Comparaison d'architectures neuronales pour l'analyse syntaxique en constituants

Maximin Coavoux, Benoît Crabbé

▶ **To cite this version:**

Maximin Coavoux, Benoît Crabbé. Comparaison d'architectures neuronales pour l'analyse syntaxique en constituants. TALN 2015 - 22ème Traitement Automatique des Langues Naturelles, Caen, 2015, 2015, Caen, France. hal-01174613

HAL Id: hal-01174613
https://inria.hal.science/hal-01174613

Submitted on 29 Jul 2022

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

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

Comparaison d'architectures neuronales pour l'analyse syntaxique en constituants

Maximin Coavoux[1] Benoît Crabbé[1,2]
(1) ALPAGE, INRIA, Université Paris Diderot, Place Paul Ricœur, 75013 Paris
(2) Institut Universitaire de France
maximin.coavoux@inria.fr, bcrabbe@linguist.univ-paris-diderot.fr

Résumé. L'article traite de l'analyse syntaxique lexicalisée pour les grammaires de constituants. On se place dans le cadre de l'analyse par transitions. Les modèles statistiques généralement utilisés pour cette tâche s'appuient sur une représentation non structurée du lexique. Les mots du vocabulaire sont représentés par des symboles discrets sans liens entre eux. À la place, nous proposons d'utiliser des représentations denses du type plongements (*embeddings*) qui permettent de modéliser la similarité entre symboles, c'est-à-dire entre mots, entre parties du discours et entre catégories syntagmatiques. Nous proposons d'adapter le modèle statistique sous-jacent à ces nouvelles représentations. L'article propose une étude de 3 architectures neuronales de complexité croissante et montre que l'utilisation d'une couche cachée non-linéaire permet de tirer parti des informations données par les plongements.

Abstract.
A Comparison of Neural Network Architectures for Constituent Parsing
The article deals with lexicalized constituent parsing in a transition-based framework. Typical statistical approaches for this task are based on an unstructured representation of the lexicon. Words are represented by discrete unrelated symbols. Instead, our proposal relies on dense vector representations (embeddings) that are able to encode similarity between symbols : words, part-of-speech tags and phrase structure symbols. The article studies and compares 3 increasingly complex neural network architectures, which are fed symbol embeddings. The experiments suggest that the information given by embeddings is best captured by a deep architecture with a non-linear layer.

Mots-clés : Analyse syntaxique en constituants lexicalisée, plongements, réseaux de neurones.

Keywords: Lexicalized constituent parsing, embeddings, neural networks.

1 Introduction

On s'intéresse dans cet article à l'analyse syntaxique lexicalisée pour des grammaires de constituants. Dans le cas robuste, pour gérer l'ambiguité inhérente à la langue naturelle, on augmente l'algorithme d'analyse d'une méthode de pondération. Ici on propose d'étudier plus spécifiquement le procédé de pondération basé sur un modèle statistique discriminant de type neuronal dans le cas de l'analyse par décalage réduction.

Pour ce problème, les propositions existantes en termes de modèle statistique s'appuient principalement sur la régression logistique multinomiale (dorénavant RLM) (Sagae & Lavie, 2006; Zhao *et al.*, 2013) ou sur le perceptron global (Zhang & Clark, 2009; Zhu *et al.*, 2013; Crabbé, 2014b). Ces architectures s'appuient sur un codage des données par des vecteurs creux de très haute dimensionnalité par l'intermédiaire de fonctions de trait. En première analyse, pour un vocabulaire Σ, chaque mot est codé sur une dimension spécifique d'un vecteur de dimension $|\Sigma|$. Autrement dit, aucune structure n'est donnée à la représentation du lexique. Dans le cas de l'analyse syntaxique lexicalisée, et dans le cas où les paramètres du modèle statistique sont estimés sur un corpus arboré, qui reste de petite taille, ce type de codage a comme inconvénient principal que les paramètres liés aux mots sont très mal estimés. Chaque mot ou couple de mots, dans un modèle d'analyse syntaxique lexicalisé, est en général vu très rarement dans les données.

Dans ce qui suit, on propose de changer la représentation des mots par des représentations vectorielles denses du type plongements lexicaux (*word embeddings*). Dans ce cas, un mot est représenté par un vecteur réel de faible dimension, ce qui permet d'espérer que les propriétés apprises pour un mot par un modèle d'analyse syntaxique puisse se transférer à des mots dont les vecteurs sont similaires. L'introduction de représentations vectorielles pour les symboles discrets nous mène

également à adapter les modèles statistiques habituels à ces nouvelles représentations, ce qui nous pousse à utiliser des modèles neuronaux. De plus, pour homogénéiser et généraliser les représentations, nous codons l'ensemble des symboles discrets de la grammaire (mots, catégories syntagmatiques, parties de discours) par des vecteurs.

L'article est organisé comme suit. Après avoir revu les résultats récents en termes d'analyse syntaxique et de représentations vectorielles de mots (section 2), on rappelle le fonctionnement de l'algorithme d'analyse syntaxique qui sert de support à notre étude (section 3). On propose en section 4 une légère reformulation des méthodes de pondération bien connues pour le cas de représentations creuses, comme la RLM, au cas des représentations vectorielles denses. Nous suggérons ensuite différentes extensions qui permettent d'acquérir à la fois des représentations vectorielles de symboles discrets et une méthode de pondération des analyses. La section 5 nous permet de comparer expérimentalement les différents modèles et de discuter leur formulation dans un cas de prédiction structurée comme l'analyse syntaxique lexicalisée en constituants.

2 Contexte scientifique et état de l'art

L'usage d'analyseurs syntaxiques avec modèle de pondération discriminant n'est en lui-même pas nouveau. L'algorithme que l'on utilise ici est inspiré des travaux de Sagae & Lavie (2006). Nous présentons un algorithme glouton de recherche de la meilleure solution, ce qui le distingue des méthodes de recherche en faisceau telles que présentées par ailleurs par (Zhu *et al.*, 2013; Crabbé, 2014a).

Ce que nous proposons de nouveau, c'est de construire un analyseur dont le modèle de pondération est fondamentalement basé sur des vecteurs denses de réels. De telles représentations, appelées plongements lexicaux ou *word embeddings* sont connues depuis longtemps en traitement automatique des langues et ont été particulièrement popularisées par Mikolov *et al.* (2013). L'usage de représentations vectorielles pour modéliser le langage n'est pas nouveau non plus. On peut d'ailleurs voir d'une certaine manière notre proposition comme une transposition du travail de Bengio *et al.* (2003) pour les modèles de langue au cas de l'analyse syntaxique en constituants. L'usage de représentations vectorielles pour la prédiction structurée en traitement automatique des langues dérive généralement du travail de Collobert & Weston (2008).

L'idée sous-jacente à l'utilisation de représentations vectorielles en analyse syntaxique lexicalisée vient du fait que les corpus arborés sont petits et que les statistiques liées aux mots – considérées comme cruciales – sont en général mal estimées. Un nombre important d'auteurs a tenté de combattre ce problème de dispersion en ajoutant des traits additionnels à des algorithmes d'analyse traditionnels. Ces traits prennent la forme de clusters ou de plongements lexicaux (Bansal *et al.*, 2014; Turian *et al.*, 2010; Candito & Crabbé, 2009; Koo *et al.*, 2008). Mais si des représentations continues ou issues de clusters permettent d'extraire des informations syntaxiques pertinentes, ces informations sont largement redondantes avec celles apportées par des traits classiques, comme les parties du discours (Andreas & Klein, 2014), de sorte que les traits basés sur les représentations continues n'apportent qu'une très faible amélioration pour l'analyse en constituants. C'est une des raisons pour lesquelles on propose ici un modèle d'analyse entièrement fondé sur des représentations vectorielles de symboles discrets.

Deux propositions principales pour l'analyse syntaxique en constituants basée sur des vecteurs ont émergé récemment. D'une part, Henderson (2004); Titov & Henderson (2007) proposent une architecture fondée sur des réseaux de neurones récurrents. Celle-ci prédate toutefois l'arrivée du *deep learning* et cet algorithme utilise des représentations lexicales qui ne correspondent pas à des plongements lexicaux. Plus récemment, Socher *et al.* (2013) proposent l'utilisation de réseaux de neurones comme algorithme de réordonnancement d'analyses pour une PCFG.

À notre connaissance, nous sommes donc les premiers à présenter un modèle de type *deep learning* pour l'analyse syntaxique en constituants. On remarquera toutefois que notre proposition est analogue à celle de Chen & Manning (2014) pour l'analyse en dépendances.

3 Algorithme d'analyse

Cette section introduit rapidement aux propriétés de l'algorithme d'analyse syntaxique par décalage réduction sous-jacent à cette étude. On commence par caractériser la grammaire utilisée par l'analyseur avant d'introduire l'algorithme proprement dit.

Pour l'analyse lexicalisée en constituants, on utilise une grammaire analogue à celle de Crabbé (2014b). On contraint

la grammaire à être en forme binaire lexicalisée (2-LCFG) qui est une forme binarisée dont les règles prennent nécessairement une des formes données en Table 1. Les symboles h, x dénotent des symboles terminaux. Les symboles de la

$$\begin{aligned} A[h] &\rightarrow B[h]\,C[x] \\ A[h] &\rightarrow B[x]\,C[h] \\ A[h] &\rightarrow h \end{aligned}$$

TABLE 1 – Formes des règles 2-LCFG

forme $A[h]$ dénotent des symboles non terminaux lexicalisés. Un tel symbole est composé d'un non terminal délexicalisé noté A, B, C et d'un terminal h ou x. Une règle 2-LCFG sera par exemple de la forme $NP[chat] \rightarrow D[le]\,N[chat]$. Le symbole de la forme $X[h]$ situé en partie droite de la règle est appelé tête de la règle. Une telle grammaire est extraite d'un corpus arboré binarisé. Dans cet article nous avons utilisé la construction décrite par Crabbé (2014b) de manière à garantir que, pour toute analyse d'une phrase de longueur n par un analyseur à décalage réduction, la dérivation a exactement une longueur de $3n - 1$ pas.

L'algorithme d'analyse par décalage-réduction (*shift-reduce*) se fonde sur deux structures de données : une pile $\mathcal{S}$ et une file $\mathcal{W}$, et deux familles d'actions : le décalage et la réduction. On appelle *configuration* ou *état*, le couple $\langle \mathcal{S}, \mathcal{W} \rangle$ à une étape t de l'analyse. À l'état initial, la pile est vide et la file contient la séquence de mots à analyser. L'action de décalage consiste à déplacer le sommet de $\mathcal{W}$ et à l'empiler sur $\mathcal{S}$. Une action de réduction est paramétrée par le symbole X. Celle-ci dépile 1 (réduction unaire) ou 2 éléments au sommet de $\mathcal{S}$, et empile le symbole non-terminal X à la place. De plus, comme les éléments de la pile sont lexicalisés, on distingue des réductions gauches et des réductions droites. Une réduction gauche empile un élément $X[h]$ si le sommet de la pile est $B[h]\,C[x]$ alors qu'une réduction droite empile un élément $X[h]$ si le sommet de la pile est $B[x]\,C[h]$. En exécutant une action a à partir d'une configuration C_i, on dérive une nouvelle configuration C_{i+1}, ce que l'on note : $C_i \overset{a}{\Rightarrow} C_{i+1}$. On appelle dérivation une séquence de configurations $C_0 \overset{a_0}{\Rightarrow} C_1 \overset{a_1}{\Rightarrow} \ldots \overset{a_{k-1}}{\Rightarrow} C_k$, telle que C_0 est l'état initial. Au terme de l'analyse, si la pile contient seulement l'axiome et si la file est vide, le mot est reconnu. On présente en figure 1 l'algorithme sous forme de règles d'inférence.

ITEM	$\langle \mathcal{S}, \mathcal{W} \rangle : w$	
ÉTAT INITIAL	$\langle \emptyset, [\mathtt{w}_1\ \mathtt{w}_2\ \ldots \mathtt{w}_n] \rangle : 0$	
ÉTAT FINAL	$\langle [\mathtt{S}], \emptyset \rangle : w$	
DÉCALAGE	$\dfrac{\langle \mathcal{S}, [\mathtt{w}_i\ \mathtt{w}_{i+1}\ \ldots \mathtt{w}_n] \rangle : w}{\langle \mathcal{S} + \mathtt{w}_i, [\mathtt{w}_{i+1}\ \ldots \mathtt{w}_n] \rangle : w + f(\text{DÉCALAGE}, \langle \mathcal{S}, \mathcal{W} \rangle)}$	
RÉDUCTION-GAUCHE-X	$\dfrac{\langle [\ \alpha\ \mathtt{B[h]}\ \mathtt{C[x]}], \mathcal{W} \rangle : w}{\langle [\ \alpha\ \mathtt{X[h]}], \mathcal{W} \rangle : w + f(\text{RG-X}, \langle \mathcal{S}, \mathcal{W} \rangle)}$	$\text{X} \rightarrow \text{B C} \in R$
RÉDUCTION-DROITE-X	$\dfrac{\langle [\ \alpha\ \mathtt{B[x]}\ \mathtt{C[h]}], \mathcal{W} \rangle : w}{\langle [\ \alpha\ \mathtt{X[h]}], \mathcal{W} \rangle : w + f(\text{RD-X}, \langle \mathcal{S}, \mathcal{W} \rangle)}$	$\text{X} \rightarrow \text{B C} \in R$
RÉDUCTION-UNAIRE-X	$\dfrac{\langle [\ \alpha\ \mathtt{A[h]}], \mathcal{W} \rangle : w}{\langle [\ \alpha\ \mathtt{X[h]}], \mathcal{W} \rangle : w + f(\text{RU-X}, \langle \mathcal{S}, \mathcal{W} \rangle)}$	$\text{X} \rightarrow \text{A} \in R$

FIGURE 1 – Algorithme pondéré pour une grammaire bilexicale. S est l'axiome de la grammaire, R est l'ensemble des règles de grammaire. Chaque état de l'analyse possède un poids, calculé comme la somme du poids d'une action et de celui de l'état dont il est issu. Chaque action a est pondérée par une fonction $f(a, \langle \mathcal{S}, \mathcal{W} \rangle)$, qui dépend de la configuration courante.

L'algorithme n'est pas déterministe. Comme les actions de réduction sont paramétrées par un symbole X qui est un symbole non terminal, en posant Σ l'ensemble de non terminaux, il y a $3|\Sigma| + 1$ actions au total. Il y a en effet trois types de réductions et l'action de décalage. Pour choisir une dérivation, on s'appuie sur une fonction de pondération qui donne un poids à chaque dérivation possible $C_{0 \Rightarrow k} = C_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{k-1}}{\Rightarrow} C_k$. Par convention, la dérivation à choisir est celle de poids maximal. La méthode de pondération s'appuie sur une fonction de scorage de la forme :

$$f(C_{0 \Rightarrow k}) = \sum_{i=0}^{k-1} f(a_i, C_i) \tag{1}$$

où f est une fonction qui dépend de l'action a_i à effectuer et d'informations C_i accessibles localement à la configuration (Figure 2). Autrement dit, on suppose que le score d'une dérivation se décompose comme la somme des scores de chacun de ses pas. Dans ce contexte, le problème de l'analyse syntaxique consiste à trouver la meilleure dérivation possible de la phrase c'est-à-dire à donner la solution de :

$$\mathcal{C}^* = \underset{C_{0\Rightarrow 3n-1} \in \textsc{Gen}(w_1^n)}{\operatorname{argmax}} f(C_{0\Rightarrow 3n-1})$$

où $\textsc{Gen}(w_1^n)$ est l'ensemble des analyses possibles pour une phrase de longueur n. L'algorithme que nous présentons ici ne construit pas l'ensemble des dérivations possibles pour une phrase donnée mais procède par recherche gloutonne. À chaque point de choix, l'action qui a le meilleur score est sélectionnée en fonction de l'information localement disponible sans possibilité de remise en question à une étape ultérieure. Cela le distingue des méthodes de recherche par faisceau comme celle de Crabbé (2014a).

4 Représentations denses pour l'analyse syntaxique déterministe

Cette section introduit notre proposition principale : celle-ci porte sur la conception de la fonction locale de scorage $f(a, C)$. On commence par reformuler le système de représentation creux traditionnel, basé sur des fonctions de traits, par un système dense dans lequel chacun des symboles (mots, parties du discours et catégories syntagmatiques) est codé par un vecteur de réels de basse dimensionnalité. On montre que ce changement de représentation ne modifie ni le comportement de la fonction de scorage ni le fonctionnement de l'algorithme d'analyse. La seconde étape du développement introduit une famille d'architectures neuronales qui d'une part permet d'obtenir par apprentissage un dictionnaire de représentations denses pour les symboles discrets et d'autre part qui vise à introduire des interactions entre les variables du modèle.

4.1 Des représentations creuses aux représentations denses

Représentations creuses Le passage de représentations creuses à des représentations denses impacte la fonction de scorage $f(a, C)$ locale à chaque pas d'analyse. Les modèles statistiques discriminants bien connus, comme le perceptron définissent typiquement $f(a, C)$ comme un produit scalaire $f(a, C) = \mathbf{w} \cdot \mathbf{\Phi}(a, C)$ où $\mathbf{w}$ dénote un vecteur de poids et $\mathbf{\Phi}(a, C)$ dénote un vecteur booléen valué par d fonctions indicatrices $\phi_j(a, C)$ appelées fonction de trait ($1 \leq j \leq d$). Dans le cas de la RLM qui nous intéresse plus particulièrement pour la suite, on normalise le produit scalaire à l'aide de la fonction softmax, et on interprète $f(a, C)$ comme la probabilité de l'action a sachant la configuration C, ce qui donne [1] :

$$f(a, C) = \frac{\exp(\mathbf{w} \cdot \mathbf{\Phi}(a, C))}{\sum_{a' \in A} \exp(\mathbf{w} \cdot \mathbf{\Phi}(a', C))} = P(a \mid C; \mathbf{w}) \tag{2}$$

Les fonctions de trait ont pour paramètres l'action a à évaluer et la configuration C de l'analyseur. Pour une configuration C, les fonctions de trait ont accès au sommet de la pile d'analyse et à la file d'attente. On donne en Figure 2 un exemple de représentation de l'information locale accessible pour valuer les fonctions de traits. Dans cet exemple, les 3 premiers éléments de la pile, les descendants directs des 2 premiers éléments de la pile et les 4 premiers éléments de la file sont accessibles. s_i et q_i représentent respectivement les i^e éléments de la pile et de la file. c_t, c_l et c_r codent respectivement les symboles non-terminaux d'un élément de la pile, de son fils gauche et de son fils droit. w_t, w_l et w_r codent les têtes de ces non-terminaux. On peut adresser les éléments de la file uniquement à l'aide des indices correspondant à leur position dans la phrase : j pour le sommet de la file, $j+1$, $j+2\ldots$ pour les suivants. Les éléments de la file comme les éléments lexicaux de la pile – notés entre crochets – sont eux-même structurés. Il s'agit de tokens lexicaux. On notera par exemple $q_i.w$ pour dénoter la forme lexicale du token et $q_i.tag$ pour dénoter le tag de ce token.

Dans un tel paradigme les fonctions de trait codent effectivement un couple (a, C) sur un vecteur creux de très haute

1. On remarque que la formule de scorage des analyses donnée en (1) reste inchangée si on se place, comme c'est le cas en pratique, dans un espace log probabilisé et en s'appuyant sur l'égalité : $\log\left(\prod_{i=1}^{k-1} P(a_i \mid C_i; \mathbf{W})\right) = \sum_{i=1}^{k-1} \log P(a_i \mid C_i; \mathbf{W})$, c'est-à-dire en posant que $f(a, C) = \log P(a \mid C; \mathbf{W})$

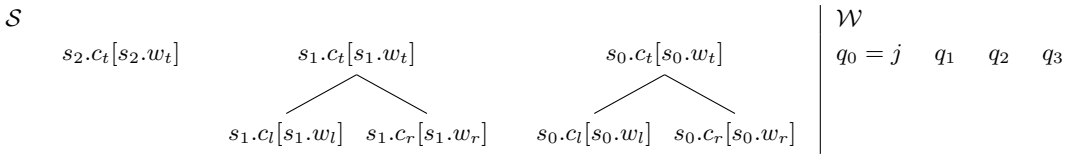


FIGURE 2 – Information accessible aux fonctions de trait depuis une configuration C

dimensionnalité (d est très grand). Celles-ci ont par exemple une forme du type :

$$\phi_j(a, C) = \begin{cases} 1 & \textbf{si l'action est } a_i \textbf{ et le mot tête en sommet de pile } (s_0.w_t) \text{ est le mot } \textit{chat} \\ 0 & \textbf{sinon} \end{cases}$$

$$\phi_k(a, C) = \begin{cases} 1 & \textbf{si l'action est } a_i \textbf{ et le mot tête en sommet de pile } (s_0.w_t) \text{ est le mot } \textit{chat} \\ & \textbf{et le symbole grammatical } (s_1.c_t) \text{ en sous-sommet de pile est } \textit{Déterminant} \\ 0 & \textbf{sinon} \end{cases}$$

Si les représentations creuses sont couramment utilisées avec succès en TAL, observons toutefois que les fonctions de trait codent sur des dimensions distinctes des informations liées à la connaissance du lexique. Par exemple on codera indépendamment sur une dimension j un trait lié au mot *chat* et sur une dimension i un trait lié au mot *chien*. Vu la petite taille des corpus arborés utilisés pour l'entraînement, on souhaiterait idéalement qu'un modèle soit capable de tirer parti du fait que *chat* et *chien* sont des mots sémantiquement proches ou encore que des catégories syntagmatiques comme 'phrase principale' et 'phrase subordonnée' sont plus proches que 'nom' et 'verbe'. Autrement dit, on souhaiterait pouvoir améliorer l'estimation des poids en intégrant dans le modèle statistique une notion de similarité entre symboles, qui sera inférée à partir des seules données du corpus lors de l'apprentissage.

Représentations denses Partant de ces deux observations, on propose ici un système de codage qui s'appuie sur des représentations denses de symboles discrets. Un tel codage peut se concevoir, en première approximation, comme un dictionnaire $\delta : \Sigma \rightarrow \mathbb{R}^d$ qui envoie un ensemble Σ de symboles discrets sur des vecteurs de réels de faible dimension (d est petit). Dans un tel contexte on peut redéfinir une fonction de codage $\mathbf{\Phi}(C)$ comme une concaténation de vecteurs $\delta(\cdot)$ à valeurs réelles qui code l'ensemble des symboles discrets $x_1 \dots x_c$ accessibles à l'analyseur depuis une configuration donnée :

$$\mathbf{\Phi}(C) = [\delta(x_1)\delta(x_2)\dots\delta(x_c)]^T$$

Chaque x_i se voit attribuer une valeur accessible depuis la configuration, comme par exemple $q_0.w$ ou $s_0.w_t$ ou encore $s_1.c_t \dots$ (voir Figure 2). En posant une liste d'actions $A = a^{(1)} \dots a^{(m)}$ de dimension $out = |A|$ et un vecteur $\mathbf{\Phi}(C)$ de dimension $in = |\mathbf{\Phi}(C)|$, on représente les poids par une matrice $\mathbf{W}^{out \times in}$ dont chaque rangée r représente le vecteur de poids $\mathbf{w}_r$ correspondant à l'action $a^{(r)}$. Dans cette nouvelle représentation on redéfinit par conséquent $f(a, C)$ par $f(a^{(r)}, C) = \mathbf{w}_r \cdot \mathbf{\Phi}(C)$. On observe plus généralement qu'en normalisant $f(a^{(r)}, C)$ à l'aide de la fonction softmax, on peut réexprimer $f(a^{(r)}, C)$ par un réseau de neurones élémentaire de type RLM :

$$f(a^{(r)}, C) = \frac{\exp(\mathbf{w}_r \cdot \mathbf{\Phi}(C))}{\sum_{j=1}^{|A|} \exp(\mathbf{w}_j \cdot \mathbf{\Phi}(C))} \quad (3)$$

Celui-ci ne diffère pas fondamentalement de l'équation (2). La seule différence est que (3) s'interprète plus directement comme un réseau de neurone à une couche. On peut d'ailleurs exprimer ce modèle dans une notation graphique inspirée de Bengio *et al.* (2003) pour décrire les architectures de réseaux de neurones, comme illustré en Figure 3.

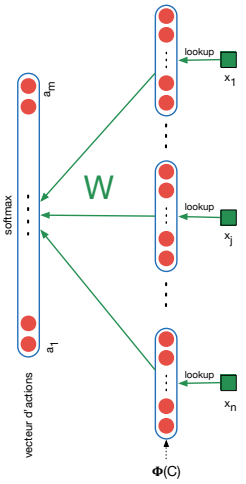


FIGURE 3 – La régression logistique multinomiale vue comme un réseau de neurones à une couche

On tire de (3) une interprétation probabilisée : $P(a^{(r)} \mid C; \mathbf{W}) = f(a^{(r)}, C)$. On a montré dans cette section que le passage au modèle dense ne change pas fondamentalement le fonctionnement de l'algorithme d'analyse [2] : il s'agit essentiellement d'une représentation alternative de la fonction de scorage.

2. En particulier, on remarquera que la représentation adoptée ici, par codage dynamique des symboles discrets se généralise en théorie aux cas d'algorithmes d'analyse qui structurent l'espace de recherche par programmation dynamique.

Par contre, il nous reste à traiter les questions suivantes. D'ou vient la fonction δ (ou dictionnaire) et quelles propriétés cette fonction est-elle supposée satisfaire ? Comment, pour le cas dense, exprimer une contrepartie des interactions entre variables, exprimées naturellement dans les modèles creux, et illustrées ci-dessus par une fonction de trait telle que $\phi_k(a, C)$? Comment estimer la matrice de poids ? On propose un ensemble de réponses à ces questions dans la section suivante, en réexprimant plus généralement $f(a^{(r)}, C)$ par une famille de réseaux de neurones artificiels à propagation avant (*feedforward*).

4.2 Architectures de réseaux

De manière à répondre aux trois questions soulevées dans la section précédente, nous instancions trois architectures de réseaux de neurones, de complexité croissante, potentiellement utilisables pour l'analyse syntaxique. On peut voir ces différentes architectures comme des extensions progressives de modèles de la famille Maximum Entropy Markov Model (MEMM).

Les deux premiers modèles que nous proposons cherchent à illustrer l'introduction du dictionnaire dense dans le modèle statistique. On suppose que dans une configuration donnée l'analyseur a accès aux informations représentées en Figure 2. Il s'agit de valeurs discrètes, que nous notons $x_1 \dots x_c$ et qui regroupent des formes de mots, des catégories syntagmatiques et des catégories morphosyntaxiques. Les formes de mots accessibles sont les formes lexicales représentant la tête lexicale d'un syntagme dans la pile, $s_0.w_t, s_1.w_t, s_2.w_t$. Il s'agit également des formes de mots accessibles dans la file et que nous notons $q_0.w, q_1.w, q_2.w, q_3.w$. Les catégories syntagmatiques sont accessibles dans la pile : $s_0.c_t, s_1.c_t, s_2.c_t$ et finalement les catégories morphosyntaxiques sont accessibles à la fois dans la pile $s_0.tag_t$, $s_0.tag_l$, $s_0.tag_r$, $s_1.tag_t$, $s_1.tag_l$, $s_1.tag_r$, $s_2.tag_t$ et dans la file : $q_0.tag, q_1.tag, q_2.tag, q_3.tag$.

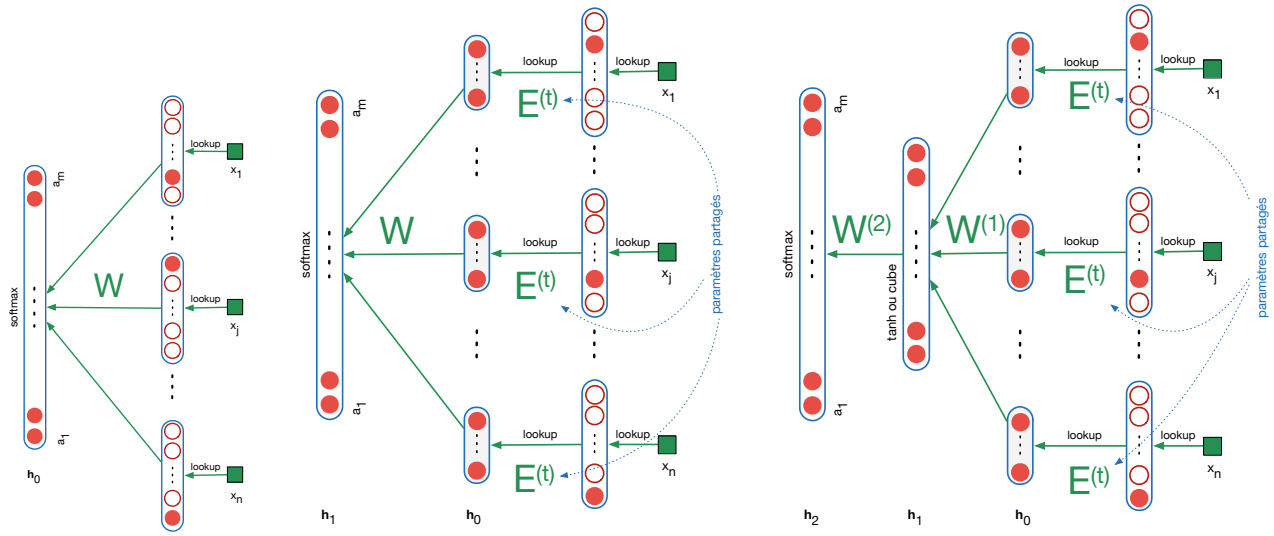
Pour cette raison, tout symbole d'une séquence $x_1, x_2 \dots x_c$ ainsi observée à partir d'une configuration, est systématiquement typé par un type t indiquant si il s'agit d'un mot (W), d'une catégorie syntagmatique (S) ou d'une catégorie morphosyntaxique (T). Pour tirer parti du typage, on distingue dorénavant trois fonctions de dictionnaire $\delta_t : \Sigma_t \mapsto \mathbb{R}^{d_t}$ de telle sorte que chaque type de symbole est potentiellement codé sur des vecteurs de dimension propre à ce type : intuitivement, on peut par exemple envisager de coder les tags sur des vecteurs de dimensionnalité plus petite que les mots. Le dictionnaire de chaque type, est représenté par une table de correspondance (*lookup table*) qui prend la forme d'une matrice $\mathbf{E}^{(t)} \in \mathbb{R}^{|t| \times d_t}$. Chaque symbole x de type t, est assigné à un unique vecteur creux $\mathbf{e}_x \in \mathbb{R}^{1 \times |t|}$ dont une valeur unique est mise à 1. On obtient sa représentation vectorielle dense $\mathbf{e}_x^{(t)} \in \mathbb{R}^{1 \times d_t}$ en le multipliant avec la matrice du type correspondant : $\mathbf{e}_x^{(t)} = \mathbf{e}_x \cdot \mathbf{E}^{(t)}$. Il doit être clair, dans la suite de cet article, que nous ne supposons pas que les représentations denses nous sont données. Au contraire, nous proposons ci-dessous des modèles statistiques qui vont inférer les plongements $\mathbf{e}_x^{(t)}$ par effet de bord de la procédure d'apprentissage.

La représentation du dictionnaire dense étant posée, on propose par la suite de construire d'abord un modèle trivial de type RLM qui ne fait pas appel à la représentation lexicale dense, on propose ensuite un second modèle qui ajoute la représentation lexicale dense au premier modèle et finalement un troisième modèle « profond » qui incorpore une couche cachée supplémentaire.

Régression logistique multinomiale comme modèle de base Ce premier modèle, utilisé comme *baseline* dans nos expériences, représente chaque symbole discret x par un vecteur creux $\mathbf{e}_x$. Celui-ci représente un modèle naïf de type RLM. On peut l'exprimer comme un réseau de neurones *feedforward* (figure 4, partie gauche). Il se compose essentiellement d'une couche de sortie softmax et s'appuie sur des représentations vectorielles creuses des symboles. Ce modèle, de paramètres $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$, se formule comme suit :

$$\mathbf{h}^{(0)} = \text{softmax}\left(\mathbf{W}\left[\mathbf{e}_{x_1}\mathbf{e}_{x_2}\cdots\mathbf{e}_{x_c}\right]^T + \mathbf{b}\right)$$
$$P(a|C, \boldsymbol{\theta}) = h_a^{(0)}$$

Réseau de neurones superficiel Le second modèle testé cherche à introduire explicitement une représentation dense de type *embeddings* pour les symboles (mots, catégories, tags). On procède par extension du modèle de RLM en ajoutant une couche intermédiaire entre la couche d'entrée et la couche de sortie (figure 4, partie centrale). Cette couche intermédiaire est obtenue par une opération de *look-up* dans les matrices $\mathbf{E}^{(\text{W})}, \mathbf{E}^{(\text{S})}, \mathbf{E}^{(\text{T})}$, ou de manière équivalente par multiplication



Rég. log. multinomiale Réseau superficiel Réseau profond

FIGURE 4 – Architectures de réseaux pour la prédiction locale. Les biais ne sont pas représentés sur les schémas.

matricielle entre la matrice $\mathbf{E}^{(t)}$ de type approprié et le vecteur $\mathbf{e}_x$. La probabilité conditionnelle d'une action prend alors la forme suivante :

$$\mathbf{h}^{(0)} = \left[\mathbf{e}_{x_1}^{(t_1)}\mathbf{e}_{x_2}^{(t_2)}\cdots\mathbf{e}_{x_n}^{(t_n)}\right]^T \qquad \text{(opération de look-up)}$$
$$\mathbf{h}^{(1)} = \text{softmax}\left(\mathbf{W}\,\mathbf{h}^{(0)} + \mathbf{b}\right)$$
$$P(a|C,\boldsymbol{\theta}) = h_a^{(1)}$$

Les paramètres à estimer sont $\boldsymbol{\theta} = (\mathbf{E}^{(\mathrm{W})}, \mathbf{E}^{(\mathrm{S})}, \mathbf{E}^{(\mathrm{T})}, \mathbf{W}, \mathbf{b})$. Il s'agit d'un réseau *feedforward*. Ce second modèle fait apparaître les plongements $\mathbf{e}_{x_i}^{(t_i)}$ qu'ils soient lexicaux, syntagmatiques ou morphosyntaxiques. Autrement dit, cette architecture cherche à tirer parti de similarités distributionnelles entre les symboles et à améliorer l'estimation des paramètres.

Notons toutefois qu'il ne prend pas en compte les interactions entre les variables. Par ailleurs, ce modèle ne permet pas d'apprendre de fonction de décision plus complexe que le modèle de RLM. On peut montrer que sa fonction de décision est équivalente à celle d'un tel modèle en réécrivant le produit $\mathbf{W}\,\mathbf{h}^{(0)}$:

$$\begin{aligned}
\mathbf{W}\mathbf{h}^{(0)} &= \mathbf{W}\left[\mathbf{e}_{x_1}^{(t_1)}\mathbf{e}_{x_2}^{(t_2)}\cdots\mathbf{e}_{x_n}^{(t_n)}\right]^T \\
&= \mathbf{W}\left(\begin{bmatrix} \mathbf{E}^{(t_1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{E}^{(t_1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{E}^{(t_n)} \end{bmatrix}\left[\mathbf{e}_{x_1}\mathbf{e}_{x_2}\cdots\mathbf{e}_{x_n}\right]^T\right) \\
&= \left(\mathbf{W}\begin{bmatrix} \mathbf{E}^{(t_1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{E}^{(t_1)} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{E}^{(t_n)} \end{bmatrix}\right)\left[\mathbf{e}_{x_1}\mathbf{e}_{x_2}\cdots\mathbf{e}_{x_n}\right]^T \\
&= \mathbf{W}'\left[\mathbf{e}_{x_1}\mathbf{e}_{x_2}\cdots\mathbf{e}_{x_n}\right]^T
\end{aligned}$$

où $\mathbf{W}'$ est la matrice de poids d'un modèle de RLM. Au lieu d'estimer directement une matrice de poids, comme dans le modèle de RLM, cette architecture tente d'apprendre une factorisation de cette matrice.

Réseau de neurones profond Dans un troisième temps, nous cherchons à modéliser des interactions entre les représentations denses. Dans le cas de représentations creuses, ces interactions sont capturées à l'aide de conjonctions de fonctions

indicatrices comme celle présentée en section 4.1. Néanmoins, le nombre de fonction de trait à utiliser pour capturer toutes les interactions entre des groupes de 2 voire 3 variables est très grand et les considérer toutes est en général redondant. En ajoutant une couche cachée non-linéaire, on peut extraire automatiquement les informations de la couche précédente pertinentes pour l'analyse (figure 4, partie droite). Nous obtenons alors un réseau *feedforward* profond avec deux couches cachées dont une non-linéaire. La première couche cachée ($\mathbf{h}^{(0)}$) récupère les représentations denses correspondant à chaque symbole de l'entrée x (plongements). La deuxième couche cachée ($\mathbf{h}^{(1)}$) extrait des traits latents à partir de la couche précédente, en lui appliquant une transformation affine suivie d'une fonction d'activation non-linéaire g. La fonction d'activation la plus couramment utilisée est la tangente hyperbolique. Enfin, la couche de sortie ($\mathbf{h}^{(2)}$) est obtenue en appliquant la fonction softmax à une transformation affine de la couche cachée. On obtient ainsi une distribution de probabilités sur l'ensemble des actions étant donné la configuration.

$P(a|C;\boldsymbol{\theta})$ se calcule de la manière suivante :

$$\begin{aligned}
\mathbf{h}^{(0)} &= \left[\mathbf{e}_{x_1}^{(t_1)}\mathbf{e}_{x_2}^{(t_2)}\cdots\mathbf{e}_{x_n}^{(t_n)}\right]^T && \text{(opération de look-up)}\\
\mathbf{h}^{(1)} &= g\left(\mathbf{W}^{(1)}\mathbf{h}^{(0)} + \mathbf{b}^{(1)}\right)\\
\mathbf{h}^{(2)} &= \text{softmax}\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right)\\
P(a|C;\boldsymbol{\theta}) &= h_a^{(2)}
\end{aligned}$$

Les paramètres du modèles sont alors $\boldsymbol{\theta} = (\mathbf{E}^{(\text{W})}, \mathbf{E}^{(\text{S})}, \mathbf{E}^{(\text{T})}, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)})$. On s'autorisera par la suite à utiliser une fonction d'activation g cubique introduite par Chen & Manning (2014) comme alternative à la fonction tangente hyperbolique.

4.3 Estimation des paramètres

Le corpus d'apprentissage est vu comme un ensemble $\mathcal{D} = \{(C^{(i)}, a^{(i)})\}_{i=1}^{N}$. Chaque $(C^{(i)}, a^{(i)})$ est un couple, configuration – action de référence, extrait à partir des séquences de dérivation de référence du corpus arboré.

Les modèles présentés modélisent la distribution $P(a^{(r)}|C;\boldsymbol{\theta})$. Chaque action est alors pondérée par sa log probabilité $f(a, C) = -logP(a|C;\boldsymbol{\theta})$. Pour estimer les paramètres, nous minimisons la log vraisemblance négative des données en supposant les données i.i.d. La fonction objective est la suivante :

$$\mathcal{L}(\mathcal{D};\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\log P(a^{(i)}|C^{(i)};\boldsymbol{\theta})$$

où $\boldsymbol{\theta}$ est l'ensemble des paramètres du modèle choisi. Les trois modèles sont optimisés en utilisant une descente de gradient stochastique dans son incarnation ADAGRAD (Duchi *et al.*, 2011). Cette méthode est moins sensible que la descente de gradient stochastique standard à la valeur du pas d'apprentissage. Elle permet également une convergence plus rapide. Pour chaque exemple, la valeur du gradient est calculée par l'algorithme de rétropropagation du gradient.

Il faut d'abord remarquer que le vecteur de paramètres $\boldsymbol{\theta}$ est initialisé par tirage au sort. Dans le cas du réseau profond, la fonction objective n'est pas convexe, et nous n'avons pas de garantie théorique de convergence vers un minimum global. On remarque finalement qu'on obtient un produit dérivé de l'estimation des paramètres. La couche cachée $\mathbf{h}^{(0)}$ des modèles superficiel et profond code des vecteurs de réels qui sont associés aux symboles d'entrée x_i et qui s'interprètent comme des plongements lexicaux, syntagmatiques ou morphosyntaxiques selon le type du symbole x_i.

5 Expériences

Les expériences s'appuient sur le jeu de données français SPMRL décrit dans (Abeillé *et al.*, 2003; Seddah *et al.*, 2013). Le jeu de données SPMRL instancie les données French Treebank dans deux scénarios : le scénario 'tags prédits' comporte un jeu de test où les tags de référence sont remplacés par des tags prédits par un tagger (exactitude d'étiquetage = 97.35%) et un scénario 'tags donnés' où le jeu de test comporte les tags de référence.

	Corpus d'entraînement			Corpus de développement	
Époque	F	Exactitude	Obj.	F	Exactitude
1	84.40	96.93	0.506	79.33	95.99
2	88.60	97.93	0.478	79.90	96.13
3	90.94	98.47	0.463	79.86	96.07
4	92.72	98.81	0.453	**80.16**	96.05
5	93.38	98.94	0.450	80.10	96.03
6	93.85	99.00	0.447	80.13	96.02

TABLE 2 – Données d'apprentissage pour le réseau profond dans le scénario 'tags donnés'. Les colonnes 'Exactitude' évaluent la qualité de la prédiction locale sur l'ensemble des couples configuration/action extraits des arbres de référence.

		tags donnés			**tags prédits**		
	apprentissage	F ≤ 40	F	Cov.	F ≤ 40	F	Cov.
Réseau profond (P+I)	local	83.5	**80.7**	99.96	81.3	**78.3**	99.96
Réseau superficiel (P)	local	80.6	77.3	99.96	79.1	75.6	99.96
Rég. log. multinomiale (RLM)	local	79.1	75.6	99.96	77.29	74.2	99.96
Perceptron moyenné	local	82.0	78.8	99.96	80.1	76.9	99.96
Perceptron moyenné (beam=4)	global	83.28	80.04	99.96	81.16	77.60	99.96
Berkeley (Petrov *et al.*, 2006)	global	86.4	84.0	99.96	83.2	80.7	99.96
Perc. moy. (beam=4, I + morpho.)	global	90.35	**87.84**	99.99	85.14	**82.38**	99.69

TABLE 3 – Résultats sur le corpus de test du FRENCH TREEBANK-SPMRL (P)longements, (I)nteractions.

Les expériences sont menées avec une implémentation de l'algorithme décrit dans l'article, écrite en C++/Eigen. Les arbres sont binarisés par une markovisation par la tête d'ordre 0. Les expériences sont réalisées sur les données de développement. Pour comparaison, nous donnons également les résultats sur les données de test pour un perceptron moyenné local et un perceptron moyenné structuré (beam = 4) qui utilisent des traits identiques aux modèles neuronaux (et ne prennent pas en compte d'interactions entre variables). On propose également deux derniers modèles qui donnent une idée de l'état de l'art : il s'agit de l'analyseur de Petrov *et al.* (2006), et enfin un modèle global – fondé sur des fonctions de traits riches incluant des interactions entre variables et des informations morphologiques supplémentaires notamment pour les mots composés – qui est une extension de (Crabbé, 2014b). Nous mesurons le F-Score et la couverture sur les données débinarisées à l'aide du logiciel `evalb` [3].

Les 3 modèles neuronaux ont été entraînés pour minimiser la fonction objective. L'apprentissage des réseaux de neurones est réputé sensible à de nombreux hyperparamètres. Pour calibrer les différents hyperparamètres et initialiser les matrices de poids, nous avons suivi les recommandations proposées par la communauté (Bengio, 2012; Do *et al.*, 2014). Le pas d'apprentissage initial est 0.04. Il est divisé par deux à la fin d'une itération si la fonction objective croît par rapport à l'itération précédente. Nous avons utilisé 300 unités dans la couche cachée pour le réseau profond. La dimension des plongements lexicaux d_{w} est fixée à 50. Celles des plongements de catégories syntagmatiques et de tags sont fixées à $d_{\mathrm{s}} = d_{\mathrm{w}} = 20$. Les coefficients initiaux des plongements sont tous initialisés aléatoirement dans l'intervalle $[-0.01, 0.01]$. Les matrices de poids des réseaux de neurones sont également initialisées aléatoirement. Nous avons comparé deux fonctions non linéaires pour le réseau profond : la tangente hyperbolique et la fonction cube proposée par Chen & Manning (2014). Nous présentons les résultats pour la fonction cube qui produit des résultats légèrement meilleurs que la tangente hyperbolique. Pour chaque modèle, le nombre d'itérations choisi est celui qui maximise le F-score sur le corpus de développement dans le scénario 'tags donnés' : 4 pour le réseau profond, 6 pour le réseau superficiel, 8 pour le modèle de RLM. On donne en table 2 les résultats de l'apprentissage sur le corpus de développement pour le réseau profond. Les 5 premiers modèles du tableau ont accès aux mêmes informations (traits) qui sont données au début de la section 4.2.

Les résultats sont présentés en table 3. Tout d'abord, on observe que la substitution de représentations denses à des représentations creuses entraîne un gain sur le modèle de RLM, sans augmenter l'expressivité de ce modèle. Cela suppose qu'une représentation structurée du lexique — et plus généralement des informations pertinentes pour désambiguïser — permet un meilleur apprentissage du comportement des mots et des relations entre les mots. L'utilisation d'une couche cachée non linéaire améliore encore ce résultat. Cela suggère que les classifieurs habituellement utilisés pour l'analyse syntaxique (perceptron, RLM) ne sont pas suffisamment puissants pour tirer pleinement parti des informations contenues dans les plongements lexicaux. La couche cachée permet d'apprendre des interactions entre les différents symboles et de sélectionner les informations pertinentes pour la désambiguïsation.

3. Nous utilisons la version standard du logiciel telle que distribuée sur le site `http://nlp.cs.nyu.edu/evalb`.

Le réseau de neurones profond utilisé apprend des représentations denses pour toutes les informations qu'il utilise lors de l'analyse. On donne en figure 5 une visualisation des plongements obtenus pour les symboles non-terminaux après une réduction de dimensionalité à l'aide de t-SNE (van der Maaten & Hinton, 2008). Notons que les représentations de symboles obtenus pourraient être en principe utilisées comme traits semi-supervisés pour toute autre tâche de traitement automatique des langues.

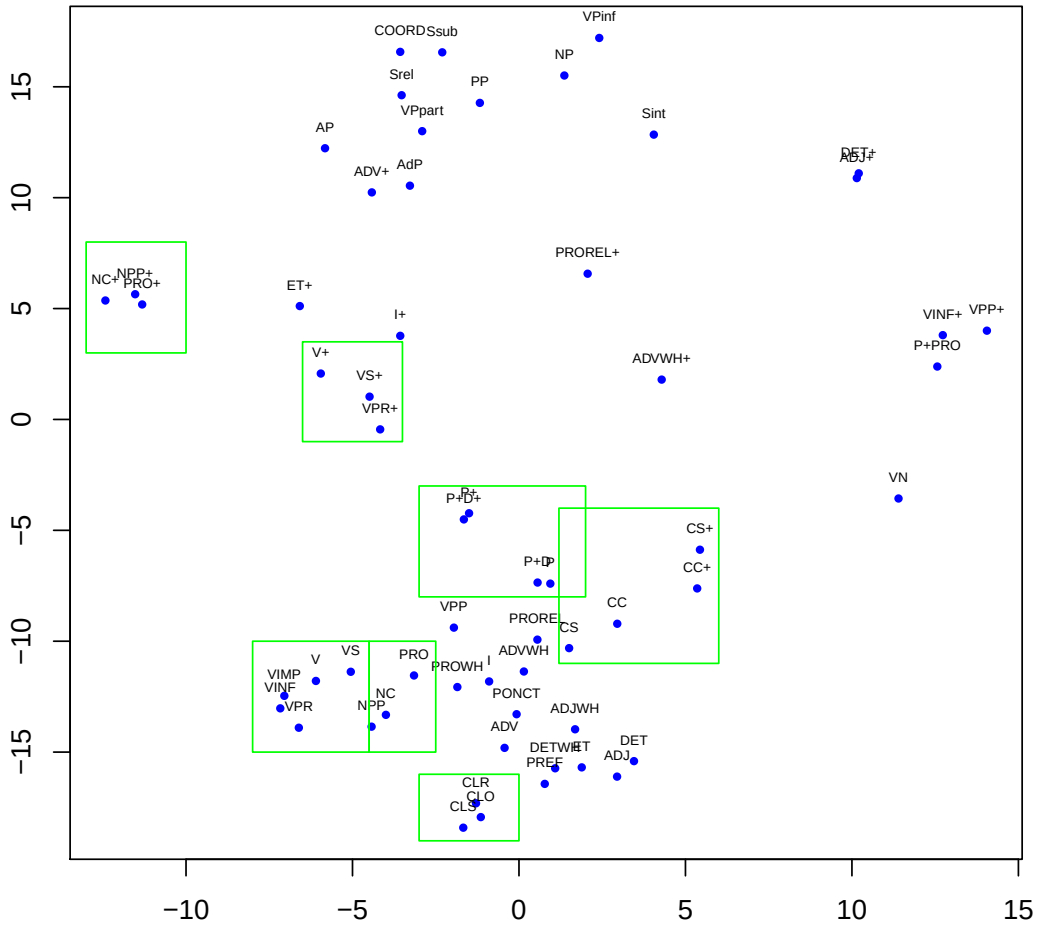


FIGURE 5 – Représentations denses de symboles non-terminaux apprises par le réseau de neurones profond. Pour leur visualisation, ces représentations ont été projetées sur 2 dimensions à l'aide de t-SNE. Le modèle parvient à capturer certaines similarités entre les différents symboles. Par exemple, les représentations pour différents tags de verbes (V, VS, VIMP, VINF, ...) sont proches, de même que celles pour les clitiques (CLO, CLS, CLR). Les symboles représentant des mots composés (suffixés par +) se regroupent entre eux ou avec leur contrepartie non composée.

L'architecture que nous proposons reste encore en deça des performances d'un analyseur génératif comme celui de Petrov *et al.* (2006), ou de perceptrons globaux comme celui de Crabbé (2014b). Nous envisageons plusieurs suites possibles pour améliorer ce résultat.

Tout d'abord, le modèle utilisé est un modèle local appliqué à un problème de classification structuré. Les modèles neuronaux présentés dans cet article utilisent tous une recherche gloutonne, de sorte que des erreurs en début d'analyse sont irrécupérables et peuvent compromettre la suite des opérations. Traditionnellement, étant donné la taille de l'espace de recherche ($\mathcal{O}(|A|^{3n-1})$), les analyseurs par transition s'appuient sur des algorithmes de recherche approximative pour rester efficace, le plus souvent une recherche par faisceau (Zhang & Clark, 2009; Zhu *et al.*, 2013; Crabbé, 2014a) ou une recherche de type meilleur d'abord (Zhao *et al.*, 2013). Or, des données empiriques (Zhang & Nivre, 2012) ont suggéré

	tags prédits		
Réseau profond	F <= 40	F	Cov.
beam=1	81.3	78.3	99.96
beam=2	81.63	78.5	99.96
beam=4	81.9	78.9	99.96
beam=8	81.9	78.9	99.96

TABLE 4 – Résultats pour le réseau profond avec une recherche par faisceau.

que, pour l'analyse syntaxique, l'utilisation d'un faisceau de recherche n'apporte un gain que dans le cas d'un algorithme d'apprentissage global. Cette observation se confirme pour l'architecture proposée (table 4), le gain est limité et plafonne avec une petite taille de faisceau. Ce gain est plus important si l'on passe du perceptron local au perceptron structuré.

De plus, la généralisation à la classification structurée pose problème. Reformuler un modèle neuronal global qui serait une contrepartie de CRF ou du perceptron structuré est rendu délicat par l'introduction de la couche cachée. Des méthodes hybrides ont été proposées pour la prédiction structurée à l'aide d'un réseau profond et de méthodes globales de type CRF (Wang & Manning, 2013). Mais celles-ci sont plus coûteuses en terme de temps de calcul. En revanche, un paradigme pour l'apprentissage structuré comme SEARN (Daumé III *et al.*, 2009), basé sur un classifieur local pourrait se révéler intéressant. De même, les méthodes plus traditionnelles (réseaux récurrents) qui cherchent à améliorer la modélisation de l'historique d'analyse ont donné de bons résultats également (Henderson, 2004; Titov & Henderson, 2007).

En second lieu, les architectures présentées apprennent elles-mêmes les représentations denses de symboles qu'elles utilisent. Au lieu de les initialiser aléatoirement, il est possible d'initialiser ces représentations pour les mots par des plongements lexicaux appris sur de gros volumes de données non annotées, par exemple à l'aide du modèle SKIP-GRAM (Mikolov *et al.*, 2013), ou d'un modèle de langue comme celui de Bengio *et al.* (2003). Cela permet généralement d'améliorer l'apprentissage (Chen & Manning, 2014).

Finalement, les sources d'informations utilisées par le classifieur sont encore relativement limitées si on regarde en comparaison les patrons de traits classiquement utilisés par les modèles d'analyse discriminants. Il serait en fait assez facile de donner au modèle accès à d'autres sources d'information : notons que si nous nous sommes limités à 3 types d'informations, le modèle proposé est assez général pour utiliser des représentations denses pour un nombre arbitraire de types de symboles , par exemple pour des informations morphologiques sous la forme de suffixes (voir également Collobert & Weston 2008).

6 Conclusion

L'article propose plusieurs architectures neuronales pour l'analyse automatique en constituants. Leur comparaison suggère 2 choses. En premier lieu, le remplacement de représentations creuses des symboles de la grammaire par des représentations denses permet d'améliorer l'estimation du modèle statistique. En second lieu, une architecture comprenant au moins une couche cachée non-linéaire tire mieux parti des informations données par ces plongements qu'un classifieur linéaire. La suite des travaux portera principalement sur le problème de la prédiction structurée et sur l'intégration de représentations pré-entraînées.

Références

ABEILLÉ A., CLÉMENT L. & TOUSSENEL F. (2003). Building a Treebank for French. In *Treebanks : Building and Using Parsed Corpora*, p. 165–188. Springer.

ANDREAS J. & KLEIN D. (2014). How much do word embeddings encode about syntax ? *Proceedings of ACL*.

BANSAL M., GIMPEL K. & LIVESCU K. (2014). Tailoring continuous word representations for dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2 : Short Papers)*, p. 809–815, Baltimore, Maryland : Association for Computational Linguistics.

BENGIO Y. (2012). *Practical recommendations for gradient-based training of deep architectures*. Rapport interne arXiv :1206.5533, U. Montreal, Lecture Notes in Computer Science Volume 7700, Neural Networks : Tricks of the Trade Second Edition, Editors : Grégoire Montavon, Geneviève B. Orr, Klaus-Robert Müller.

BENGIO Y., DUCHARME R. & VINCENT P. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, **3**, 1137–1155.

CANDITO M. & CRABBÉ B. (2009). Improving generative statistical parsing with semi-supervised word clustering. In *IWPT*, p. 138–141 : The Association for Computational Linguistics.

CHEN D. & MANNING C. D. (2014). A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

COLLOBERT R. & WESTON J. (2008). A unified architecture for natural language processing : Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, p. 160–167, New York, NY, USA : ACM.

CRABBÉ B. (2014a). An LR-inspired generalized lexicalized phrase structure parser. In *Proceedings of the twenty-fifth International Conference on Computational Linguistics*, Dublin, Ireland.

CRABBÉ B. (2014b). Un analyseur discriminant de la famille lr pour l'analyse en constituants. In *TALN*.

DAUMÉ III H., LANGFORD J. & MARCU D. (2009). Search-based structured prediction.

DO Q.-K., ALLAUZEN A. & YVON F. (2014). Modèles de langue neuronaux : une comparaison de plusieurs stratégies d'apprentissage. In *Actes de la 21e conférence sur le Traitement Automatique des Langues Naturelles*, p. 256–267, Marseille, France : Association pour le Traitement Automatique des Langues.

DUCHI J., HAZAN E. & SINGER Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, **12**, 2121–2159.

HENDERSON J. (2004). Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, p. 95–102, Barcelona, Spain.

KOO T., CARRERAS X. & COLLINS M. (2008). Simple semi-supervised dependency parsing. In *Proceedings of ACL-08 : HLT*, p. 595–603, Columbus, Ohio : Association for Computational Linguistics.

MIKOLOV T., CHEN K., CORRADO G. & DEAN J. (2013). Efficient estimation of word representations in vector space. *CoRR*, **abs/1301.3781**.

PETROV S., BARRETT L., THIBAUX R. & KLEIN D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, p. 433–440, Sydney, Australia : Association for Computational Linguistics.

SAGAE K. & LAVIE A. (2006). A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL on Main conference poster sessions*, p. 691–698 : Association for Computational Linguistics.

SEDDAH D., TSARFATY R., KÜBLER S., CANDITO M., CHOI J. D., FARKAS R., FOSTER J., GOENAGA I., GOJENOLA GALLETEBEITIA K., GOLDBERG Y., GREEN S., HABASH N., KUHLMANN M., MAIER W., NIVRE J., PRZEPIÓRKOWSKI A., ROTH R., SEEKER W., VERSLEY Y., VINCZE V., WOLIŃSKI M., WRÓBLEWSKA A. & DE LA CLERGERIE E. V. (2013). Overview of the SPMRL 2013 shared task : A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, p. 146–182, Seattle, Washington, USA : Association for Computational Linguistics.

SOCHER R., BAUER J., MANNING C. D. & NG A. Y. (2013). Parsing With Compositional Vector Grammars. In *ACL*.

TITOV I. & HENDERSON J. (2007). Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, p. 632–639, Prague, Czech Republic : Association for Computational Linguistics.

TURIAN J., RATINOV L. & BENGIO Y. (2010). Word representations : A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, p. 384–394, Stroudsburg, PA, USA : Association for Computational Linguistics.

VAN DER MAATEN L. & HINTON G. (2008). Visualizing high-dimensional data using t-sne.

WANG M. & MANNING C. D. (2013). Effect of non-linear deep architecture in sequence labeling. In *Proceedings of the 6th International Joint Conference on Natural Language Processing (IJCNLP)*.

ZHANG Y. & CLARK S. (2009). Transition-based parsing of the chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, p. 162–171, Stroudsburg, PA, USA : Association for Computational Linguistics.

ZHANG Y. & NIVRE J. (2012). Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *COLING (Posters)*, p. 1391–1400.

ZHAO K., CROSS J. & HUANG L. (2013). Optimal incremental parsing via best-first dynamic programming. In *EMNLP*, p. 758–768 : ACL.

ZHU M., ZHANG Y., CHEN W., ZHANG M. & ZHU J. (2013). Fast and accurate shift-reduce constituent parsing. In *ACL (1)*, p. 434–443 : The Association for Computer Linguistics.