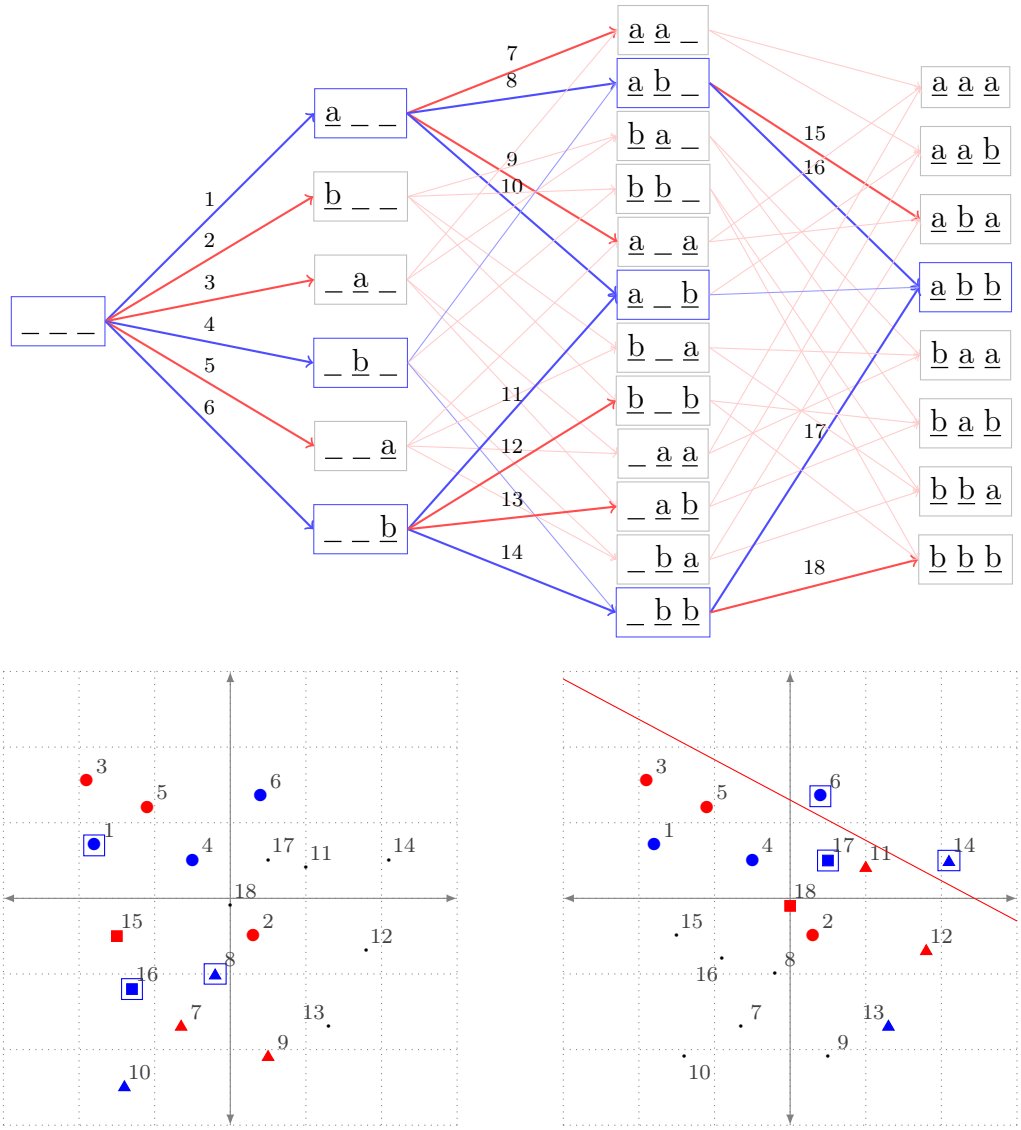


FIGURE 4.3 – Séparabilité de deux chemins de l'expert différents : l'espace de recherche considéré est celui de la figure 4.2, les actions de l'expert sont représentées en bleu et les actions impliquant des pertes de potentiel non-nulles en rouge (pour l'étiquetage de référence « abb »). Les actions sont représentées par des points dans un espace à deux dimensions correspondant aux caractéristiques qui leur sont associées. Sur chaque plan, les points encadrés correspondent aux actions de l'expert du chemin désiré. La droite rouge du plan de droite est le séparateur (dans le sens large du terme) : il représente un modèle capable de réaliser les choix nécessaires. Le chemin de l'expert représenté sur le plan de gauche ne possède pas de tel séparateur dans la classe des modèles linéaires.

4.2.1 Choix de la politique de l'expert

Il existe des cas où certains chemins de l'expert ont de nets avantages par rapport aux autres. L'exemple de la figure 4.3 représente l'espace de recherche de la figure 4.2 avec « abb » comme étiquetage de référence. Les actions en gras et numérotées sont les actions pertinentes pour notre problème d'apprentissage. Ces actions sont représentées par des points en deux dimensions ; leurs caractéristiques sont les coordonnées des points correspondants. Toutes les actions de l'expert sont en bleu et celles impliquant une perte de potentiel non-nulle sont en rouge ; au total, six chemins de l'expert existent.

Pour suivre un chemin de l'expert particulier, le modèle doit être capable de choisir les actions le composant dans tous les états visités lors de l'inférence. Choisir une action parmi un ensemble, dans ce contexte, signifie lui attribuer le plus grand score de confiance. Dans cet exemple nous interprétons la distance jusqu'à l'hyperplan-séparateur en tant que score de confiance.

Par exemple, afin de pouvoir suivre le chemin de l'expert $1 \rightarrow 8 \rightarrow 16$, le modèle doit être capable de choisir l'action 1 parmi les actions 1 à 6, ainsi que l'action 8 parmi les actions 7 à 10 ainsi que préférer l'action 16 à l'action 15. Comme on le voit sur le plan de gauche, cela n'est pas possible à réaliser avec un modèle linéaire. Par contre, pour le chemin de l'expert composé des actions $6 \rightarrow 14 \rightarrow 18$ il existe un modèle réalisant les choix corrects tout au long du chemin ; ce modèle choisira donc l'action 6 parmi les actions 1 à 6, l'action 14 parmi les actions 11 à 14 et préférera l'action 17 à l'action 18. Notons que les actions 17 et 18 sont toutes les deux du même côté du séparateur et ont des scores de confiance négatifs, mais l'action 17 est plus proche au séparateur et possède donc le score le moins négatif, elle sera donc préférée à l'action 18.

Le fait qu'un des meilleurs chemins puisse être suivi sans aucune erreur indique qu'il est préférable de réaliser l'apprentissage sur un chemin séparable car de bonnes performances sont dans ce cas garanties. Toutefois, comme nous l'avons déjà noté au chapitre 1 pour le cas de la classification simple, en pratique ces situations sont rares ; généralement le comportement de l'expert ne peut pas être imité parfaitement. Dans le cas général, la question du choix du chemin de l'expert qui amène aux meilleures performances n'est pas résolu. Dans [Goldberg and Nivre, 2012], [Aufrant et al., 2017] et [He et al., 2012] la stratégie de choix suivante a été appliquée avec succès : le chemin de l'expert choisi est le chemin composé des actions de l'expert ayant le meilleur score selon la politique apprise précédemment [1]. Cela cor-

1. [He et al., 2012] proposent une stratégie encore plus générale en imitant une politique qui n'est pas expert mais de bonne qualité et facile à apprendre. Dans nos

respond à l'opération suivante :

$$\pi_{\text{exp}}(s|\pi) = \arg\max_{a \in A_s^*} (\text{score}_\pi(\text{a})) \tag{4.1}$$

où $\text{score}_\pi(a)$ est le score de confiance donné à l'action a par la politique π. Cette approche repose sur l'intuition que la politique courante n'a besoin de changer que légèrement ses paramètres pour être améliorée. [Aufrant et al., 2017] démontrent que cette stratégie, en combinaison avec d'autres améliorations par rapport au système de référence, permet d'obtenir des distributions de caractéristiques représentant les actions explorées plus proches (dans le sens de distance de Kullback-Leibler) lors de l'entrainement et du test.

4.2.2 Adaptation de SEARN au cas de l'expert non-déterministe

Nous avons choisi de suivre la même intuition afin d'adapter SEARN au cas de l'expert non-déterministe. La première méthode théorique que nous proposons est la suivante : une politique apprise après la première exécution de SEARN est utilisée pour construire une nouvelle politique experte ; cette nouvelle politique choisira parmi les actions de l'expert non-déterministe l'action ayant le meilleur score selon la politique apprise précédemment. SEARN est ensuite exécuté avec la nouvelle politique experte, la procédure se répète un nombre arbitraire de fois [2], ce qui est formalisé dans l'algorithme 11 [3].

Notons que dans le cas de SEARN, la politique π_m obtenue à chaque itération $m \in [1, M]$ est stochastique. La question de comment obtenir un score de confiance avec cette politique se pose donc. Une solution qui semble valide consiste à utiliser les scores des classifieurs pondérés par les probabilités β. Néanmoins, le lien entre la simplicité d'apprentissage d'une action et le score de confiance est très heuristique. Utiliser uniquement le score du dernier classifieur appris peut être suffisant. Cela permet de

expériences préliminaires nous n'avons pas trouvé d'effet positif d'une telle stratégie sur nos problèmes ; nous resterons donc sur la problématique du choix des actions de l'expert plus simple à imiter.

2. En général, les données de validation sont utilisées afin de déterminer si les performances ont cessées de progresser.

3. La même idée peut être appliquée non seulement à SEARN, mais à tout autre algorithme décrit au chapitre 3. Pour cela il suffit de remplacer SEARN par l'algorithme correspondant dans la ligne 3 avec les paramètres adaptés.

Algorithme 11 : Utilisation de la politique expert basée sur la politique apprise précédemment.

Données : données d'apprentissage représentant $\widetilde{D^+}$, ensemble paramétrique Π, nombre d'itérations M, nombre d'itérations N interne, vitesse d'apprentissage β

1 Initialisation : $\pi_{\text{exp}}(\cdot) = \text{rand}(A^*)$;
2 **pour** $1 \leq m \leq M$ **faire**
3 $\quad \pi_m = \text{SEARN}(E_G, \Pi, \beta, N, \pi_{\text{exp}})$;
4 $\quad \pi_{\text{exp}}(\cdot) = \pi_{\text{exp}}(\cdot|\pi_m)$;
5 **fin**
6 **retourner** π_M;

modifier (4.1) de la manière suivante :

$$\pi_{\text{exp}}(s|\pi_m) = \arg\max_{a \in A^*_s} (\text{score}_{\tau_m}(\text{a})) \tag{4.2}$$

où τ_m est le dernier classifieur composant π_m.

Le principal avantage de l'algorithme 11 est que, quelle que soit la manière de choisir la politique experte à la ligne 3, *les garanties théoriques de SEARN restent valides*. En effet, les garanties présentées au chapitre 3 se basent sur le fait que la politique apprise s'éloigne à vitesse contrôlée de la politique initiale ; elles resteront valides pour n'importe quelle politique initiale, y compris pour des politiques de l'expert différentes.

L'algorithme 11 présente toutefois une difficulté pratique : devoir répéter la procédure d'apprentissage de SEARN plusieurs fois augmente significativement le temps de calcul. En s'inspirant de l'idée présentée dans [Goldberg and Nivre, 2012], une alternative consiste à mettre à jour la politique expert à chaque itération de SEARN. Plus précisément, le mélange des politiques défini dans l'équation (3.29), est remplacé par :

$$\text{RandomMix}(\pi_{n-1}, \tau_n, \beta)(s) = \begin{cases} \pi_{\text{exp}}(s|\tau_n) & \text{avec } p = (1-\beta)^n \\ \tau_1(s) & \text{avec } p = (1-\beta)^{n-1} \cdot \beta \\ \dots & \\ \tau_{n-1}(s) & \text{avec } p = (1-\beta) \cdot \beta \\ \tau_n(s) & \text{avec } p = \beta \end{cases} \tag{4.3}$$

L'algorithme obtenu combine donc les deux idées que sont l'utilisation d'un expert simple à apprendre et l'éloignement progressif de cet expert.

Notons que *les garanties théoriques de SEARN présentées au chapitre 3 ne sont plus applicables à cet algorithme*. En effet, ces garanties reposent

sur le fait que les politiques utilisées afin de construire les chemins d'apprentissage de deux itérations consécutives diffèrent très peu. Dans le cas de SEARN, cela repose sur le fait qu'une politique est construite à partir de la précédente et que la probabilité de s'en éloigner est faible. Dans cette nouvelle version de SEARN, à chaque itération la politique expert est mise-à-jour et donc toutes les politiques apprises peuvent elles aussi changer de manière importante. Toutefois, malgré l'absence de garanties théoriques, cet algorithme donne de bons résultats en pratique.

Dans le cas de AGGRAVATE cette stratégie simplifiée d'adaptation de l'expert peut aussi être utilisée. Pour cela, il suffit de remplacer la procédure de mélange donnée par l'équation (3.14) par :

$$\pi_n(s) = \begin{cases} \pi_{\text{exp}}(s, \pi_{n-1}) & \text{avec } p = \beta_n \\ \pi_{n-1}(s) & \text{avec } p = 1 - \beta_n \end{cases} \tag{4.4}$$

Notons que *pour AGGRAVATE avec un expert adapté de cette manière-là les garanties théoriques présentées au chapitre 3 sont conservées*. En effet, l'utilisation de la politique expert est « optionnelle » et peut être remplacée par n'importe quelle autre politique. Comme nous l'avons vu au chapitre 3, pour que le théorème 3.4.1 soit respecté, la seule condition nécessaire est que la probabilité d'utiliser cette autre politique diminue suffisamment vite avec le nombre d'itérations.

4.3 Expérimentations

Dans cette section, nous étudions les performances des deux modèles introduits dans ce travail, le modèle gauche-droite et le modèle en ordre libre, sur différentes tâches de traitement automatique de la langue. Dans ces expériences nous poursuivons les objectifs suivants :

1. déterminer si l'utilisation du modèle en ordre libre permet d'obtenir de meilleurs résultats grâce à son degré de liberté supplémentaire ;
2. pour le modèle en ordre libre, étudier l'intérêt pratique de l'utilisation de la technique d'adaptation de l'expert présentée dans la section précédente ;
3. comparer les performances de nos deux modèles à ceux d'un système d'état de l'art.

Les résultats obtenus ont été publiés dans [Knyazeva et al., 2015b], [Knyazeva et al., 2015a] et [Burlot et al., 2016]. Nous commencerons par expliciter les derniers détails de l'adaptation de SEARN à la prédiction

de séquences, ensuite nous présenterons notre protocole expérimental puis nous rapporterons et discuterons les résultats.

4.3.1 Mise en œuvre de SEARN

Dans la section précédente nous avons présenté deux espaces de recherche possibles ainsi que les experts associés. Afin de décrire l'application de SEARN aux tâches considérées, il est nécessaire de préciser certains aspects de l'algorithme.

Classifieur sensible aux coûts SEARN nécessite un classifieur sensible aux coûts pour attribuer des scores aux différentes actions. Dans ce chapitre nous allons utiliser la version simple de SEARN avec le problème d'apprentissage avec l'expert qui demande moins de calculs pour évaluer les coûts associés aux actions. Plus précisément, pour évaluer le coût associé à une action on suppose que toutes les actions suivantes sont choisies par l'expert. Comme il a été dit dans la section 4.1, pour établir les coûts des actions nous utilisons une métrique simple comptant le nombre d'étiquettes bien attribuées sur l'ensemble de la séquence. Dans ce cas les coûts des actions peuvent être soit 0 (pour les actions $a \in A^*$) soit 1 (toutes les autres actions).

Dans ces conditions, si l'expert est supposé déterministe, le classifieur sensible aux coûts est simplement un classifieur multi-classe. Ce classifieur peut être approximé à l'aide d'un ensemble de classifieurs binaires en utilisant la stratégie *un contre tous* qui présente l'avantage de fournir des scores de confiance pour chaque décision [Bishop, 2006].

Dans le cas de l'expert non-déterministe, la situation est plus complexe puisque plusieurs actions correctes sont possibles à chaque instant. La stratégie que nous avons adoptée consiste à regrouper les actions en fonction de la position sur laquelle elles agissent et d'appliquer un classifieur similaire au précédent sur chacun de ces groupes. L'action choisie au final est l'action ayant la plus grande confiance parmi toutes les actions possibles.

Dans toutes nos expériences nous utilisons comme classifieur binaire un séparateur à vaste marge[4] (SVM) avec un noyau linéaire et une régularisation ℓ_2 ; la valeur du paramètre contrôlant la régularisation est systématiquement déterminée sur un ensemble de validation.

4. Nous avons utilisé l'implémentation des SVM fournie dans la bibliothèque Scikit-Learn [Pedregosa et al., 2011]

Ensemble d'apprentissage, échantillonnage Le décodage en ordre libre a pour conséquence de faire exploser la taille de l'espace de recherche. En effet, les actions accessibles depuis un état permettent de choisir non seulement l'étiquette mais aussi la position à laquelle elle doit être placée, le nombre d'actions accessibles s'accroit donc de manière importante.

La prédiction d'une étiquette à une position donnée peut être réalisée à n'importe quel moment du décodage. Elle va donc donner lieu à un grand nombre d'exemples d'apprentissage similaires qui ne diffèrent que par la quantité d'information disponible. En effet, plus cette prédiction est réalisée tard, plus l'information présente dans l'historique des décisions est grande.

L'explosion de la taille de l'ensemble d'apprentissage peut rendre les calculs non-réalisables et rendre nécessaire l'utilisation de méthodes de réduction de la taille de cet ensemble. Différentes stratégies sont possibles telles qu'un sous-échantillonnage aléatoire de l'ensemble d'apprentissage.

Dans ce travail, nous avons choisi de réduire, si la complexité l'impose, la taille de l'ensemble d'apprentissage en ne retenant, à chaque étape du décodage, que les exemples concernant la position choisie par la politique courante. Cela signifie que chaque action ne donne lieu qu'à une quantité fixe d'exemples d'apprentissage (autant que d'étiquettes possibles) qui capturent les informations qui étaient disponibles au moment où une décision pour la position correspondante a été prise. Cette stratégie, que l'on appellera « équilibrée » permet de conserver un ensemble d'apprentissage de taille équivalente à celui obtenu pour un décodage en ordre fixe et de conserver une bonne répartition des exemples sur les différentes positions. Nos expériences préliminaires montrent que cette stratégie est très avantageuse par rapport à l'échantillonnage aléatoire avec le même nombre d'exemples d'apprentissage.

La figure 4.4 montre un exemple d'échantillonnage « équilibré » de l'ensemble d'apprentissage obtenu à l'aide de la stratégie présentée pour l'espace de recherche de la figure 4.2. La séquence considérée est décodée de manière suivante : l'étiquette « b » est placée en deuxième position, ensuite l'étiquette « a » est placée en première position, et enfin l'étiquette « a » est placée en troisième position. Les actions qui sont enlevées de l'ensemble d'apprentissage après la procédure d'échantillonnage sont les suivantes : à la première étape, ce sont les actions concernant la première et la troisième position, à la deuxième étape ce sont les actions concernant la troisième position. À la troisième étape les deux actions possibles sont retenues car elles concernent toutes les deux la dernière position restant à étiqueter. Les actions qui ont été placées dans l'ensemble d'apprentissage échantillonné sont représentées par des flèches continues tandis que celles qui ont été enlevées lors de l'échantillonnage sont en pointillés.

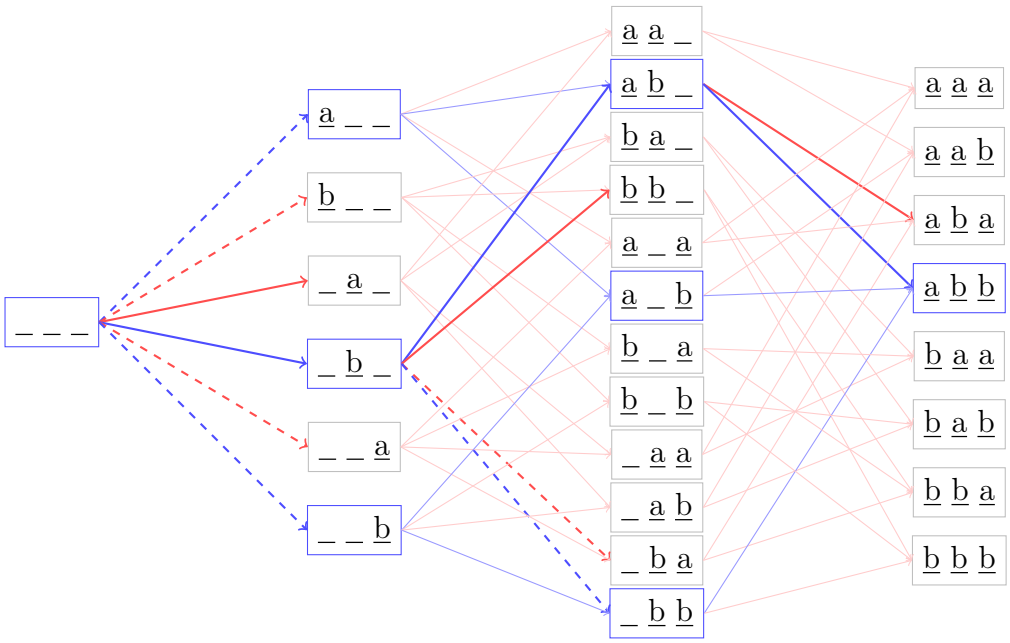


FIGURE 4.4 – Échantillonnage « équilibré » de l'ensemble d'apprentissage. L'espace de recherche considéré est celui de la figure 4.2. Le résultat du décodage est « abb » ; les étiquettes ont été placées sur les positions dans l'ordre $2 \rightarrow 1 \rightarrow 3$. Les actions qui ont été enlevées lors de l'échantillonnage sont en pointillés, contrairement aux actions qui sont entrées dans l'ensemble d'apprentissage « équilibré ».

Système de référence Les résultats de nos deux modèles sont comparés à un SVM multi-classes « simple » n'utilisant aucune information sur les étiquettes du voisinage et à un CRF de chaine linéaire [5] considérant uniquement l'étiquette précédente comme information de structure et réalisant une recherche exacte de la solution optimale.

4.3.2 Les tâches d'étiquetage de séquences considérées

Toutes les tâches considérées ont été décrites en détail dans le chapitre 2. On se concentrera ici sur la présentation des corpus utilisés pour l'apprentissage et le test ainsi que sur l'exploitation de l'information à longue distance.

5. Nous avons utilisé l'implémentation Wapiti [Lavergne et al., 2010] `http://wapiti.limsi.fr`

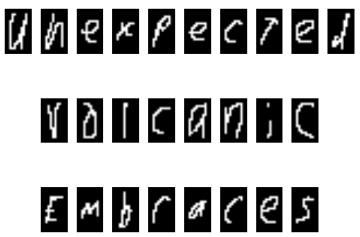


FIGURE 4.5 – Exemple de données pré-segmentées pour la reconnaissanse de l'écriture manuscrite.

Reconnaissance de l'écriture manuscrite Le corpus [6] utilisé contient 44 images de 150 mots (soit 6 600 exemples au total). Il s'agit d'un corpus artificiel, très structuré (la plupart des combinaisons d'étiquettes sont interdites et la connaissance d'une étiquette désambiguïse fortement les lettres voisines) qui est généralement utilisé pour tester les méthodes d'apprentissage structuré.

Chaque image est pré-segmentée en séquence d'images de lettres de taille 8×16 pixels. Quelques exemples de données pré-segmentées sont représentées sur la Figure 4.5. L'évaluation est réalisée en validation croisée : les données sont réparties en dix paquets ; neuf d'entre eux seront utilisés pour l'entraînement et un pour le test.

Nous utilisons comme caractéristiques la valeur de chacun des 144 pixels ainsi que les 9 étiquettes précédentes.

Prononciation automatique Dans nos expériences, nous utilisons le corpus NETTALK [7] [Sejnowski and Rosenberg, 1987], qui contient 20 008 mots anglais accompagnés d'une information de prononciation. Pour chaque mot anglais contenant T lettres, la représentation phonologique correspondante est encodées par deux séquences de T symboles : une séquence de phonèmes, utilisant un alphabet de 51 étiquettes (dont un symbole NULL) et une séquence décrivant la structure prosodique (pour les consonnes, la position dans la syllabe, pour les voyelles le degré d'accentuation) avec un alphabet de 6 symboles (dont un symbole NULL). Un extrait de NETTALK est représenté sur la figure 4.6. Nous utilisons 80% des données pour l'entraînement, 10% pour l'optimisation des hyper-paramètres et 10% pour le test.

6. http://www.seas.upenn.edu/~taskar/ocr/
7. https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Nettalk+Corpus)

aerodrome	E-rxdrom-	$1- < 0 >> 2 < -$
aeronaut	E-rxnc-t	$1- < 0 > 2- <$
aeronautics	E-rxnc-tIks	$2- < 0 > 1- < 0 <<$
aeroplane	E-rxplen-	$1- < 0 >> 2 < -$

FIGURE 4.6 – Extrait de NETTALK : à chaque mot est associé une séquence de phonèmes (avec '-' un symbole « NULL »), et structure prosodique (0,1,2 marquent differents degrés d'accentuation pour les voyelles, $>$ et $<$ marquent respectivement les attaques et coda de syllabes).

ADV	VMFIN	PIS	ADV	ADV	ADJA	NN	APPR	NN	VVINF	$
Zeitweise	müsse	man	allerdings	noch	längere	Wartezeiten	in	Kauf	nehmen	.

FIGURE 4.7 – Extrait de TIGER : une phrase allemande avec les catégories syntaxiques des mots.

Les caractéristiques sont les 4-grammes de lettres dans une fenêtre de taille ± 9 par rapport à la position courante. Les caractéristiques de structure correspondent aux étiquettes décodées (phonèmes ou marques prosodiques) ainsi que leurs combinaisons avec les caractéristiques de base.

Analyse morpho-syntaxique de l'allemand Dans ce travail nous avons décider de réaliser l'analyse morpho-syntaxique de l'allemand. Pour cela nous avons utilisé le corpus TIGER [8], contenant 50 000 phases allemandes, soit 900 000 mots étiquetés avec leur catégorie syntaxique. Ce corpus distingue 54 catégories grammaticales. Un exemple de phrase étiquetée du corpus est présenté sur la figure 4.7. Nous utilisons le partitionnement standard de ce corpus (80% des données pour l'entraînement, 10% pour la validation et 10% pour le test).

Dans nos expériences nous utilisons les caractéristiques suivantes : les informations de surface relatives au mot courant (présence de majuscules, de chiffres...), les préfixes et les suffixes de taille de 1 à 4 ainsi que les mots voisins dans une fenêtre de taille 2 par rapport au mot considéré. Le mot courant est également combiné avec les 9 étiquettes précédentes.

Identification structurée du locuteur Les expériences pour cette tâche ont été conduites sur la première saison de la série télévisuelle « The Big Bang Theory ». Les annotations manuelles sont disponibles pour les épisodes 1 à 6 avec les étiquettes suivantes : absence de parole, Howard,

8. http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger.en.html

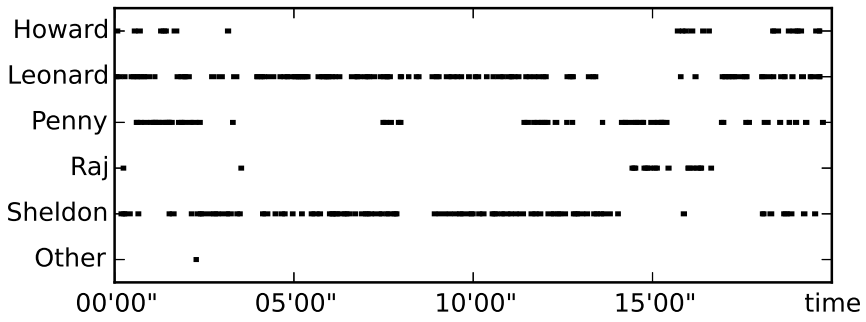


FIGURE 4.8 – Répartition du temps de parole sur la durée de l'épisode 2 de la première saison de la télé-série « The Big Bang Theory ».

Leonard, Penny, Raj, Sheldon, autre. L'étiquette *autre* réunit tous les personnages que l'on peut rencontrer dans la série à l'exception des 5 personnages principaux. Le corpus décrit est petit ($\approx$ 2 heures de contenu vidéo, repartis sur 6 épisodes), nous utilisons donc une validation croisée sur les 6 configurations différentes (4 épisodes pour l'entrainement, 1 pour le développement et 1 pour le test).

La métrique d'évaluation pour cette tâche est différente de celle prenant en compte juste la proportion de bonnes étiquettes, car la détection d'absence de parole n'a pas le même poids que la détection d'un bon locuteur. Nous utilisons la métrique standard pour cette tâche appelée proportion d'erreur d'identification (Identification Error Rate [Knyazeva et al., 2015a]) :

$$\text{IER} = \frac{\text{ND} \cdot \text{FD} \cdot \text{CONF}}{\text{DUR}}$$

où ND est la durée totale de parole non-détectée (étiquetée « absence de parole »), FD est la durée totale de parole incorrectement détectée, CONF est la durée de parole détectée mais où un mauvais locuteur a été attribué, DUR est la durée totale de la parole selon la référence.

Ré-inflexion Les données d'entrainement ont été extraites du projet *Universal Dependencies Treebank*[9]. Les corpus *Czech* and *Czech-CAC* ont étés utilisés, ce qui représente 2 millions de mots (dont 170 000 ont été réservés pour le jeu de développement).

9. http://universaldependencies.org

La préparation des données avec les informations morphologiques manquantes a été réalisée à l'aide de l'outil Morphodita [Straková et al., 2014] : chaque mot a été remplacé par son lemme combiné avec une étiquette contenant les informations morphologiques de manière à ce que la forme originale puissent être retrouvée à l'aide d'un dictionnaire. Ensuite, les attributs suivants ont été retirés [10] :

— le cas pour les noms, les adjectifs, les pronoms et les numériques ;
— le genre pour les adjectifs et les numériques ;
— le nombre pour les adjectifs et les numériques.

L'évaluation a été réalisée dans le cadre de l'étude des performances d'un système de traduction réalisant la première étape de traduction pour les données avec les attributs manquants et reconstruisant la forme fléchie lors d'une deuxième étape. Pour cela le système de traduction NCODE, présenté plus en détails dans le chapitre suivant, a été utilisé [11].

4.3.3 Résultats

Dans cette section nous présentons les résultats de l'application de SEARN aux différentes tâches d'étiquetage de séquences que nous avons étudiées.

Les principaux résultats expérimentaux sont résumés dans le tableau 4.1. Ce tableau compare les performances obtenues par nos systèmes de référence : SVM non-structuré et CRF linéaire ; ainsi que deux systèmes SEARN : en ordre gauche-droite (SEARN-GD) et en ordre libre (SEARN-OL). L'analyse des résultats sera réalisée selon trois axes :

— comparaison entre les deux variantes de SEARN ;
— comparaison entre SEARN et la prédiction non-structurée ;
— comparaison entre SEARN et le CRF structuré.

SEARN gauche-droite et SEARN en ordre libre Nous avons vu dans la section 4.2 que SEARN en ordre libre dispose de plus de liberté pour choisir un ordre avantageux de prise de décisions. À condition d'exploiter correctement cette puissance, SEARN en ordre libre doit donner des résultats au moins équivalents à ceux de SEARN gauche-droite, car l'ordre gauche-droite fait partie des stratégies explorées en ordre libre.

10. Les détails sur le choix des attributs à retirer peuvent être trouvés dans [Burlot et al., 2016].

11. Nous ne présentons ici les résultats que pour un système de traduction ; les données utilisées pour son entrainement, ainsi que des résultats plus complets pour d'autres systèmes de traduction peuvent être trouvés dans [Burlot et al., 2016].

Tâche	Métrique	Non-struct.	CRF	SEARN GD	SEARN OL
Écriture	ER (min)	25,03%	9,72%	8,07%	**2,72%**
Phonèmes		6,83%	6,98%	7,49%	**6,68%**
Accents		8,51%	8,29%	9,22%	**7,72%**
Mor. synt.		3,66%	**3,00%**	3,52%	3,09%
Locuteurs	IER (min)	29,8	29,3	30,0	**29,1**
Réinflex.	BLEU (max)	NE	19,26	**19,53**	19,50

Tableau 4.1 – Performances de SEARN vis-à-vis de deux systèmes de référence (CRF et non-structuré) sur les 6 tâches considérées. "Non-struct." signifie la méthode de maximum d'entropie (pour la réinflexion) et le SVM non-structuré (pour les autres tâches). Pour chaque tâche on indique la métrique utilisée ainsi que l'objectif associé (maximisation ou minimisation de cette métrique). Les deux dernières tâches étant les tâches intermédiaires dans une chaine de traitement, les résultats présentés caractérisent le résultat final du système utilisant les étapes indiquées. Les abréviations utilisées : GD - gauche-droite, OL - ordre libre, NE - non évalué.

Pour illustrer la capacité de notre système à choisir d'abord les décisions qui sont les plus sûres, nous avons représenté, sur la figure 4.9, le taux d'erreur en fonction de l'indice de la décision dans la séquence pour la tâche graphème-phonème. Cette figure montre clairement que, dans le modèle « ordre libre », les erreurs apparaissent beaucoup plus tardivement lors du décodage, ce qui limite le problème de la propagation d'erreurs et permet d'obtenir les bonnes performances observées.

Cette figure montre clairement que le choix de l'action à réaliser en fonction de la confiance du classifieur est raisonnable. Elle ne nous donne par contre pas d'information sur la qualité du résultat du décodage par rapport à l'ordre gauche-droite : en effet, l'ordre libre pourrait ne faire que reporter les erreurs sur les dernières décisions sans en changer la quantité. Pour voir si les décisions tardives parviennent effectivement à exploiter l'information supplémentaire dont elles disposent, il est nécessaire de comparer les performances des deux approches sur nos différentes tâches.

La comparaison des deux dernières colonnes du tableau 4.1 montre que le décodage en ordre libre obtient de meilleurs résultats que le décodage gauche-droite pour l'ensemble des tâches à l'exception de la ré-inflexion où les deux résultats sont équivalents.

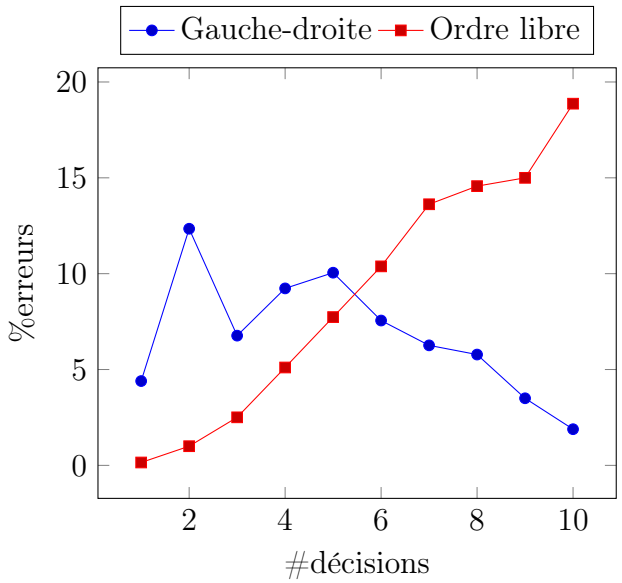


FIGURE 4.9 – Conversion graphème-phonème. Taux d'erreur en fonction de l'indice de la décision : modèle gauche-droite (en bleu) et modèle ordre libre (en rouge).

SEARN et la prédiction non-structurée Intuitivement, SEARN GD qui dispose des informations supplémentaires sur l'historique des décisions, ne doit pas donner de résultats moins bons que la prédiction non-structurée (avec un classifieur de même puissance).

En théorie, cela peut se produire à cause de la propagation d'erreur, qui ne peut pas être évitée complètement quel que soit l'algorithme d'apprentissage. En pratique, avec les algorithmes adaptés dont SEARN fait partie, on arrive à bien maitriser ce problème (à condition que les conditions d'apprentissage soient favorables : les données d'entrainement soient bien cohérentes avec les données de test et disponibles en quantité suffisante). Les résultats attendus devraient donc être au moins aussi bon que ceux de la prédiction non-structurée.

Cela est confirmé par nos expériences ; les éventuels différences négatives (comme dans le cas de SEARN gauche-droite pour les deux tâches de prononciation et de la reconnaissance de locuteur) sont dues à des phénomènes d'apprentissage sur une petite quantité de données.

SEARN et CRF Conformément à l'intuition, la recherche exacte, mise en œuvre par le CRF, améliore légèrement les résultats par rapport à l'approche gloutonne sauf lorsque la structure de la sortie joue un rôle crucial

comme pour la tâche de reconnaissance de l'écriture. En effet, dans ce cas la connaissance d'un historique riche (les 9 dernières étiquettes prédites alors que le CRF ne considère des dépendances qu'entre deux étiquettes consécutives) apporte une information suffisante pour compenser les erreurs induites par l'algorithme de recherche approchée. Ce phénomène apparaît également lorsque l'on considère un espace de recherche plus grand : le modèle « ordre libre » obtient des performances au moins aussi bonnes que le CRF et arrive même souvent à obtenir des résultats meilleurs.

4.3.4 La procédure d'adaptation de l'expert est-elle efficace ?

Afin de vérifier l'efficacité de l'apprentissage sur un chemin de l'expert plus simple à imiter, on représente sur la figure 4.10 les courbes d'apprentissage pour la tâche de conversion graphème-phonème des deux configurations. La première utilise SEARN tel qu'il est présenté au chapitre 3, c'est-à-dire en utilisant toujours l'expert aléatoire. Dans la seconde configuration on utilise SEARN avec la formule de mélange stochastique simplifiée (4.3) ; l'expert est donc réadapté de manière heuristique après chaque itération. On peut voir que le deuxième algorithme converge plus rapidement vers son niveau de performances final, ce qui justifie l'utilisation de l'adaptation de l'expert dès le début du processus d'apprentissage. Il est néanmoins important d'observer que le niveau de performances atteint par SEARN sans adaptation de l'expert est le même que celui obtenu par SEARN avec adaptation de l'expert. Notons aussi que l'application directe de l'algorithme 11 ne permet pas d'obtenir de meilleurs résultats que ceux de la méthode heuristique. Contrairement au résultats de [Goldberg and Nivre, 2012] et de [Aufrant et al., 2017] sur la tâche de l'analyse syntaxique en dépendances, nous n'avons pas réussi à améliorer les performances avec la méthode d'adaptation de l'expert présentée ; nous avons uniquement pu accélérer le processus d'apprentissage de la politique.

Une étude qualitative de l'ordre dans lequel le système effectue les actions montre que le choix par le système des chemins faciles à imiter semble correspondre à notre intuition. Par exemple, dans le cadre de la tâche de prédiction de la prosodie, on peut imaginer que l'accent secondaire peut être plus simple à prédire lorsque l'on connaît déjà la position de l'accent primaire. En analysant l'ordre d'attribution des étiquettes, on peut voir que l'accent secondaire (quand il est présent) dans un mot est décodé après l'accent primaire dans 70% des cas. De manière générale, le système a tendance à décoder les consonnes (qui composent 31% du nombre total

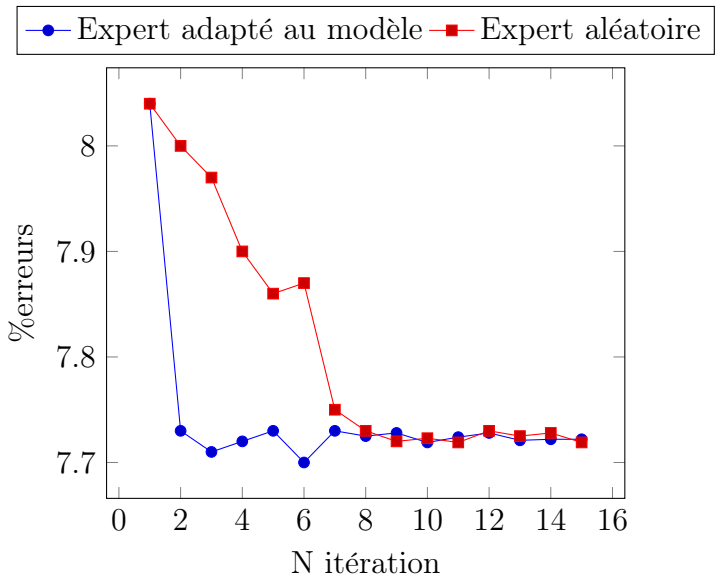


FIGURE 4.10 – Prononciation automatique (prédiction de l'accentuation). L'évolution du taux d'erreur au cours de l'apprentissage : expert adapté au modèle (en bleu) et expert aléatoire (en rouge).

d'erreurs) avant les voyelles (62% de toutes les erreurs) ; le reste des erreurs étant réalisées sur les sons muets (7%).

4.4 Conclusion

Dans ce chapitre nous avons présenté une adaptation de SEARN à l'étiquetage de séquences ainsi que des solutions aux différents problèmes qui se posent dans ce contexte.

Nous avons montré que l'utilisation de SEARN permet de réaliser le décodage des différentes positions dans un ordre adapté à la tâche. Cette approche a pour principale difficulté de rendre la politique expert non-déterministe. Deux stratégies ont été présentées afin de résoudre cette difficulté : une première qui conserve les garanties théoriques présentées dans le chapitre précédent mais qui est peu utilisable en pratique, et une deuxième moins coûteuse qui, même si elle ne conserve pas les garanties théoriques, montre de bons résultats en pratique.

Tous ces travaux ont été validés sur quatre tâches classiques de traitement automatique de la langue ainsi que sur deux tâches plus originales. Ces résultats ont été comparés à des systèmes de référence de qua-

lité et notre approche a montré des résultats équivalents ou meilleurs. Des expérimentations complémentaires ont permis de mettre en évidence l'intérêt du décodage en ordre libre dans le cadre de l'étiquetage de séquences.

Dans le chapitre suivant, nous étendrons ce système afin de l'appliquer au problème de la traduction automatique.

Chapitre 5

Traduction

Sommaire

Dans ce chapitre, après avoir présenté les principes généraux de la traduction automatique à base de segments, nous montrerons comment la démarche SEARN détaillée dans les chapitres précédents peut être appliquée à cette tâche. Nous décrirons ensuite une méthodologie d'évaluation des différentes étapes du processus de traduction tel que réalisé par SEARN, et nous l'appliquerons afin de diagnostiquer et corriger les problèmes présents dans notre système.

5.1 Généralités, traduction automatique à base de segments

Dans cette section nous allons préciser l'objectif de la traduction automatique et l'évaluation de celle-ci. Nous présenterons aussi le système de traduction NCODE [Crego et al., 2011], développé au LIMSI, qui nous servira de système de référence dans la suite du chapitre.

5.1.1 Objectif d'un système de traduction automatique : le score BLEU

Mesure de la qualité de traduction. Comme nous l'avons vu au chapitre 2, l'objectif de la traduction automatique est de transformer une phrase depuis une langue source vers une langue cible. Nous allons commencer ce chapitre par préciser cet objectif. La phrase cible proposée par un système de traduction devrait, dans l'idéal, être une bonne traduction de la phrase source. La question importante est, comment déterminer si la traduction est de bonne qualité. Cette question est complexe car il n'existe pas de critère exact permettant de déterminer cette qualité. Un humain est capable de résoudre cette tâche en utilisant des critères tels que la clarté, la préservation du sens ou encore la correction de la grammaire, mais même un humain ne peut formuler un critère bien défini. Par conséquent, deux humains jugeront différemment une même traduction.

Les systèmes de traduction utilisés dans ce chapitre se basent sur des méthodes d'apprentissage automatique et nécessitent donc une mesure de qualité calculée de manière automatique. Étant une tâche complexe à définir, l'évaluation de la qualité de traduction est problématique pour une machine. Différentes solutions existent dans la littérature, nous utiliserons ici la métrique BLEU [Papineni et al., 2002] qui est la métrique standard utilisée dans la communauté de la traduction automatique. La recherche de la meilleure traduction selon cette métrique sera donc en pratique l'objectif de ces systèmes de traduction automatique.

Calcul du score BLEU d'un ensemble de traductions. Le calcul du score BLEU nécessite une traduction produite par un humain de l'ensemble des phrases sources, appelée aussi *traduction de référence*. Cette traduction de référence est produite une fois et peut ensuite être utilisée pour évaluer n'importe quel système de manière automatique. Le score BLEU se base sur le nombre de *n-grammes* (segments de n mots consécutifs) communs entre la traduction résultant du processus de traduction automatique et la traduction de référence. Plus précisément, soient $\mathcal{E}$ et $\mathcal{R}$ deux ensembles de phrases parallèles : $\mathcal{E} = \mathbf{e}_{j=1}^{J}$ est l'ensemble des traductions produites par le système et $\mathcal{R} = \mathbf{r}_{j=1}^{J}$ l'ensemble des traductions de référence. Le score BLEU correspondant de degré n se calcule de la manière suivante :

$$\text{BLEU}(\mathcal{E}, \mathcal{R}) = BP \cdot \left(\prod_{m=1}^{n} \frac{c_m(\mathcal{E}, \mathcal{R})}{c_m(\mathcal{E})} \right)^{(1/n)} \tag{5.1}$$

avec $c_m(\mathcal{E}, \mathcal{R})$ le nombre de m-grammes communs entre $\mathcal{E}$ et $\mathcal{R}$ (on considère qu'un m-gramme est commun s'il existe j tel que le m-gramme appartient à la fois à $\mathbf{e}_j$ et $\mathbf{r}_j$) et $c_m(\mathcal{E})$ le nombre total de m-grammes dans $\mathcal{E}$. La pénalité de brièveté est un facteur supplémentaire pénalisant les phrases trop courtes :

$$BP = \min\left(1, e^{\left(1-\frac{c_1(\mathcal{R})}{c_1(\mathcal{E})}\right)}\right)$$

Le score BLEU tel que nous venons de le décrire présente toutefois un problème : pour chaque phrase source il est généralement possible de trouver au moins un mot de la phrase cible de manière sûre. Il devient alors possible de répéter un grand nombre de fois ce mot dans la traduction afin de gonfler artificiellement le nombre d'unigrammes corrects. Afin d'éviter ce problème, les quantités $c_m(\mathcal{E}, \mathcal{R})$ sont limitées au nombre d'occurrences de m-gramme correspondant dans la référence.

Le score BLEU varie de 0.0 (aucun m-gramme en commun entre l'hypothèse et la référence) à 1.0 (coïncidence complète de l'hypothèse avec la référence). Il est généralement présenté sous la forme d'un pourcentage entre 0.0 et 100.0. Remarquons que le score BLEU suppose l'évaluation de l'hypothèse en fonction d'une référence donnée, il ne prend donc pas en compte la variété des traductions possibles pour une phrase source ; seule la référence est considérée comme une traduction correcte [1].

Remarquons aussi que le score BLEU n'a de sens que si les précisions des n-grammes ne sont pas nulles, dans le cas contraire le score complet se réduit à zéro. C'est pour cette raison qu'il doit être calculé sur un corpus complet de phrases : cela minimise le risque d'avoir une précision nulle pour les grandes valeurs de n, ce qui est fréquent au niveau des phrases.

5.1.2 Le système NCODE

Un système de traduction automatique a donc pour objectif de trouver une traduction de bonne qualité selon la métrique choisie. Cette tâche est loin d'être triviale : l'espace des sorties est constitué de toutes les phrases possibles en langue cible, dans le cas des langues naturelles la bonne traduction doit alors être choisie parmi un nombre d'hypothèses gigantesque.

Le processus de traduction du système NCODE [Crego et al., 2011] s'effectue en deux phases. La première phase est la réduction, à l'aide de critères

1. Il existe des variantes de score BLEU adaptées au cas où plusieurs références sont disponibles. Elles sont peu utilisées en pratique à cause du manque de données disposant de multiples références adaptées. Dans ce travail nous ne considèrerons pas ce type d'évaluation.

simples, de l'espace des sorties à un ensemble d'hypothèses de taille raisonnable parmi lesquelles la traduction sera cherchée. Cet ensemble d'hypothèses est organisé de manière compacte et appelé par la suite le *treillis de recherche*. La deuxième phase consiste à choisir la meilleure hypothèse de traduction dans ce treillis à l'aide d'un critère plus raffiné.

Première phase : construction du treillis de recherche. La construction du treillis de recherche est réalisée en trois étapes. La phrase source est tout d'abord réordonnée de manière à ce que les mots qui la composent soient dans l'ordre attendu dans la phrase cible. La phrase source réordonnée est ensuite segmentée et traduite segment par segment de manière monotone.

L'étape de réordonnancement produit un treillis qui encode de manière compacte une partie des permutations de la phrase source. Cette étape est réalisé à l'aide de règles sur les séquences de catégories morpho-syntaxiques et permet de modéliser des phénomènes tels que l'inversion du nom et de l'adjectif entre le français et l'anglais. Ces règles sont collectées lors de l'apprentissage sur des corpus parallèles alignés et leur application sur la phrase source permet de construire un treillis des mots sources réordonnés.

NCODE appelle *tuple* une paire de séquences de mots où la première séquence est une séquence arbitraire des mots de la langue source et la deuxième séquence est une de ses traductions possibles. Voici quelques exemples de tuples pour la traduction du français vers l'anglais :

```
(nous, we)
(voulons, want)
(nous voulons, we want)
(traductions, translations)
(des traductions, translations)
```

Les étapes suivantes de construction du treillis de recherche reposent sur la *table des tuples* qui est composée de toutes les paires connues du système. Le treillis de réordonnancement est transformé en treillis segmenté pour lequel les arcs représentent non pas un seul mot source mais des segments continus. Le treillis de traduction, où chaque arc représente un tuple, est ensuite obtenu en appliquant la table des tuples à ce treillis de segmentation. La figure 5.1 illustre les étapes de réordonnancement, segmentation et traduction pour une phrase.

L'exemple de la figure 5.2 représente un treillis de traduction complet, ainsi qu'un zoom sur une partie de ses nœuds et arcs pour une meilleure lisibilité. Chaque chemin dans le treillis de traduction représente une traduction complète de la phrase source pouvant être produite par le système.

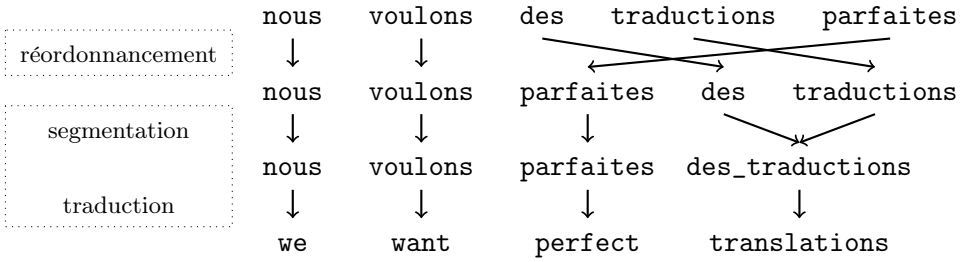


FIGURE 5.1 – Les étapes de décodage de NCODE. Cet exemple est une reprise de la figure 2 de [Crego et al., 2011].

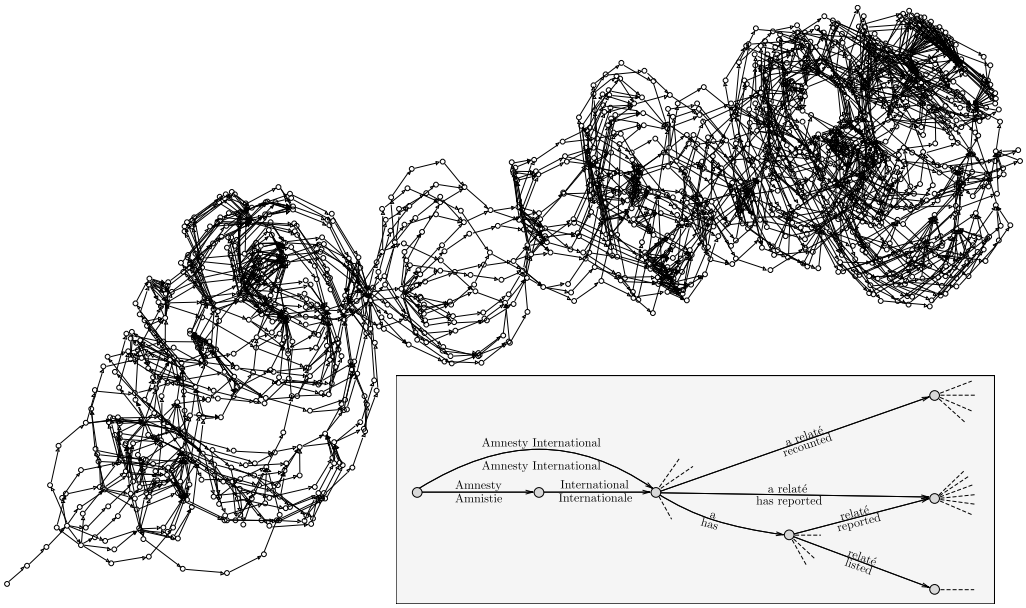


FIGURE 5.2 – Exemple de treillis de traduction de taille réelle.

En pratique, ce treillis peut être gigantesque et n'est donc pas construit entièrement avant la recherche mais au fur et à mesure de celle-ci. Une recherche en faisceau est utilisée afin de n'explorer qu'une partie raisonnable du treillis complet sans trop dégrader les performances du système.

Deuxième phase : recherche de la meilleure hypothèse du treillis. C'est lors de la deuxième phase que l'on choisit la meilleure traduction dans le treillis de recherche. Pour cela, un score est attribué à chaque hypothèse de traduction en combinant les scores individuels de modèles qui évaluent différents aspects de la traduction. Les principaux modèles utilisés sont :

— modèle de traduction : les scores associés à la paire segment source / segment cible qui évaluent la plausibilité de cet appariement ;
— modèle de réordonnancent : les scores associés à une permutation des mots ;
— modèle de langue : le score donné par un modèle de langue n-grammes sur la traduction.

Tous ces modèles sont entrainés de manière heuristique sur des corpus parallèles. Le score final d'une traduction est obtenu par une combinaison linéaire des scores des modèles estimés de manière à maximiser [2] le score BLEU d'un corpus de développement.

Le choix d'un algorithme d'optimisation performant est crucial pour construire une traduction de bonne qualité. Néanmoins aucun algorithme ne pourra donner de bon résultats, si le treillis de recherche ne contient que des hypothèses de mauvaise qualité. Les deux phases de traduction déterminent ainsi la qualité de la traduction produite par NCODE.

Le système NCODE représentait l'état de l'art de la traduction automatique ces dernières années, ce qui a été confirmé par plusieurs campagnes d'évaluation WMT. La situation change en ce moment avec l'arrivée à maturité de systèmes de traduction à base de réseaux de neurones qui a profondément changé la traduction automatique et domine actuellement les campagnes d'évaluation [Bojar et al., 2017].

5.1.3 Le score LinBLEU et la traduction oracle

La qualité des traductions produites par NCODE est donc limitée par la qualité des hypothèses de traduction présentes dans le treillis de recherche. La question qui vient alors naturellement est celle de la qualité de traduction potentiellement atteignable dans le treillis de recherche utilisé.

Pour répondre à cette question, il est possible de calculer, à l'aide de la référence, le *meilleur* score BLEU que l'on peut obtenir en cherchant une traduction dans ce treillis. Dans ce travail nous appellerons *oracle* le concept qui suppose l'utilisation de la référence afin de trouver la meilleure hypothèse de traduction présente dans un treillis. Nous utiliserons également le terme de *traduction oracle* afin de désigner la traduction obtenue de cette manière.

Afin obtenir la traduction oracle exacte il est nécessaire de calculer le score BLEU de chaque traduction possible *du corpus entier* pour ensuite

2. Plusieurs algorithmes d'optimisation peuvent être utilisés tels que MERT [Och, 2003] ou MIRA [Crammer et al., 2006].

de choisir la meilleure. Cela n'est pas réalisable en pratique car beaucoup trop couteux.

Une manière de simplifier ce calcul est d'utiliser la métrique approchée appelée BLEU' qui correspond à calculer le score BLEU phrase par phrase [Wisniewski et al., 2010, Sokolov et al., 2012]. Plus précisément, afin de calculer les traductions oracle du corpus, pour chaque phrase on choisit l'hypothèse du treillis qui maximise le score BLEU calculé comme si cette phrase composait le corpus entier.

L'utilisation de la métrique BLEU' permet de réduire la complexité de calcul, mais exige toujours la recherche de la meilleure hypothèse de traduction par énumération de toutes les hypothèses du treillis. Dans [Leusch et al., 2008] il a été montré que ce problème est NP-difficile. Plusieurs manière de simplifier le calcul de l'hypothèse maximisant le score BLEU' existent dans la littérature [Li and Khudanpur, 2009, Sokolov et al., 2012]. Dans ce travail nous avons choisi d'utiliser la métrique LinBLEU, initialement présentée dans [Tromble et al., 2008] et utilisée pour le calcul des traductions oracle dans [Sokolov et al., 2012].

De même que le score BLEU (ou BLEU'), le score LinBLEU reflète la similarité entre deux phrases en comparant le nombre de n-grammes communs entre une phrase et la référence. Pour une traduction $\mathbf{e}$ et une référence $\mathbf{r}$, le score LinBLEU se calcule de la manière suivante :

$$\text{LinBLEU}(\mathbf{e}, \mathbf{r}) = \theta_0 |\mathbf{e}| + \sum_{m=1}^{n} \theta_m \tilde{c}_m(\mathbf{e}, \mathbf{r}) \tag{5.2}$$

où $\tilde{c}_m(\mathbf{e}, \mathbf{r})$ est le nombre de m-grammes dans $\mathbf{e}$ présents aussi dans $\mathbf{r}$. Notons que, contrairement au score BLEU, le compte de n-grammes communs n'est pas limité par le nombre de fois que le n-gramme est présent dans la référence. Par exemple, pour la phrase $\mathbf{e}$ =*oui , oui , oui* et la référence $\mathbf{r}$ =*oui* le compte correspondant $\tilde{c}_1(\mathbf{e}, \mathbf{r}) = 3$ (le 1-gramme *oui* est considéré comme présent 3 fois, bien qu'on ne le trouve qu'une seule fois dans la référence) [3].

Le score LinBLEU pour le corpus complet $\mathcal{E} = \mathbf{e}_{j=1}^{J}$ et la référence $\mathcal{R} = \mathbf{r}_{j=1}^{J}$ peut être calculé comme une simple moyenne des scores de ces phrases :

$$\text{LinBLEU}(\mathcal{E}, \mathcal{R}) = \frac{1}{J} \sum_{j=1}^{J} \text{LinBLEU}(\mathbf{e}_j, \mathbf{r}_j)$$

3. En pratique ce n'est pas un réel problème puisque les treillis sont construits lors d'une première phase de recherche, qui contraint à ne traduire qu'une seule fois chaque segment source.

En fonction de n, le treillis peut être adapté de manière à ce que le score LinBLEU des hypothèses soit décomposable sur les arcs [4], et donc l'hypothèse ayant le meilleur LinBLEU peut être trouvée en calculant le plus court chemin dans ce treillis.

La recherche de la traduction minimisant le score LinBLEU permet d'obtenir la traduction oracle qui, on l'espère, maximise le score BLEU. Pour cela, il est nécessaire de déterminer les scores à associer au différents n-grammes. Dans [Tromble et al., 2008] il a été montré qu'il n'est pas nécessaire d'optimiser l'ensemble des scores θ_i, mais uniquement deux paramètres p et r, qui déterminent ensuite les valeurs des θ_i pour $i \in [0, n]$ de la manière suivante :

$$\theta_n = \begin{cases} 1 & \text{si } n = 0 \\ \dfrac{-1}{4 \cdot p \cdot r^{n-1}} & \text{si } n > 0 \end{cases}$$

Notons que pour la majorité des applications pratiques, il n'est pas nécessaire de trouver l'oracle optimal et ceux obtenus à l'aide de paramètres relativement proches de l'optimum sont suffisants. Dans [Sokolov et al., 2012] il a été montré que la plage de valeurs des paramètres qui amène à des résultats proches de l'optimum est relativement grande et indépendante de la langue.

Typiquement, le score BLEU prend en compte les n-grammes pour $n \leq 4$, il serait donc logique de considérer un score LinBLEU basé lui aussi sur les 1-4-grammes. Les auteurs de [Sokolov et al., 2012] mettent en évidence le fait qu'une hypothèse contenant un grand nombre de 2-grammes communs avec la référence contient aussi de nombreux n-grammes communs pour $n > 2$; il est donc possible de se limiter à l'ordre 2 sans perdre trop de précision. Nous allons utiliser la notation n-LinBLEU pour désigner la taille maximale de n-gramme considérée, par exemple 4-LinBLEU pour le LinBLEU basé sur les 1-4-grammes et 2-LinBLEU pour celui basé sur les 1-2-grammes.

Notons que bien qu'à l'origine le score LinBLEU ait été conçu pour être une approximation du score BLEU, dans ce travail les scores LinBLEU ne doivent pas être considérés comme proches en valeur absolue, mais uniquement corrélés. Notamment, les valeurs LinBLEU dépendent des paramètres choisis : deux scores LinBLEU calculés avec des paramètres différents ne peuvent être comparés entre eux. Un exemple de calcul peut être trouvé

4. Pour cela, on transforme le treillis en un treillis équivalent (c'est-à-dire contenant exactement les mêmes hypothèses de traduction) mais ayant la propriété que pour chaque nœud, tous les chemins entrants se terminent par le même n-gramme. Cette procédure de décomposition est décrite en détails dans [Sokolov et al., 2012].

Phrase à évaluer	# (0g, 1g, 2g)	2-LinBLEU $\theta = [1, -1, -4]$ ($p = 0,25$, $r = 0,25$)	2-LinBLEU $\theta = [1, -0,5, -1]$ ($p = 0,5$, $r = 0,5$)	BLEU
the big red apple . *can I take it ?*	(5, 5, 4) (5, 5, 4)	-16	-1,5	100,0
big red apple . *may I take it ?*	(4, 4, 3) (5, 4, 3)	-11,5	-0,5	71,44
a yellow lemon ! *should I take it ?*	(4, 0, 0) (5, 4, 3)	-3,5	2	35,72

Tableau 5.1 – Exemple de calcul de scores 2-LinBLEU (avec deux jeux de paramètres) et BLEU. La traduction de référence est *the big red apple . can I take it ?*

dans la table 5.1. L'exemple représente trois traductions différentes du corpus jouet composé des deux phrases suivantes :

la grande pomme rouge. puis-je la prendre ?

Dans le tableau on voit trois traductions de ce corpus de qualité différente. Nous constatons que la meilleure traduction (notons qu'elle coïncide avec la traduction de référence) a le score BLEU maximal de 100, puis, plus la traduction perd en qualité, moins son score BLEU est élevé [5]. Pour le score LinBLEU la tendance est opposée : plus la traduction est de bonne qualité, plus son score est bas. Notons que, contrairement au score BLEU, le score LinBLEU ne représente pas une valeur qui peut être interprétée de manière absolue. Notamment, la traduction de référence obtiens un score de -21 pour le premier jeu de paramètres et de -3 pour le deuxième jeu de paramètres, il est également possible d'avoir des paramètres assignant une valeur positive à la référence. Il est malheureusement difficilement envisageable de mettre ce score dans un cadre plus déterminé (par exemple, normalisé par le score de référence, ou transformé en pourcentage). En effet, contrairement au cas du score BLEU, le score LinBLEU n'est borné ni d'un côté, ni de l'autre : on peut imaginer des traductions ayant un score de $\pm\infty$ (ces traductions ayant aussi une longueur infinie). Cela implique aussi que ce score n'est applicable que dans des conditions où les hypothèses à évaluer sont déjà d'une longueur raisonnable, ce qui est en général le cas dans les treillis de recherche.

5. Les scores BLEU présentés ici ne sont pas réalistes. Pour un corpus de taille raisonnable traduit par un système classique on s'attend plutôt à des scores compris entre 15 et 40 suivant la paire de langues. La nécessité d'avoir au moins un 4-gramme commun sur un corpus aussi petit biaise ici les résultats.

5.2 Traduction automatique séquentielle avec SEARN

5.2.1 Traduction automatique séquentielle

Le principe général de la traduction automatique séquentielle est de décomposer le processus de traduction en une séquence d'actions correspondant chacune à une décision :

— sélection d'un segment de la phrase source ;
— choix de la traduction de ce segment ;
— placement de cette traduction dans la phrase cible.

Dans le cas général, le segment source représente un sous-ensemble des mots de la phrase source (non-traduits par les actions précédentes), sa traduction est un ensemble de mots de la langue cible et elle peut être intégrée à la phrase cible de manière arbitraire par rapport aux mots déjà traduits. La seule limitation générale sur la suite des actions formant une traduction est que chaque mot de la phrase source doit être traduit (cela signifie être inclus dans un segment à traduire) une et une seule fois. Par exemple, pour traduire la phrase *la grande pomme rouge* on peut effectuer les actions suivantes :

— *the big red apple* → *la grande*
— *the big red apple* → *la grande pomme rouge*

le terme de gauche représente la phrase à traduire, les mots soulignés étant le segment sélectionné pour la traduction, tandis que le terme de droite représente la traduction (partielle) produite, les mots soulignés étant la traduction du segment source placé par rapport aux mots déjà produits.

Les actions sont réalisées l'une après l'autre et le processus se termine lorsque toute la phrase source est traduite. La séquence d'actions est choisie de manière à construire la meilleure traduction possible en fonction des capacités du système.

Remarquons que dans le cas général, pour une phrase de plus d'un mot, il existe de nombreuses suites d'actions permettant d'obtenir chaque traduction. Voici, pour l'exemple considéré, une autre suite d'actions qui amène au même résultat :

— *the big red apple* → *la pomme rouge*
— *the big red apple* → *la grande pomme rouge*.

Réduction de l'ensemble des actions. Le processus de traduction que nous venons de décrire nécessite de choisir à chaque étape une action parmi un nombre gigantesque de choix possibles. Théoriquement, si l'on ne pose

aucune restriction sur la longueur des segments cibles produits, le nombre d'actions possibles est infini. Même en limitant la longueur de la phrase cible, on obtient un nombre d'actions impossible à traiter en pratique. Par exemple, si la longueur maximale pour un segment cible est L, le nombre d'actions possibles dans un état où il reste m mots à traduire est :

$$\sum_{n=1}^{m} C_m^n \left(\sum_{l=0}^{L} (|V|^l \right),$$

où $|V|$ est la taille du vocabulaire cible. Notons que, comme nous l'avons vu dans la section précédente, certains chemins conduisent à la même traduction, mais doivent quand même être considérés séparément.

Afin de rendre la complexité raisonnable, il est donc nécessaire d'imposer certaines restrictions sur l'ensemble des actions possibles. Ces restrictions peuvent être imposées aussi bien au niveau du choix du segment source à traduire qu'au niveau du choix de sa traduction ou de la manière de l'insérer dans la phrase cible. Par exemple, l'ensemble d'actions dans l'espace de recherche que nous allons considérer dans ce chapitre est construit avec les contraintes suivantes [6] :

- le segment source est un ensemble de mots consécutifs de la phrase source ;
- le segment traduit est intégré dans la phrase cible dans l'ordre gauche-droit ;
- l'ensemble des traductions possibles est restreint par la table de traduction.

Remarquons que les limitations sur l'ensemble d'actions peuvent amener à des conséquences différentes : certaines limitations font que l'ensemble des traductions pouvant être obtenues avec ces actions contient moins d'hypothèses ; d'autres ne font que réduire le nombre de manières d'obtenir une même hypothèse.

Potentiel des méthodes séquentielles. Les algorithmes exacts de recherche de la traduction ayant le meilleur score dans le treillis sont exposés à différentes contraintes. Tout d'abord, la fonction de score doit être décomposable, ce qui amène généralement à l'utilisation de fonctions linéaires. Ensuite, les caractéristiques doivent concerner des arcs voisins, dans le cas contraire l'espace de recherche exploserait comme cela a été montré dans la section 1.2.2. La complexité de la recherche exacte est une contrainte importante qui limite la taille de l'espace de

6. Ces contraintes sont propres à l'ensemble d'actions issu du système NCODE.

recherche considéré. La traduction séquentielle est une méthode alternative qui n'est pas concernée par les contraintes sur la classe de fonctions de score et les caractéristiques, ce qui permet potentiellement d'enrichir le modèle. Cette méthode présente également l'avantage d'avoir une complexité linéaire par rapport au nombre d'actions sortantes de l'état ce qui permet éventuellement d'explorer un espace de recherche plus grand. Ce potentiel est la principale motivation de l'étude de ce cadre de traduction action par action.

L'application des méthodes séquentielles à la traduction à été étudié récemment,par exemple dans [Durrani et al., 2011] à été étudié la possibilité de réaliser la traduction par un processus séquentiel intégrant les actions effectuant la traduction des mots et le réordonnancement. Cette idée a été ensuite développée dans [Guta et al., 2015] où l'apprentissage du modèle joint de traduction et de réordonancement est effectué à l'aide de réseaux de neurones. Notons que le principe de traduction séquentielle est aussi à la base des réseaux de neurones récurrents [Bahdanau et al., 2015].

5.2.2 Mise en œuvre de SEARN

La suite de ce chapitre porte sur l'utilisation de SEARN pour la traduction automatique séquentielle. Nous allons commencer par présenter dans cette section comment SEARN peut être appliqué aux treillis de recherche de NCODE.

Espace de recherche. Nous allons appeler l'*historique de taille n* associée à un chemin la suite des n derniers mots de la traduction partielle correspondant à ce chemin. L'espace de recherche de SEARN peut être déduit du treillis de NCODE de la manière suivante.

Dans le cas où toute la traduction partielle produite est utilisée pour informer les actions suivantes, l'ensemble des états possibles correspond à l'ensemble des traductions partielles que l'on peut trouver dans le treillis de NCODE. À l'exception des chemins produisant la même traduction partielle et qui peuvent être fusionnés, un état correspond à un chemin partiel. Dans le cas où l'historique utilisée est de taille limitée, les chemins ayant le même historique de cette taille peuvent également être fusionnés [7]. L'ensemble des actions réalisables dans un état correspond à l'ensemble des arcs sortants des derniers nœuds des chemins partiels correspondants à cet

7. En pratique, pour l'historique de longueur au moins 4, il n'y a pas de procédure de fusionnement car ces chemins sont déjà fusionnés dans les treillis NCODE, qui suit la règle de fusionnement des chemins lorsque les quatre dernier mots produits sont les mêmes. (Le cas de l'historique de plus petite longueur n'est pas considéré car sans perspectives.)

état [8]. L'état initial est l'état où aucun segment source n'est traduit, et les états finaux sont les états correspondants à des traductions complètes de la phrase source.

L'évaluation du système étant réalisée à l'aide du score BLEU, la somme des récompenses des actions composant un chemin devrait refléter, à une constante près, le score BLEU de la traduction correspondante à ce chemin. Pour calculer de telles récompenses, nous devrions trouver une décomposition du score BLEU sur les actions composantes la traduction. Mais, comme le montre la formule (5.1), cela n'est pas possible car le score BLEU n'est pas décomposable. Nous allons, comme dans la section 5.1.3, utiliser le score LinBLEU qui fournit une bonne approximation de BLEU et se décompose le long des chemins du treillis de traduction.

Pour déterminer la récompense associée au couple (s, a) où l'historique de l'état de départ est $e_{i=1}^{l-K}$ et l'action a apporte les K mots suivants : $e_{i=l-K+1}^{l}$, il suffit donc de faire le calcul suivant :

$$r(s,a) = \theta_0 + \sum_{k=1}^{K} \sum_{m=1}^{n} \theta_m o\left(e_{i=l-k-m+1}^{l-k}, \mathbf{r}\right) \tag{5.3}$$

où $e_{i\in I}$ est le m-gramme des mots consécutifs avec les indices en I, $o(e_{i\in I}, \mathbf{r})$ est l'indicateur de présence de m-gramme $e_{i\in I}$ dans la traduction de référence $\mathbf{r}$. La récompense associée au couple (état, action) représente alors l'apport dans le score LinBLEU des m-grammes finissants par les mots apportés par l'action considérée.

Pour notre première implémentation, nous avons optimisé les paramètres de LinBLEU de manière assez grossière [9] sur le corpus BTEC, un petit corpus français-anglais contenant des phrases courtes du domaine touristique [10].

Soient $\mathcal{G}_{\text{NCODE}}$ les treillis issus de la première phase de traduction de *NCODE* de l'ensemble de phrase en langue source $\mathcal{E} = \mathbf{e}_{j=1}^{J}$, ainsi que $\mathcal{R} = \mathbf{r}_{j=1}^{J}$ l'ensemble de traductions de référence. Nous appelons InitSEARN$(\mathcal{G}_{\text{NCODE}}, \mathcal{R})$ l'ensemble des espaces de recherche dont les 5 composants $\langle S, A, S_f, s_i, r\rangle$ sont obtenues comme c'est décrit ci-dessus.

Perte de potentiel et politique expert/tuteur. Comme nous avons vu dans le chapitre 3, SEARN nécessite un moyen de calculer les pertes de potentiel vis-à-vis d'une politique donnée. Le choix de l'action ayant la

8. Grâce au fait qu'il n'y a pas de procédure de fusionnement d'états, il suffit de prendre les actions sortant de l'état final unique des chemins considérés.

9. Le choix des paramètres de LinBLEU sera re-discuté dans la section 5.4.3.

10. *Basic Travel Expression Corpus*, http://iwslt2010.fbk.eu/node/32

perte de potentiel de zéro vis-à-vis de la politique expert ou autre permettra ensuite de construire la politique expert et tuteur respectivement.

Les pertes de potentielle vis-à-vis de la politique de l'expert peuvent être calculées de manière efficace pour tout le treillis à l'aide de l'algorithme Bellman-Ford [Bellman, 1958, Ford, 1956] ; la politique de l'expert en sera déduite de manière directe en choisissant l'action ayant la perte de potentiel de zéro [11]. Nous allons noter InitExp($\mathcal{G}$) la procédure de calcul de la politique expert qui renvoie la politique π_{exp} pour tous les espaces de recherche de $\mathcal{G}$.

Les pertes de potentiel associées à l'état s et l'action a vis-à-vis d'une politique stochastique se calculent en effectuant plusieurs simulations en utilisant la politique stochastique considérée [Daumé et al., 2009] ; la différence entre la récompense cumulée maximale parmi les actions sortantes de l'état s et la récompense cumulée moyenne associée à l'action a est considérée comme la perte de potentiel du couple (s, a). Ces calculs permettent de définir la politique tuteur vis-à-vis de la politique stochastique. Nous allons noter SimulTraj$(s, a, \pi; \#(\text{sim}))$ la fonction qui effectue $\#(\text{sim})$ simulations de trajectoires depuis l'état s commençant par l'action a et où les autres actions sont choisie à l'aide de la politique π ; cette fonction revoie la récompense cumulée moyenne associée à l'ensemble de trajectoires produites.

Dans notre configuration nous avons utilisé l'estimation simple de perte de potentiel vis-à-vis d'une politique stochastique en l'approximant par celle vis-à-vis de l'expert. Les expériences préliminaires ont montré que cette technique est suffisamment précise ; l'utilisation d'échantillons de trajectoires n'amène pas à l'amélioration des performances.

Classifieur sensible aux coûts et ensemble d'apprentissage. Le classifieur sensible aux coûts doit apprendre à choisir une action impliquant une perte de potentiel minimale vis-à-vis d'une certaine politique. Nous avons ici choisi d'approximer ce classifieur sensible aux coûts par un régresseur. Le principe de cette approximation est le suivant : la perte de potentiel de chaque action possible est estimée à l'aide du régresseur, puis l'action ayant la perte de potentiel estimée la plus faible est choisie en tant que résultat de la classification. Nous avons utilisé le régresseur linéaire

11. La question de l'expert non-déterministe peut se poser également pour cette tâche, par exemple dans la situation où il y a le choix de traduction d'un segment entier ou mot par mot. Nous avons utilisé la méthode d'adaptation de l'expert décrite dans la section 4.2.2.

fourni par la bibliothèque `sklearn`[12].

L'apprentissage de ce régresseur nécessite un ensemble d'apprentissage. La construction de cet ensemble découle directement de celui de SEARN : chaque état considéré dans l'ensemble d'apprentissage apporte autant d'exemples que d'actions disponibles dans cet état. Les caractéristiques associées à chaque exemple sont les caractéristiques $\dot{\phi}(s, a)$ associées au couple état-action correspondant ; son étiquette est la perte de potentiel associée. Les caractéristiques utilisées sont les mêmes que celles exploitées par le système NCODE. Néanmoins, afin de permettre au système de disposer de valeurs de modèles comparables et indépendantes de la longueur des segments, on utilise également des valeurs normalisées.

Mélange stochastique. Comme nous avons vu dans le chapitre 3, la politique produite par SEARN est stochastique ; elle est composée d'un ensemble de classifieurs, dont la règle d'application est définie par la formule telle que (3.28) ou (3.33). La stratégie spécifique de mélange est due au fait que la politique apprise doit changer lentement d'une itération à l'autre. En pratique nous avons appliqué le dernier classifieur appris à la place de toutes les composantes sauf l'expert (la probabilité de faire appel à celui-ci diminue après chaque itération) ; cela correspond à prendre dans les formules récursives (3.28) et (3.33) $\tau_1 ... \tau_{n-1}$ égaux à τ_n. Tous nos expériences montrent les performances équivalentes au cas d'utilisation des formules originelles.

Nous avons également définie la politique $\bar{\pi}$ comme une politique dans laquelle la composante expert est remplacée par la politique aléatoire. En pratique, il est plus intéressant de remplacer la composante aléatoire par un des classifieurs appris, généralement le dernier.

L'algorithme SEARN appliqué à la traduction automatique, tel que c'est décrit dans la présente section, est formalisé dans l'algorithme 12.

5.2.3 Premiers résultats

Dans ce chapitre, nous avons choisi d'évaluer notre système sur la traduction depuis l'anglais vers quatre langues : le russe, le tchèque, le roumain et le français. Les données utilisées sont issues des participations du LIMSI à des campagnes d'évaluation WMT des années précédentes. Toutes les données ont été proposées pour la tâche de la traduction automatique de textes journalistiques (la tâche principale de WMT). La table 5.2 fait

12. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Algorithme 12 : SEARN pour la traduction automatique. Les fonctions InitSEARN, InitExp et SimulTraj sont décrite dans la section présente, la fonction RandomMix et la politique $\bar{\pi}_N$ sont définies dans la section 3.5.2.

Données : ensemble de treillis issus de NCODE $\mathcal{G}_{\text{NCODE}}$, ensemble de traductions de référence $\mathcal{R}$, ensemble paramétrique Π, nombre d'itérations N, vitesse d'apprentissage β

Initialisation : $\mathcal{G} \leftarrow \text{InitSEARN}(\mathcal{G}_{\text{NCODE}}, \mathcal{R})$, $\mathcal{S} \leftarrow \emptyset$, $\pi_{\text{exp}} \leftarrow \text{InitExp}(\mathcal{G})$;
pour $n = 1..N$ **faire**
  pour $G = (S, A, s_i, S_f, r) \in \mathcal{G}$ **faire**
    $s \leftarrow s_i$;
    tant que $s \notin S_f$ **faire**
      pour $a \in A_s$ **faire**
        $\hat{Q}_a \leftarrow \text{SimulTraj}(s, a, \pi_{n-1}; \#(\text{sim}))$;
      fin
      $\hat{Q}_{\max} = \max_{a \in A_s}(\hat{Q}_a)$;
      pour $a \in A_s$ **faire**
        $\hat{l}(s, a; \pi_{n-1}) = \hat{Q}_{\max} - \hat{Q}_a$;
        $\mathcal{S} = \mathcal{S} \cup (\dot{\phi}(s, a), \hat{l}(s, a; \pi_{n-1}))$;
      fin
      $s \leftarrow s \circ \pi_{n-1}$;
    fin
    Apprendre τ_n résolvant $\mathcal{S}$ (via le régresseur ρ_n);
    $\pi_n = \text{RandomMix}(\pi_{\text{n}-1}, \tau_{\text{n}}, \beta)$;
  fin
fin
retourner $\bar{\pi}_N$;

le bilan des différents corpus de développement et de test utilisés dans la construction des systèmes présentés dans ce chapitre.

Système	developement	test
anglais-russe [13]	6000	2981
anglais-thèque [14]	3057	2773
anglais-roumain [15]	1000	999
anglais-français [16]	2525	2489

Tableau 5.2 – Taille en nombre de phrases des corpus de développement et de test utilisés.

Quelques exemples de phrases parallèles issues des corpus utilisées sont présentées dans le tableau 5.3.

(russe) поездка в Коста - Рику - это сбывшаяся мечта и для нас тоже . going to Costa Rica is a dream come true for us too .
(tchèque) a čím více upřímně jsem mluvila , tím více jsem byla žádána mluvit . and the more candidly I spoke , the more I was asked to speak .
(roumain) « Vă primim cu bratele deschise » , a răspuns Felipe . « we welcome you with open arms , » Felipe replied .
(français) il n' est pas certain que l' État puisse agir sans cet argent . it is still not clear if the state can get by without this money .

Tableau 5.3 – Quelques exemples de phrases parallèles issues des corpus utilisés.

Le détail des données, des différentes étapes de leur préparation ainsi que de la construction des systèmes NCODE de référence peut-être trouvé dans les articles décrivant les systèmes de traduction soumis par le LIMSI [Allauzen et al., 2013, Allauzen et al., 2016].

L'algorithme SEARN tel qu'il est décrit dans cette section a été implémenté et testé sur ces quatre corpus parallèles. Dans la suite de ce chapitre nous les appelons selon leur langue cible : russe, thèque, roumain

13. Les données de test de 2012, 2013 et 2014 ont été mélangées et partagées entre les deux corpus.

14. Les données de test de 2010 et 2011 forment le corpus de dévelopement tandis que celle de 2012 et 2013 forment le corpus de test.

15. Les données de test de 2016 à été partagé entre les deux corpus.

16. Les données de test de 2009 sont utilisées pour le corpus de dévelopement et celles de 2010 pour le corpus de test.

	Russe	Tchèque	Roumain	Français
MIRA/MERT	25,11	18,74	28,12	30,47
SEARN	14,80	13,04	18,43	16,95
ORACLE	42,19	37,63	47,50	48,92

Tableau 5.4 – Premiers résultats en BLEU comparant MIRA (pour le russe, le tchèque et le roumain)/MERT (pour le français) et SEARN, ainsi que les scores des traductions de l'oracle.

et français. NCODE a été utilisé afin de réaliser la première phase de traduction et fournir les treillis d'hypothèses pour les quatre corpus. Les poids associés à chaque modèle pour le système NCODE ont étés optimisés sur un ensemble de développement à l'aide de l'algorithme MIRA, utilisé comme système de référence pour les langues russe, tchèque, roumain et MERT pour le français.

Le tableau 5.4 présente les résultats obtenus par cette première implémentation de SEARN ainsi que par le système de référence et l'oracle. Premièrement, on peut voir que les résultats de l'oracle sont significativement meilleurs que ceux du système de référence. Cela signifie que le système a encore un potentiel d'amélioration sérieux lors de la deuxième phase de traduction, et il est donc intéressant d'explorer les possibilités de perfectionnement de l'algorithme de recherche de la meilleure traduction dans le treillis. Ensuite, on voit que les performances de SEARN sont très largement inférieures à celles du système de référence. Le reste de ce chapitre sera dédié à l'analyse de ce premier résultat négatif.

5.3 Les étapes de l'analyse du système

Pour comprendre l'origine des résultats décevants de la section précédente nous allons dissocier les différentes étapes du processus d'apprentissage avec SEARN et les évaluer séparément. Chacune de ces étapes caractérise la capacité du système à résoudre un problème donné, du plus simple au plus complexe. Les performances de ces différents niveaux sont liées ; si un problème sérieux est détecté dans un des niveaux inférieurs, il est quasiment certain que les niveaux plus élevés ne pourront obtenir de bons résultats.

Dans les sections suivantes nous allons présenter des stratégies d'évaluation des quatre grandes étapes suivantes :

1. *Performances en régression :* capacité à évaluer la perte de potentiel associée à une action ;

2. *Performances en classification :* capacité à choisir une bonne action dans un état donné ;
3. *Évaluation en LinBLEU :* capacité à choisir une suite complète de bonnes actions ;
4. *Évaluation en BLEU :* évaluation finale ;

Cette construction par étapes ne nous garantit pas de pouvoir déterminer de manière exacte la cause des mauvais résultats observés mais elle devrait nous permettre de mieux les comprendre et nous donner des pistes afin d'améliorer notre système.

5.3.1 Système et politique de référence

Afin de comparer SEARN avec le système de référence, nous allons définir la politique de référence, la politique en utilisant laquelle la traduction de référence est construite. L'utilisation de cette politique nous permettra de comparer ces deux systèmes plus finement que ne le permet une simple comparaison des scores BLEU.

Définir une politique signifie de définir une règle universelle permettant de choisir une action dans un état donné, ce qui est en général réalisé à l'aide d'une fonction de score. Les scores donnés aux actions par le système de référence sont donnés de manière à ce que le score total du chemin soit tant meilleur que la traduction correspondante a le meilleure BLEU. Pour transformer ces scores locaux en *scores globaux* qui peuvent définir la politique (c'est-à-dire que la politique choisit dans chaque état l'action ayant le meilleur score), il suffit d'appliquer l'algorithme de Bellman-Ford, comme pour le calcul des pertes de potentiel expliqué dans la section 5.2.2. Définie de cette manière, la *politique du système de référence*, ou simplement *la politique de référence*, choisit bien la même traduction que le système de référence, et dans les états qui ne sont pas sur son chemin, elle choisit la solution ayant le meilleur score possible du chemin complet selon le système de référence.

5.3.2 Première étape : Performances en régression

La première étape consiste à évaluer la performance du régresseur. Apprendre un régresseur permettant de bien évaluer les pertes de potentiel des différentes actions est crucial car c'est sur ces estimations que le choix des actions se base ensuite. De bons choix sont indispensables afin de construire une traduction de bonne qualité.

Soit $\mathcal{E}$ l'ensemble de couples (état, action) ; à la n-ème itération la qualité Q_{ρ_n} du régresseur ρ_n est estimée en observant la moyenne des écarts entre la perte de potentiel $l(s, a, \pi_{n-1})$ [17] et son estimation prédite par le régresseur $\hat{l}_{\rho_n}(s, a, \pi_{n-1})$ pour tous les couples $(s, a) \in \mathcal{E}$:

$$Q_{\rho_n} = \frac{1}{|\mathcal{E}|} \sum_{(s,a)\in\mathcal{E}} |l(s, a, \pi_{n-1}) - \hat{l}_{\rho_n}(s, a, \pi_{n-1})|. \tag{5.4}$$

De mauvaises performances pour un régresseur peuvent être expliquées par les difficultés de l'apprentissage, telles que :

— une insuffisance d'information dans les caractéristiques ;
— une classe de fonctions considérée trop pauvre ;
— trop ou trop peu de généralisation ;
— erreur d'optimisation

Les deux premiers points étant deux aspect d'un même problème : l'absence d'une fonction séparatrice, soit par des arguments insuffisants, soit par une classe de fonctions trop restreinte.

Notons que la qualité du régresseur n'est importante que pour une évaluation interne du système, c'est-à-dire afin de comparer différentes implémentations de SEARN (par exemple, différents ensembles de caractéristiques ou différentes constantes de régularisation). En effet, l'estimation des pertes de potentiel de chaque action n'a d'importance que pour pouvoir ensuite résoudre le problème de classification et n'est pas nécessaire de manière générale. La politique de référence dispense de cette tâche intermédiaire, elle choisit ses actions sans attribuer de scores qui peuvent être interprétés comme les pertes de potentiel en score LinBLEU. Il n'est donc pas possible de comparer SEARN et le système de référence à ce niveau.

5.3.3 Deuxième étape : Performances en classification

L'étape suivante est l'évaluation des performances en classification ; ces performances sont définies par la capacité du classifieur à choisir dans un état donné une action qui amènera à une perte de potentiel minimale. L'enchainement de bonnes actions permet ensuite de construire une traduction de qualité.

Pour un ensemble d'états $\mathcal{S}$ à la n-ème itération la qualité du classifieur τ_n s'estime simplement en calculant la moyenne des pertes de potentiel des

17. Dans notre cas pratique, comme nous avons utilisé l'approximation de la perte de potentiel vis-à-vis d'une politique stochastique par celle vis-à-vis de l'expert, nous allons évaluer la capacité du régresseur de prédire la quantité $l(s, a, \pi_{\text{exp}})$. Cela est également vrai pour les étapes d'évaluation suivantes.

actions choisies par le classifieur τ_n dans chacun des états $s \in \mathcal{S}$:

$$Q_{\tau_n} = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} l(s, \tau_n(s), \pi_{n-1}) \tag{5.5}$$

Une évaluation externe est ici possible en mesurant la perte de potentiel moyenne de la politique apprise par SEARN et de la politique de référence π_{ref}.

Notons qu'afin de comparer les performances, il est nécessaire de choisir un ensemble d'états sur lequel la perte totale sera calculée. Il peut donc se trouver qu'un système soit meilleur que l'autre pour un ensemble donné, mais moins bon pour un autre ensemble. Par exemple un système peut être meilleur en moyenne sur tous les états de l'espace de recherche, mais moins bon sur l'ensemble des états composant le chemin d'inférence. À cause de gaspillage de ressources, il sera moins bon de manière générale.

Dans ces conditions, un résultat négatif correspondra au cas où l'on constate que le classifieur est moins bon que le système de référence sur le chemin d'entrainement du classifieur (obtenu par la même politique que celle utilisée lors de l'entrainement). On peut en déduire que la classe des fonctions considérée ou l'ensemble de caractéristiques utilisé sont moins puissants que celui du système de référence. Un résultat positif correspond au cas où l'on observe de meilleures performances de la part du classifieur sur un ensemble quelconque, par exemple sur le chemin de la politique de référence. Ce cas serait un résultat encourageant mais indiquerait la nécessité de poursuivre l'évaluation.

5.3.4 Troisième étape : Performance en classification sur le chemin d'inférence

De bonnes performances des classifieurs τ_n composant une politique stochastique (3.28) ou (3.33) évaluées sur le chemin d'apprentissage de chacun des classifieurs est un bon signe mais ne permet pas d'évaluer les performances du système au niveau de la traduction produite. Par exemple, comme nous l'avons vu dans la section 3.2.1, de bonnes performances en classification sur les états visités par l'expert signifient uniquement que le classifieur est capable de correctement résoudre les problèmes rencontrés par l'expert un par un. Dans le cas où une erreur se produit, on ne peut rien dire sur ses performances pour les décisions suivantes.

Au chapitre 3 nous avons vu l'importance d'apprendre sur un chemin qui soit le plus proche possible du chemin qui sera utilisé lors de l'inférence. Néanmoins, il n'est en pratique pas réalisable d'apprendre sur le chemin

de l'inférence ; il est donc nécessaire de tenir compte d'un éventuel effet d'avalanche des erreurs. Pour évaluer à quel point notre système souffre de cet effet, il est possible de comparer le taux d'erreur de la politique $\bar{\pi}_n$ construite à la n-ème itération (sans appel à l'expert) sur le chemin d'apprentissage et sur le chemin obtenu en utilisant la politique $\bar{\pi}_n$ elle-même.

La longueur du chemin L pouvant varier, on utilisera une mesure de la qualité normalisée :

$$Q_{\bar{\pi}_n} = \frac{1}{|\mathcal{S}_{\bar{\pi}_n}|} \sum_{s \in \mathcal{S}_{\bar{\pi}_n}} l(s, \bar{\pi}_n(s), \pi_{n-1}) \tag{5.6}$$

où $\mathcal{S}_{\bar{\pi}_n}$ est l'ensemble d'états visités par la politique $\bar{\pi}_n$.

De même que pour la deuxième étape, une évaluation externe est possible en comparant les résultats obtenus par notre système avec ceux du système de référence en utilisant la politique de référence π_{ref}. Notons que la quantité (5.6) évaluée sur le chemin $\mathcal{S}_\pi$ multipliée par la longueur de ce chemin $|\mathcal{S}_\pi|$ représente la différence entre le score LinBLEU de la traduction construite à partir de ce chemin et le score LinBLEU de la traduction de l'oracle. Cette évaluation est alors équivalente à la simple comparaison des scores LinBLEU des traductions obtenues par les deux politiques. Néanmoins, nous préférons d'étudier les deux quantités, car (5.6) peut être facilement comparé avec (5.5), ce qui permet de vérifier que la bonne qualité des classifieurs se transforme en bonne qualité de la politique stochastique.

5.3.5 Quatrième étape : Évaluation en BLEU

La quatrième étape d'évaluation est l'évaluation en score BLEU. Cette étape étant la dernière, les raisons qui peuvent amener à de mauvaises performances ici sont les plus variées. Les approximations réalisées lors des étapes précédentes sont ici accumulées et la contribution de chacune d'entre elle devient difficile à évaluer.

Nous nous intéresserons plus particulièrement à la dernière approximation que nous n'avons pas encore évaluée : l'approximation du score BLEU par une métrique décomposable utilisée pour calculer la politique de l'expert (le score LinBLEU dans notre cas). On s'attachera ici à vérifier la cohérence entre le BLEU et la métrique choisie et contrôler qu'un système imitant bien l'expert (qui a un bon score LinBLEU) a aussi un bon score BLEU par rapport au système de référence. Si ce n'est pas le cas, il sera alors nécessaire de revoir notre utilisation du LinBLEU.

5.4 Évaluation

Dans la section précédente nous avons présenté les différentes étapes nécessaires à la compréhension des résultats nettement inférieurs au système de référence de notre premier système de traduction utilisant SEARN. Nous allons maintenant présenter les résultats de ces analyses et les améliorations possibles de ce système.

5.4.1 Première évaluation

Le tableau 5.4.1 présente les résultats détaillés des quatre étapes d'évaluation proposées pour le système développé dans la section précédente. La colonne « Régr. » donne l'erreur moyenne 5.4 par décision pour SEARN uniquement puisque cette mesure ne peut pas être calculée pour le système de référence ou l'oracle. La colonne « Classification SC » (sous-colonne gauche) indique l'erreur moyenne 5.5 sur le chemin obtenu à l'aide de la politique d'apprentissage. La colonne « Classification SC » (sous-colonne droite) présente l'erreur moyenne 5.6 sur le chemin choisi par SEARN et donc obtenu à l'aide de la politique évaluée. Pour la classification sensible aux coûts on donne également le pourcentage de réponses correctes (ayant le coût de 0). Enfin, les deux dernières colonnes indiquent les scores LinBLEU et BLEU de la traduction obtenue. Tous les résultats sont donnés pour la 15-ème itération de SEARN.

Performances en classification On constate que dès la deuxième étape d'évaluation (score de classification sur le chemin construit à l'aide de la politique d'apprentissage) les résultats sont inférieurs à ceux du système de référence. Après avoir exclus les problèmes liés à la qualité d'apprentissage (sur-apprentissage, mauvais réglage de paramètres), nous pouvons constater que l'expressivité de notre modèle est plus faible que celle du système de référence.

Rôle de la structure des treillis En observant la structure des treillis on constate que le résultat obtenu est attendu. Pour les tâches présentées au chapitre précédent, la structure des treillis était telle que le choix d'une action n'ait pas d'impact sur les actions possibles sur la suite du chemin et amène donc à une perte bien définie. Cela a pour conséquence qu'il est toujours possible pour le système de retrouver un chemin de bonne qualité après une erreur (même si en pratique cela peut être difficile car les erreurs peuvent se propager).

Langue	Système	Régr.	Classification SC		LinBLEU	BLEU
Russe	Oracle	-	0.0 (100%)	0.0 (100%)	2.82	42.19
	MIRA	-	0.85 (71%)	0.85 (71%)	17.72	25.11
	SEARN	2.12	1.19 (66%)	1.19 (66%)	18.22	14.86
Tchèque	Oracle	-	0.0 (100%)	0.0 (100%)	5.10	37.63
	MIRA	-	0.83 (68%)	0.83 (68%)	16.62	18.74
	SEARN	1.95	1.11 (65%)	1.11 (65%)	16.71	13.03
Roumain	Oracle	-	0.0 (100%)	0.0 (100%)	1.75	47.50
	MIRA	-	0.89 (70%)	0.89 (70%)	18.54	28.12
	SEARN	2.15	1.24 (65%)	1.24 (65%)	19.52	18.54
Français	Oracle	-	0.0 (100%)	0.0 (100%)	-0.22	48.92
	MERT	-	0.89 (70%)	0.90 (69%)	18.03	30.47
	SEARN	2.28	1.45 (60%)	1.46 (60%)	20.51	17.01

Tableau 5.5 – Résultats détaillés des quatre étapes d'évaluation proposées pour le système développé dans la section précédente. Les colonnes 3 à 7 représentent les 5 étapes (dont une sous-étape) d'évaluation : performances en régression, performances en classification sensible aux coûts sur le chemin d'apprentissage/sur le chemin d'inférence, évaluation en LinBLEU et en BLEU. Les poids correspondants aux $0, 1, 2$-grammes dans LinBLEU sont respectivement 1, -0.5 et -1.

Les treillis construits par NCODE n'ont pas cette propriété : certaines actions induisent des puits car elles entrainent le décodeur dans des zones du treillis qui ne contiennent aucun chemin correspondant à une bonne traduction. Une recherche exacte dans le treillis va pouvoir prendre en compte le potentiel des chemins qui suivent une action et donc éviter de tomber dans ces puits. Cette information sur le futur est donc cruciale et manque au décodage glouton. La politique du système de référence utilisant un décodage exact a donc accès à cette information globale ce qui lui donne un avantage important par rapport à SEARN.

Un exemple d'action qui amène dans un puits peut être trouvé dans le treillis NCODE correspondant à la traduction en russe de la phrase

that number was reached Friday before the game against Panama...
эта цифра была достигнута в прошлую пятницу в игре против Панамы...

La traduction proposée par l'oracle sur le treillis complet est très proche de la référence, il y manque seulement une traduction de *before* :

эта цифра была достигнута в пятницу в игре против Панамы...

Le pronom *that* est ici correctement traduit par le pronom indicatif au nominatif : *эта*. Une autre action de traduction possible dans le treillis de recherche est de traduire *that* par le même pronom mais à l'accusatif : *эту*. Cette erreur amène le décodeur dans une zone du treillis où aucun chemin proche de la référence n'existe, l'oracle obtenu après cette décision contient de nombreuses erreurs (les portions rouges représentent les parties erronées de la traduction) :

эту цифру было достигнуто в пятницу в преддверии матча против Панамы...

L'erreur dans la déclinaison du pronom se propage au nom suivant qui est lui aussi mal décliné, les erreurs suivantes sont par contre indépendantes et liées au filtrage du treillis.

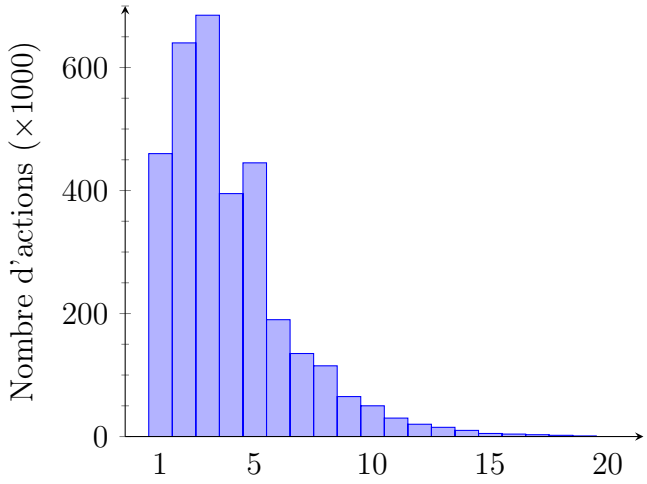
La figure 5.4.1 montre la répartition des pertes de potentiel pour l'ensemble des treillis du corpus de test. Dans un treillis régulier [18] une mauvaise action amènerait à une perte maximale de 5 points LinBLEU, alors qu'ici on peut observer qu'une importante part des actions est associée à des pertes supérieures pointant vers les irrégularités ou puits.

Solutions envisagées Afin de corriger ce problème, une solution possible est d'augmenter le nombre d'hypothèses considérées et diminuer ainsi la quantité de puits dans le treillis. Dans l'idéal on souhaiterait considérer l'espace de recherche régulier et non pas le treillis élagué par la recherche en faisceau réalisée par NCODE. Cette solution permettrait de créer des conditions plus adaptées à un décodage glouton tout en conservant une complexité linéaire. Elle nécessiterait par contre de réimplémenter tout le processus de construction de l'espace de recherche actuellement réalisé par NCODE, avant de mettre en œuvre cette solution radicale et il est pertinent de poursuivre l'évaluation du système actuel avant de l'envisager.

Une solution alternative, que nous développons dans la section suivante, consiste à utiliser les informations sur les futures possibilités de traductions.

18. On considère ici comme étant régulier un treillis théorique représentant des hypothèses de traduction de même longueur avec à chaque état l'ensemble des mots du vocabulaire cible possible comme action. Cette contrainte de longueur est peu naturelle pour le cas de la traduction mais permet d'établir les pertes de manière homogène. Les hypothèses de traduction présentes dans le treillis sont relativement peu variées en longueur ($\pm 10\%$), cette contrainte ne perturbe donc que peu l'évaluation.

FIGURE 5.3 – Le répartition des pertes de potentiel (vis-à-vis de l'expert) des actions pour l'ensemble des treillis du corpus de test.



5.4.2 Deuxième évaluation

Utilisation du coût futur comme une caractéristique supplémentaire Dans la suite de ce travail nous adoptons la solution suivante : nous utilisons les *scores globaux* des actions donnés par le système de référence (définis dans la section 5.3.1) en tant que caractéristiques supplémentaires permettant de juger sur les perspectives offertes par les différentes actions. Cette solution est triviale dans le sens où il n'est pas nécessaire d'effectuer de calculs supplémentaires pour l'implémenter, et en même temps elle peut nous permettre d'étudier d'autres aspects de notre système et ainsi découvrir d'autres problèmes éventuels avant de chercher le bon compromis entre l'élargissement de l'espace de recherche et l'élargissement du champ de vision. Les résultats correspondants se trouvent dans le tableau 5.4.2.

On peut voir ici que les résultats en régression et en classification sont significativement meilleurs par rapport au système précédent aussi bien que par rapport au système de référence. On peut donc considérer que cette étape d'évaluation est réussie.

Remarquons aussi que l'ajout du coût futur dans l'ensemble de caractéristiques fait plus que donner la connaissance sur le futur. Cette caractéristique nous assure qu'à la fois l'ensemble de caractéristiques et la classe de fonctions considérées sont suffisants pour atteindre les résultats du système de référence. Si l'algorithme ne trouve pas une solution équivalente

Langue	Système	Régr.	Classification SC		LinBLEU	BLEU
Russe	Oracle	-	0.0 (100%)	0.0 (100%)	2.82	42.19
	MIRA	-	0.85 (71%)	0.85 (71%)	17.72	25.11
	SEARN	1.94	0.80 (71%)	0.80 (71%)	16.72	24.30
	diff.	-0.18	-0.39 (+5%)	-0.39 (+5%)	-1.50	+9.44
Tchèque	Oracle	-	0.0 (100%)	0.0 (100%)	5.10	37.63
	MIRA	-	0.83 (68%)	0.83 (68%)	16.62	18.74
	SEARN	1.82	0.80 (69%)	0.80 (69%)	16.13	18.55
	diff.	-0.13	-0.31 (+4%)	-0.31 (+4%)	-0.58	+5.52
Roumain	Oracle	-	0.0 (100%)	0.0 (100%)	1.75	47.50
	MIRA	-	0.89 (70%)	0.89 (70%)	18.54	28.12
	SEARN	2.00	0.84 (71%)	0.84 (71%)	17.39	27.91
	diff.	-0.15	-0.40 (+6%)	-0.40 (+6%)	-2.13	+9.37
Français	Oracle	-	0.0 (100%)	0.0 (100%)	-0.22	48.92
	MERT	-	0.89 (70%)	0.90 (69%)	18.03	30.47
	SEARN	2.08	0.85 (70%)	0.85 (70%)	16.86	29.88
	diff.	-0.20	-0.60 (+10%)	-0.61 (+10%)	-3.65	+12.87

Tableau 5.6 – Résultats détaillés des quatre étapes d'évaluation proposées pour le système intégrant les coûts futurs. Les colonnes 3 à 7 représentent les 5 étapes (dont une sous-étape) d'évaluation : performances en régression, performances en classification sur le chemin d'apprentissage/sur le chemin d'inférence, évaluation en LinBLEU et en BLEU. Les poids correspondants aux $0, 1, 2$-grammes dans LinBLEU sont respectivement 1, -0.5 et -1. Pour la classification, on donne le pourcentage de bonnes réponses en plus de la perte moyenne. (entre parenthèses) La dernière ligne de chaque bloc (« diff ») représente les améliorations par rapport aux résultats obtenus par le système précédent n'intégrant pas les coûts futurs.

au système de référence, le problème provient d'une autre source que de la pauvreté d'informations ou la pauvreté de la classe de fonctions.

Effet d'avalanche L'étape suivante est l'évaluation des performances en classification sur le chemin construit par la politique en train d'être évaluée. Comme il a été dit dans la section précédente, la comparaison de ce résultat avec le précédent permet de conclure sur l'effet d'avalanche.

La comparaison des résultats des deuxième et troisième colonnes montre des performances équivalentes pour notre système. Cela indique que l'effet d'avalanche est ici absent.

Évaluation de cohérence des métriques La comparaison des scores LinBLEU et BLEU, par contre, met en évidence un nouveau problème. De manière systématique notre système obtient des scores LinBLEU meilleurs que le système de référence, tandis que l'évaluation en BLEU montre un résultat opposé. Il y a donc ici un problème de cohérence entre les deux métriques.

5.4.3 Troisième évaluation

Les résultats précédents ont montré que la métrique optimisée, LinBLEU, est insuffisamment corrélée avec le score BLEU utilisée pour l'évaluation. Une observation des oracles obtenus à l'aide de LinBLEU nous permet de constater qu'ils ont une forte tendance à être significativement plus courts que les traductions de système de référence. Le système SEARN apprenant à reproduire ces oracles a une nette tendance à produire lui aussi des traductions courtes, son évaluation finale est donc fortement impactée par la pénalité de brièveté.

Cette observation nous amène à remettre en question les paramètres utilisés dans le calcul du LinBLEU. Les paramètres choisis : $(1, -0.5, -1)$, ont été optimisés sur le corpus BTEC et n'ont pas été modifiés ensuite. Les oracles obtenus sont de bonne qualité comme le montrent les scores des tableaux précédents mais ils sont probablement inadaptés pour notre application. Le corpus BTEC est constitué principalement de phrases courtes et simples à traduire, la proportion d'unigrammes et de bigrammes communs y est donc importante. Dans les corpus utilisés ici pour l'évaluation de notre système, les phrases à traduire sont nettement plus longues et plus complexes à traduire. La proportion d'unigrammes et de bigrammes communs est donc plus faible et les mêmes poids ne peuvent compenser

correctement la pénalisation donnée aux 0-grammes, ce qui amène à des traductions oracles très courtes.

Pour valider cette hypothèse, les paramètres du LinBLEU ont été réoptimisés pour chaque langue sur des données issues des corpus d'évaluation et les résultats sont présentés dans le tableau 5.4.3. Ces nouveaux paramètres permettent d'obtenir des oracles d'une taille plus proche de la référence et donc un système SEARN produisant lui aussi des traductions plus longues. Le score final est donc beaucoup moins impacté par la pénalité de brièveté et l'on peut voir que notre système obtient maintenant des résultats de même niveau que ceux de système de référence.

5.5 Conclusion et perspectives

Dans ce chapitre nous avons présenté une adaptation de SEARN pour la traduction automatique ainsi qu'une méthodologie d'évaluation et de diagnostique adaptée. Différents instruments d'évaluation des problèmes d'apprentissage ont été utilisés afin de mettre en évidence les problèmes standards d'apprentissage (sur-apprentissage, manque d'information, classe des fonctions insuffisante, algorithme d'optimisation sous-optimal) aussi bien que les problèmes plus spécifiques de SEARN (effet d'avalanche, inexactitude de l'oracle). Cette évaluation méthodique nous a permis de déterminer et de corriger les raisons des mauvaises performances initiales de notre système et d'obtenir au final des résultats comparables à ceux du système de référence. Ces résultats doivent cependant être relativisés, notre système final reposant toujours sur les caractéristiques de coûts futurs dont la complexité de calcul est équivalente à la recherche du plus court chemin.

Les applications de SEARN à d'autres tâches présentées au chapitre précédent nous ont montré que ces coûts futurs ne sont pas indispensables si le treillis est suffisamment régulier. Une première perspective de poursuite de ces travaux est donc de vérifier que l'utilisation de SEARN sur le treillis complet pour la traduction permet de se passer des coûts futurs et donc de pouvoir pleinement bénéficier des avantages de la recherche gloutonne.

La recherche gloutonne est un avantage net de SEARN par rapport au décodage exact ou en faisceau réalisé classiquement par les décodeurs des systèmes de traduction automatique. Cette approche ouvre de nombreuses perspectives d'amélioration du système actuel car il est possible d'intégrer des caractéristiques tirant profit de l'historique complet des décisions et non d'un ensemble limité de décision locales. De telles caractéristiques pourraient permettre à SEARN de disposer d'un avantage important par rapport au système de référence et ainsi d'améliorer les résultats du système actuel.

Langue	Système	Régr.	Classification SC		LinBLEU	BLEU
Russe	Oracle	-	0.0 (100%)	0.0 (100%)	-52.81	42.77
$\theta = [1, -1.5, -2]$	MIRA	-	1.74 (67%)	1.74 (67%)	-22.15	25.11
	SEARN	4.38	1.75 (67%)	1.75 (67%)	-22.08	24.97
	diff.	-	-	-	-	+0.67
Tchèque	Oracle	-	0.0 (100%)	0.0 (100%)	-34.81	36.71
$\theta = [1, -1.5, -2]$	MIRA	-	1.72 (66%)	1.72 (65%)	-10.80	18.74
	SEARN	4.18	1.72 (66%)	1.72 (66%)	-10.83	18.78
	diff.	-	-	-	-	+0.23
Roumain	Oracle	-	0.0 (100%)	0.0 (100%)	-29.74	45.32
$\theta = [1, -1.25, -0.5]$	MIRA	-	1.84 (63%)	1.84 (63%)	5.00	28.12
	SEARN	4.04	1.81 (63%)	1.82 (63%)	4.32	28.41
	diff.	-	-	-	-	+0.50
Français	Oracle	-	0.0 (100%)	0.0 (100%)	-21.57	49.15
$\theta = [1, -1, -1]$	MERT	-	1.05 (73%)	1.05 (73%)	-0.23	30.47
	SEARN	2.59	1.04 (74%)	1.04 (74%)	-0.43	30.35
	diff.	-	-	-	-	+0.47

Tableau 5.7 – Résultats détaillés des quatre étapes d'évaluation proposées pour le système utilisant les nouveaux oracles. Les colonnes 3 à 7 représentent les 5 étapes (dont une sous-étape) d'évaluation : performances en régression, performances en classification sur le chemin d'apprentissage/sur le chemin d'inférence, évaluation en LinBLEU et en BLEU. Les poids correspondants aux $0, 1, 2$-grammes dans LinBLEU sont donnés pour chaque langue dans la première colonne. Pour la classification, on donne le pourcentage de bonnes réponses en plus de la perte moyenne. (entre parenthèses) La dernière ligne de chaque bloc (« diff ») représente les améliorations par rapport aux résultats obtenus par le système précédent utilisant les paramètres θ par défaut. On ne donne pas de différences pour les étapes d'évaluation qui utilisent LinBLEU, car avec les nouveaux paramètres $[\theta_0, \theta_1, \theta_2]$ les résultats ne sont pas comparables.

Conclusion

Les travaux présentés dans ce manuscrit ont été consacrés à l'étude des méthodes de la famille « apprendre à chercher » dans le cadre de différentes tâches du traitement automatique des langues. Les chapitres 1 et 2 ont été dédiés à un état de l'art des méthodes d'apprentissage et à la présentation des tâches du TAL et de leur spécificités. Nos contributions ont été détaillées dans les chapitres 3, 4 et 5.

Au chapitre 3, nous avons réalisé une étude détaillée des méthodes « apprendre à chercher » existant dans la littérature et des garanties théoriques qui leur sont associées. Les méthodes AGGRAVATE et SEARN, présentées dans les dernières sections de ce chapitre, représentent actuellement l'état de l'art de cette famille de méthodes. Lors des expériences préliminaires, ces deux méthodes ont obtenu des résultats similaires et nous avons choisi de concentrer notre présentation sur SEARN qui, après son adaptation au cas de séquences d'actions finies proposée dans la section 3.5.2.1, demande relativement peu d'itérations afin de limiter l'effet d'avalanche.

L'étude de l'application de SEARN aux différentes tâches d'étiquetage de séquences du traitement automatique des langues a fait l'objet du chapitre 4. Ces tâches ont pour avantage d'avoir à la fois une structure et des procédures d'évaluation simples, ce qui permet une application directe des différentes configurations de SEARN. Nous avons aussi étudié l'impact sur les performances des différentes manières de construire la séquence recherchée (les différents ordres possibles d'attribution des étiquettes). Nous avons pu observer qu'exploiter ce degré de liberté supplémentaire afin de placer en priorité les étiquettes les plus simples a globalement une influence très positive sur les performances. Il faut toutefois noter que cela entraine une accroissement de la complexité et donc du temps de traitement.

Le chapitre 5 a été consacré à l'application de SEARN à la traduction automatique, une tâche complexe qui apporte des difficultés supplémentaires par rapport aux tâches du chapitre précédent, aussi bien au niveau de la structure que des méthodes d'évaluation. Nous avons présenté une adapta-

tion de SEARN pour cette tâche ainsi qu'une chaine d'évaluation permettant d'identifier les problèmes qui peuvent se poser à chacune des étapes et avoir un impact sur les performance finales, tels que :

— la difficulté de prédire ce que nous avons appelé « perte de potentiel », qui mesure l'effet sur la performance globale des choix effectués localement ;
— les effets liés aux différences entre les chemins rencontrés lors de l'apprentissage et lors du décodage ;
— la non-corrélation des métriques d'évaluation entre elles.

Nous avons pu observer que dans le cas de la traduction le principal problème vient de la difficulté de prédire les pertes de potentiel liée à la présence des puits dans l'espace de recherche. L'autre difficulté rencontrée, la faible corrélation entre les métriques BLEU et LinBLEU, peut être mitigée à l'aide d'un choix soigneux des paramètres de cette dernière, tandis que les différences entre les chemins entre l'apprentissage et l'inférence ne semblent pas poser de problèmes dans les configurations que nous avons étudiées.

Ces différentes études de SEARN nous permettent d'apporter des éléments de réponse à la question du compromis entre l'approximation introduite au niveau du modèle et celle introduite au moment de la recherche de la meilleure solution selon un modèle.

L'approximation au moment de la recherche permet d'exploiter plus de dépendances et de réaliser le décodage dans un ordre libre. Nous avons pu observer que ces informations et degrés de libertés supplémentaires profitent à certaines tâches et que les performances obtenues sont supérieures à celles des méthodes choisissant un décodage exact pour un modèle approximé. Les dépendances supplémentaires sont naturellement utiles dans le cas où elles apportent une information nouvelle par rapport aux dépendances locales comme dans le cas de la tâche de prédiction de la prononciation [19].

Dans le cas de la traduction automatique, la nature des puits présents dans les treillis semble laisser peu d'espoir sur la possibilité de compenser l'inexactitude de la recherche par des caractéristiques portant uniquement sur les actions passées. En revanche, de nouvelles caractéristiques combinées à de l'information sur les possibilités futures, comme par exemple le coût futur utilisé dans le chapitre 5, pourraient permettre de rendre la recherche approchée compétitive.

19. On notera que le cas particulier du petit vocabulaire permet lui aussi d'exploiter efficacement ces dépendances à longue distance.

La nature des tâches du traitement automatique des langues que nous avons étudiées limite les observations que l'on a pu réaliser afin d'explorer le compromis entre les deux formes d'approximation. Pour la plupart de ces tâches, les informations nécessaires à un étiquetage correct sont disponibles localement. Une étude sur des tâches d'autres domaines où les dépendances à longue distance sont plus importantes, tels que pour le dialogue ou la détection de co-références, pourrait permettre de mieux étudier ce compromis.

Dans ce travail nous avons comparé les deux approches extrêmes du compromis entre les deux formes d'approximation. L'étape suivante naturelle consiste à évaluer des méthodes hybrides qui combineraient les deux approches. En effet, ce travail s'est concentré sur des méthodes utilisant une recherche gloutonne afin de pouvoir explorer en temps raisonnable l'espace de recherche complet, mais il existe d'autres algorithmes de recherche approchée moins naïfs qui permettrait d'explorer un espace de recherche de taille intermédiaire.

Les travaux du chapitre 5 ont montré que lorsque des informations sur le futur sont incluses, les méthodes « apprendre à chercher » peuvent être compétitives sur une tâche complexe telle que la traduction automatique. L'étape suivante de cette étude est d'explorer d'autres configurations incluant de nouvelles caractéristiques plus riches que celles utilisées par le système de référence.

Si l'avenir de la traduction automatique semble actuellement être dominé par les systèmes neuronaux, ceux-ci utilisent des mécanismes non-markoviens similaires à ceux que nous avons étudiés ici et il serait intéressant d'étudier comment ils peuvent en profiter. De plus, les résultats présentés dans ce travail sont toutefois encourageant sur les performances des approches « apprendre à chercher » pour différentes tâches du traitement automatique des langues, notamment par leurs capacités à exploiter facilement les dépendances à longue distance. Certains problèmes restent toutefois ouverts tels que l'utilisation de références non-déterministes ou bien comportant des données latentes pour lesquels la mise en œuvre des méthodes « apprendre à chercher » reste à étudier.

Bibliographie

[Abney, 1991] Abney, S. (1991). Parsing by chunks. In *Principle-Based Parsing*, pages 257–278. Kluwer Academic Publishers. *[Cité page 25]*

[Allauzen et al., 2016] Allauzen, A., Aufrant, L., Burlot, F., Lacroix, O., Knyazeva, E., Lavergne, T., Wisniewski, G., and Yvon, F. (2016). LIMSI@WMT16 : Machine translation of news. In *Proceedings of the First Conference on Machine Translation*, pages 239–245, Berlin, Germany. Association for Computational Linguistics. *[Cité page 105]*

[Allauzen et al., 2013] Allauzen, A., Pécheux, N., Do, Q. K., Dinarelli, M., Lavergne, T., Max, A., Le, H.-S., and Yvon, F. (2013). LIMSI @ WMT13. In *Proceedings of the Eighth Workshop on Statistical Machine Translation*, pages 62–69, Sofia, Bulgaria. *[Cité page 105]*

[Allen et al., 1987] Allen, J., Hunnicutt, M. S., Klatt, D. H., Armstrong, R. C., and Pisoni, D. B. (1987). *From Text to Speech : The MITalk System*. Cambridge University Press, New York, NY, USA. *[Cité page 24]*

[Anguera Miro et al., 2012] Anguera Miro, X., Bozonnet, S., Evans, N., Fredouille, C., Friedland, G., and Vinyals, O. (2012). Speaker diarization : A review of recent research. *Trans. Audio, Speech and Lang. Proc.*, 20(2) :356–370. *[Cité page 28]*

[Aufrant et al., 2017] Aufrant, L., Wisniewski, G., and Yvon, F. (2017). Don't stop me now ! using global dynamic oracles to correct training biases of transition-based dependency parsers. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2 : Short Papers*, pages 318–323. *[Cité pages 73, 74, et 86]*

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the first International Conference on Learning Representations*. *[Cité page 100]*

[Beal, 2003] Beal, M. J. (2003). *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London. *[Cité page 11]*

[Bellman, 1958] Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16(1) :87–90. *[Cité page 102]*

[Bengio et al., 2015] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1171–1179, Cambridge, MA, USA. MIT Press. *[Cité page 64]*

[Berger et al., 1996] Berger, A. L., Della Pietra, V. J., and Della Pietra, S. A. (1996). A maximum entropy approach to natural language processing. *Computational Linguistic*, 22(1) :39–71. *[Cité page 16]*

[Bergstra and Bengio, 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13 :281–305. *[Cité page 18]*

[Bertsekas, 1987] Bertsekas, D. P. (1987). *Dynamic Programming : Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA. *[Cité page 65]*

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific, 1st edition. *[Cité page 65]*

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. *[Cité pages 15 et 77]*

[Bojar et al., 2017] Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huang, S., Huck, M., Koehn, P., Liu, Q., Logacheva, V., Monz, C., Negri, M., Post, M., Rubino, R., Specia, L., and Turchi, M. (2017). Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation, Volume 2 : Shared Task Papers*, pages 169–214, Copenhagen, Denmark. Association for Computational Linguistics. *[Cité page 94]*

[Bottou, 1991] Bottou, L. (1991). Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France. EC2. *[Cité page 18]*

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA. *[Cité page 15]*

[Braud, 2015] Braud, C. (2015). *Automatically Identifying Implicit Discourse Relations using Annotated Corpora and Raw Data*. Theses, Universite Paris Diderot-Paris VII. *[Cité page 27]*

[Burlot et al., 2016] Burlot, F., Knyazeva, E., Lavergne, T., and Yvon, F. (2016). Two-Step MT : Predicting Target Morphology. In *International Workshop on Spoken Language Translation*, IWSLT'16, Seattle, WA, United States. *[Cité pages 76 et 83]*

[Cerny, 1985] Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1) :41–51. *[Cité page 18]*

[Chang et al., 2015] Chang, K.-W., Krishnamurthy, A., Agarwal, A., Daumé III, H., and Langford, J. (2015). Learning to search better than your teacher. In *Proceedings of the International Conference on Machine Learning (ICML)*. *[Cité page 51]*

[Chapelle et al., 2010] Chapelle, O., Schlkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. The MIT Press, 1st edition. *[Cité page 5]*

[Collins, 2002] Collins, M. (2002). Discriminative training methods for hidden markov models : Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '02, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 19]*

[Collins and Roark, 2004] Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité pages 20 et 32]*

[Cormen et al., 1990] Cormen, T. T., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA. *[Cité page 11]*

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, 20(3) :273–297. *[Cité page 16]*

[Crammer et al., 2006] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7 :551–585. *[Cité page 94]*

[Crego et al., 2011] Crego, J. M., Yvon, F., and Mariño, J. B. (2011). N-code : an open-source Bilingual N-gram SMT Toolkit. *Prague Bulletin of Mathematical Linguistics*, 96 :49–58. *[Cité pages 89, 91, et 93]*

[Daumé et al., 2009] Daumé, III, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine Learning*, 75(3) :297–325. *[Cité pages 2, 32, 38, 59, 60, 62, 64, et 102]*

[Daumé III and Marcu, 2005] Daumé III, H. and Marcu, D. (2005). Learning as search optimization : Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*, Bonn, Germany. *[Cité page 32]*

[DeRose, 1988] DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Comput. Linguist.*, 14(1) :31–39. *[Cité page 25]*

[Domingos, 1999] Domingos, P. (1999). Metacost : A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 155–164, New York, NY, USA. ACM. *[Cité page 17]*

[Domke, 2011] Domke, J. (2011). Parameter learning with truncated message-passing. In *Proceedings of CVPR*. *[Cité page 20]*

[Duda et al., 2000] Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2Nd Edition)*. Wiley-Interscience. *[Cité pages 5 et 6]*

[Durrani et al., 2011] Durrani, N., Schmid, H., and Fraser, A. M. (2011). A joint sequence translation model with integrated reordering. In Lin, D., Matsumoto, Y., and Mihalcea, R., editors, *ACL*, pages 1045–1054. The Association for Computer Linguistics. *[Cité page 100]*

[Ehrmann, 2008] Ehrmann, M. (2008). *Les Entitées Nommées, de la linguistique au TAL : Statut théorique et méthodes de désambiguïsation. (Named entities, from Linguistics to NLP : Theoretical status and disambiguation methods)*. PhD thesis, Paris Diderot University, France. *[Cité page 25]*

[Ford, 1956] Ford, L. R. (1956). *Network Flow Theory*. Paper P-923. The Rand Corporation. *[Cité page 102]*

[Fraser et al., 2012] Fraser, A., Weller, M., Cahill, A., and Cap, F. (2012). Modeling inflection and word-formation in smt. In *Proceedings of the*

13th Conference of the European Chapter of the Association for Computational Linguistics, EACL '12, pages 664–674, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 27]*

[Freund and Schapire, 1999] Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3) :277–296. *[Cité page 14]*

[Geist, 2016] Geist, M. (2016). *Contrôle optimal et apprentissage automatique, applications aux interactions homme-machine*. Habilitation à diriger des recherches, Université de Lille 1. *[Cité page 65]*

[Gimpel and Smith, 2010] Gimpel, K. and Smith, N. A. (2010). Softmax-margin crfs : Training log-linear models with cost functions. In *Human Language Technologies : The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 733–736, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 17]*

[Goldberg and Nivre, 2012] Goldberg, Y. and Nivre, J. (2012). A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics*. *[Cité pages 33, 37, 38, 64, 73, 75, et 86]*

[Guta et al., 2015] Guta, A., Alkhouli, T., Peter, J.-T., Wuebker, J., and Ney, H. (2015). A comparison between count and neural network models based on joint translation and reordering sequences. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1401–1411, Lisbon, Portugal. Association for Computational Linguistics. *[Cité page 100]*

[He et al., 2012] He, H., Daumé, III, H., and Eisner, J. (2012). Imitation learning by coaching. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 3149–3157, USA. Curran Associates Inc. *[Cité page 73]*

[Hirschman and Gaizauskas, 2001] Hirschman, L. and Gaizauskas, R. (2001). Natural language question answering : The view from here. *Nat. Lang. Eng.*, 7(4) :275–300. *[Cité page 28]*

[Hobbs, 1986] Hobbs, J. (1986). Readings in natural language processing. chapter Resolving Pronoun References, pages 339–352. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. *[Cité page 28]*

[Hsu et al., 2010] Hsu, C.-W., Chang, C.-C., and Lin, C.-J. (2010). A practical guide to support vector classification. *[Cité page 19]*

[Huang et al., 2012] Huang, L., Fayong, S., and Guo, Y. (2012). Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, NAACL HLT '12, pages 142–151, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité pages 19 et 20]*

[Jurafsky and Martin, 2000] Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing : An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition. *[Cité page 28]*

[Kakade, 2003] Kakade, S. (2003). *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London. *[Cité page 65]*

[Kassel, 1995] Kassel, R. H. (1995). *A Comparison of Approaches to Online Handwritten Character Recognition*. PhD thesis, Cambridge, MA, USA. AAI0576397. *[Cité page 23]*

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *SCIENCE*, 220(4598) :671–680. *[Cité page 18]*

[Knyazeva et al., 2015a] Knyazeva, E., Wisniewski, G., Bredin, H., and Yvon, F. (2015a). Structured prediction for speaker identification in TV series. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 195–199. *[Cité pages 28, 76, et 82]*

[Knyazeva et al., 2015b] Knyazeva, E., Wisniewski, G., and Yvon, F. (2015b). Apprentissage par imitation pour l'étiquetage de séquences : vers une formalisation des méthodes d'étiquetage "easy-first". In *Conférence sur le Traitement Automatique des Langues Naturelles (TALN 2015)*, page (12), Caen, France. ATALA. *[Cité page 76]*

[Koehn, 2010] Koehn, P. (2010). *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition. *[Cité page 26]*

[Kulesza and Pereira, 2008] Kulesza, A. and Pereira, F. (2008). Structured learning with approximate inference. In *Advances in Neural Information Processing Systems 20*. *[Cité page 20]*

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields : Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Confe-*

rence on Machine Learning, (ICML'01), pages 282–289, Williamstown, MA. Morgan Kaufmann, San Francisco, CA. *[Cité page 19]*

[Lavergne et al., 2010] Lavergne, T., Cappé, O., and Yvon, F. (2010). Practical very large scale crfs. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 504–513, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 79]*

[Le and Fokkens, 2017] Le, M. and Fokkens, A. (2017). Tackling error propagation through reinforcement learning : A case of greedy dependency parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics : Volume 1, Long Papers*, pages 677–687. Association for Computational Linguistics. *[Cité page 42]*

[Leusch et al., 2008] Leusch, G., Matusov, E., and Ney, H. (2008). Complexity of finding the bleu-optimal hypothesis in a confusion network. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 839–847, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 95]*

[Li and Khudanpur, 2009] Li, Z. and Khudanpur, S. (2009). Efficient extraction of oracle-best translations from hypergraphs. In *Proceedings of Human Language Technologies : The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume : Short Papers*, NAACL-Short '09, pages 9–12, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 95]*

[Liang et al., 2006] Liang, P., Bouchard-Côté, A., Klein, D., and Taskar, B. (2006). An end-to-end discriminative approach to machine translation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 761–768, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 37]*

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45 :503–528. *[Cité page 17]*

[Mani, 1999] Mani, I. (1999). *Advances in Automatic Text Summarization*. MIT Press, Cambridge, MA, USA. *[Cité page 28]*

[Mohri, 2002] Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3) :321–350. *[Cité page 19]*

[Ng, 2004] Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA. ACM. *[Cité page 15]*

[Nivre, 2005] Nivre, J. (2005). Dependency grammar and dependency parsing. Technical report, Växjö University. *[Cité page 25]*

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. World Scientific. *[Cité page 18]*

[Och, 2003] Och, F. J. (2003). Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 160–167, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 94]*

[Paek and Pieraccini, 2008] Paek, T. and Pieraccini, R. (2008). Automating spoken dialogue management design using machine learning : An industry perspective. *Speech Commun.*, 50(8-9) :716–729. *[Cité page 28]*

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu : A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 90]*

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. *[Cité page 11]*

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn : Machine learning in python. *J. Mach. Learn. Res.*, 12 :2825–2830. *[Cité page 77]*

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition. *[Cité page 32]*

[Rahman and Ng, 2009] Rahman, A. and Ng, V. (2009). Supervised models for coreference resolution. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing : Volume 2 - Volume 2*, EMNLP '09, pages 968–977, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 28]*

[Ranzato et al., 2016] Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2016). Sequence level training with recurrent neural networks. In *International Conference on Learning Representations*. *[Cité pages 42 et 64]*

[Roark et al., 2004] Roark, B., Saraclar, M., Collins, M., and Johnson, M. (2004). Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 19]*

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386. *[Cité page 13]*

[Ross and Bagnell, 2010] Ross, S. and Bagnell, J. A. (2010). Efficient reductions for imitation learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 9, pages 661–668. *[Cité pages 32, 42, 44, 52, 53, et 64]*

[Ross and Bagnell, 2014] Ross, S. and Bagnell, J. A. (2014). Reinforcement and imitation learning via interactive no-regret learning. In *arXiv preprint arXiv :1406.5979v1*. *[Cité pages 2, 32, 47, 48, 49, et 51]*

[Ross et al., 2011] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA. PMLR. *[Cité pages 2, 51, et 59]*

[Schneier, 1995] Schneier, B. (1995). *Applied Cryptography (2Nd Ed.) : Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA. *[Cité page 22]*

[Sejnowski and Rosenberg, 1987] Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1 :145–168. *[Cité pages 24 et 80]*

[Shalev-Shwartz, 2012] Shalev-Shwartz, S. (2012). Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2) :107–194. *[Cité pages 48 et 49]*

[Shalev-Shwartz et al., 2007] Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos : Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA. ACM. *[Cité page 16]*

[Shen et al., 2007] Shen, L., Satta, G., and Joshi, A. (2007). Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767. Association for Computational Linguistics. *[Cité page 67]*

[Smith and Eisner, 2008] Smith, D. A. and Eisner, J. (2008). Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 145–156, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 19]*

[Smith, 2011] Smith, N. A. (2011). *Linguistic Structure Prediction*. Morgan & Claypool Publishers, 1st edition. *[Cité page 29]*

[Sokolov et al., 2012] Sokolov, A., Wisniewski, G., and Yvon, F. (2012). Computing lattice bleu oracle scores for machine translation. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 120–129, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité pages 95 et 96]*

[Stoyanov and Eisner, 2012] Stoyanov, V. and Eisner, J. (2012). Minimum-risk training of approximate crf-based nlp systems. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, NAACL HLT '12, pages 120–130, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 20]*

[Straková et al., 2014] Straková, J., Straka, M., and Hajič, J. (2014). Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics : System Demonstrations*, pages 13–18, Baltimore, Maryland. Association for Computational Linguistics. *[Cité page 83]*

[Sutton et al., 2007] Sutton, C., McCallum, A., and Rohanimanesh, K. (2007). Dynamic conditional random fields : Factorized probabilistic models for labeling and segmenting sequence data. *J. Mach. Learn. Res.*, 8 :693–723. *[Cité page 19]*

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition. *[Cité pages 5, 32, et 65]*

[Toutanova et al., 2003] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 25]*

[Tromble et al., 2008] Tromble, R. W., Kumar, S., Och, F., and Macherey, W. (2008). Lattice minimum bayes-risk decoding for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 620–629, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité pages 95 et 96]*

[Tsochantaridis et al., 2005] Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Machine Learning*, 6 :1453–1484. *[Cité page 19]*

[Tu and Lin, 2010] Tu, H.-H. and Lin, H.-T. (2010). One-sided support vector regression for multiclass cost-sensitive classification. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 1095–1102. *[Cité page 17]*

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA. *[Cité page 6]*

[Vishwanathan et al., 2006] Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., and Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 969–976, New York, NY, USA. ACM. *[Cité page 19]*

[Viterbi, 1967] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2) :260–269. *[Cité page 10]*

[Vlachos, 2013] Vlachos, A. (2013). An investigation of imitation learning algorithms for structured prediction. In Deisenroth, M. P., Szepesvári, C., and Peters, J., editors, *Proceedings of the Tenth European Workshop on Reinforcement Learning*, volume 24 of *Proceedings of Machine Learning Research*, pages 143–154, Edinburgh, Scotland. PMLR. *[Cité page 65]*

[Wisniewski et al., 2010] Wisniewski, G., Allauzen, A., and Yvon, F. (2010). Assessing phrase-based translation models with oracle decoding. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 933–943, Stroudsburg, PA, USA. Association for Computational Linguistics. *[Cité page 95]*

[Wright, 2015] Wright, S. J. (2015). Coordinate descent algorithms. *Math. Program.*, 151(1) :3–34. *[Cité page 18]*

[Zadrozny et al., 2003] Zadrozny, B., Langford, J., and Abe, N. (2003). Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 435–, Washington, DC, USA. IEEE Computer Society. *[Cité page 17]*

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Apprendre par imitation: application à quelques problèmes d'apprentissage structuré en traitement des langues

Mots clés : Apprentissage structuré, apprentissage par imitation, traitement automatique des langues

Résumé : L'apprentissage structuré est devenu omniprésent dans le traitement automatique des langues naturelles. De nombreuses applications qui font maintenant partie de notre vie telles que des assistants personnels, la traduction automatique, ou encore la reconnaissance vocale, reposent sur ces techniques. Les problèmes d'apprentissage structuré qu'il est nécessaire de résoudre sont de plus en plus complexes et demandent de prendre en compte de plus en plus d'informations à des niveaux linguistiques variés (morphologique, syntaxique, etc.) et reposent la question du meilleurs compromis entre la finesse de la modélisation et l'exactitude des algorithmes d'apprentissage et d'inférence. L'apprentissage par imitation propose de réaliser les procédures d'apprentissage et d'inférence de manière approchée afin de pouvoir exploiter pleinement des structures de dépendance plus riches. Cette thèse explore ce cadre d'apprentissage, en particulier l'algorithme SEARN, à la fois sur le plan théorique ainsi que ses possibilités d'application aux tâches de traitement automatique des langues, notamment aux plus complexes telles que la traduction. Concernant les aspects théoriques, nous présentons un cadre unifié pour les différentes familles d'apprentissage par imitation, qui permet de redériver de manière simple les propriétés de convergence de ces algorithmes; concernant les aspects plus appliqués, nous utilisons l'apprentissage par imitation d'une part pour explorer l'étiquetage de séquences en ordre libre; d'autre part pour étudier des stratégies de décodage en deux étapes pour la traduction automatique.

Title : Imitation Learning: Application to Several Structured Learning Tasks in Natural Language Processing

Keywords : Structured Learning, Imitation Learning, Natural Language Processing

Abstract : Structured learning has become ubiquitous in Natural Language Processing; a multitude of applications, such as personal assistants, machine translation and speech recognition, to name just a few, rely on such techniques. The structured learning problems that must now be solved are becoming increasingly more complex and require an increasing amount of information at different linguistic levels (morphological, syntactic, etc.). It is therefore crucial to find the best trade-off between the degree of modelling detail and the exactitude of the inference algorithm. Imitation learning aims to perform approximate learning and inference in order to better exploit richer dependency structures. In this thesis, we explore the use of this specific learning setting, in particular using the SEARN algorithm, both from a theoretical perspective and in terms of the practical applications to Natural Language Processing tasks, especially to complex tasks such as machine translation. Concerning the theoretical aspects, we introduce a unified framework for different imitation learning algorithm families, allowing us to review and simplify the convergence properties of the algorithms. With regards to the more practical application of our work, we use imitation learning first to experiment with free order sequence labelling and secondly to explore two-step decoding strategies for machine translation.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France