Actes de la 29e Conférence sur le Traitement Automatique des Langues Naturelles
Avignon, France, 27 juin au 1er juillet 2022

Tâches auxiliaires pour l'analyse biaffine en graphes de dépendances

Marie Candito
Laboratoire de Linguistique Formelle, Université Paris Cité / CNRS,
8 place Paul Ricoeur 75013 Paris, France
marie.candito@u-paris.fr

RÉSUMÉ

L'analyseur biaffine de Dozat & Manning (2017), qui produit des arbres de dépendances syntaxiques, a été étendu avec succès aux graphes de dépendances syntaxico-sémantiques (Dozat & Manning, 2018). Ses performances sur les graphes sont étonnamment hautes étant donné que, sans la contrainte de devoir produire un arbre, les arcs pour une phrase donnée sont prédits indépendamment les uns des autres. Pour y remédier partiellement, tout en conservant la complexité $O(n^2)$ et l'architecture hautement parallélisable, nous proposons d'utiliser des tâches auxiliaires qui introduisent une forme d'interdépendance entre les arcs. Les expérimentations sur les trois jeux de données anglaises de la tâche 18 SemEval-2015 (Oepen *et al.*, 2015), et sur des graphes syntaxiques profonds en français (Ribeyre *et al.*, 2014) montrent une amélioration modeste mais systématique, par rapport à un système de base performant, utilisant un modèle de langue pré-entraîné. Notre méthode s'avère ainsi un moyen simple et robuste d'améliorer l'analyse vers graphes de dépendances.

ABSTRACT

Auxiliary tasks to boost Biaffine Semantic Dependency Parsing

The biaffine syntactic parser of Dozat & Manning (2017) was successfully extended to semantic dependency graph parsing (SDP) (Dozat & Manning, 2018). Its performance on graphs is surprisingly high given that, without the constraint of producing a tree, all arcs for a given sentence are predicted independently from each other. To address this issue, while retaining the $O(n^2)$ complexity and highly parallelizable architecture, we propose to use simple auxiliary tasks that introduce some form of interdependence between arcs. Experiments on the three English acyclic datasets of SemEval-2015 task 18 (Oepen *et al.*, 2015), and on French deep syntactic cyclic graphs (Ribeyre *et al.*, 2014) show modest but systematic performance gains on a near-SOTA baseline using transformer-based contextualized representations. This provides a simple and robust method to boost SDP performance.

MOTS-CLÉS : Analyse en graphes de dépendances ; Apprentissage multi-tâche.

KEYWORDS: Semantic dependency parsing ; Multi-task learning ; Biaffine dependency parsing.

1 Introduction et travaux reliés

Un graphe de dépendances est une structure utilisée pour représenter des dépendances entre mots au sein d'une phrase, dépassant le cadre syntaxique. Selon les jeux de données, ces dépendances peuvent encoder des relations prédicat-argument, où les étiquettes numérotent les arguments sémantiques (comme par exemple Figure 1-droite), ou bien des dépendances de statut intermédiaire entre syntaxe

Volume 1 : conférence principale, pages 424–433.

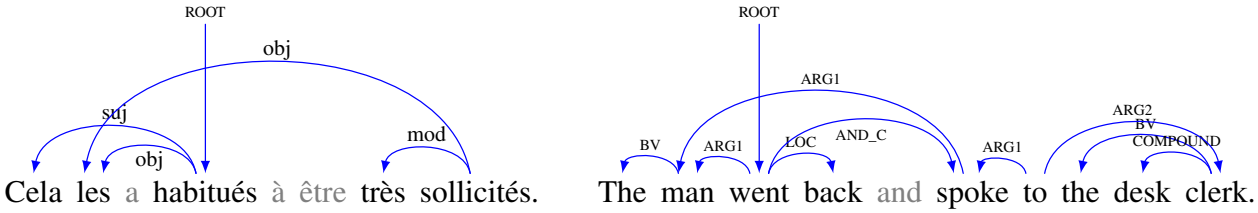

FIGURE 1 – **Gauche :** Graphe syntaxique profond dans le schéma de Candito *et al.* (2014). **Droite :** Graphe sémantique dans le format DM des données SemEval 2015 - tâche 18 (Oepen *et al.*, 2015). Les tokens n'ayant aucun arc entrant ni sortant sont considérés comme ne faisant pas partie de la représentation (en gris ci-dessus, les auxiliaires et prépositions régies par exemple).

et sémantique, étiquetées avec des fonctions "canoniques" qui neutralisent les alternances syntaxiques (par exemple, Figure 1-gauche, le clitique *les* est objet canonique de *sollicités*).

On distingue ainsi l'analyse vers *arbres* de dépendances syntaxiques versus l'analyse vers *graphes* de dépendances (ci-après **AGD**). Dans le premier cas, par construction, la contrainte d'arbre capture une interdépendance entre les décisions de rattachement : choisir un gouverneur pour un dépendant d bloque tout autre gouverneur pour d. En AGD, la tâche est plus difficile, car même si les arcs sont tout aussi interdépendants, le format ne contraint pas le nombre d'arcs entrants pour un mot.

Il existe plusieurs approches dans la littérature pour capturer l'interdépendance des arcs en AGD, qui sont souvent des adaptations d'analyseurs produisant des arbres de dépendances. L'une d'elles consiste à scorer le graphe de sortie via une somme de "facteurs" contenant plus d'un arc. Wang *et al.* (2019) améliorent l'état de l'art (sans utilisation de modèles de langue pré-entraînés) en utilisant des facteurs d'ordre deux pour scorer les graphes, mais au prix d'une complexité en $O(n^3)$.

Un autre grand type d'approche consiste à prendre des décisions séquentiellement, ce qui permet à un temps t d'encoder en entrée une partie du graphe construit jusqu'ici. C'est le cas dans l'analyseur à base de transitions de Fernández-González & Gómez-Rodríguez (2020) (ci-après **F&G20**), ou dans le système de Kurita & Søgaard (2019), qui à chaque itération sélectionne un nouveau gouverneur pour certains mots, en apprenant par renforcement une politique pour ordonner cette sélection de gouverneurs. Ces deux modèles ont une complexité en $O(n^2)$ (hors détection de cycles), avec des décisions séquentielles bénéficiant d'un encodage des arcs précédemment prédits. Pour pallier à la propagation d'erreur occasionnée, Bernard (2021) s'inspire de (Kurita & Søgaard, 2019), mais en autorisant son système à écraser des décisions précédentes et ainsi potentiellement se corriger.

Au contraire, l'analyseur biaffine de Dozat & Manning (2018) (ci-après **D&M18**) calcule simultanément le score de tous les arcs candidats, et les décisions de retenir tel ou tel arc sont indépendantes entre elles. On obtient un système en $O(n^2)$ hautement parallélisable, dont la performance est étonnamment haute, bien que moindre que l'analyse d'ordre 2 (Wang *et al.*, 2019).

Comme pour la plupart des tâches de TAL, les performances en AGD augmentent en encodant l'entrée via un modèle de langue pré-entraîné. Sur les données anglaises SemEval-2015-18 (Oepen *et al.*, 2015), F&G20 obtiennent une amélioration de +0,7 et +2,0 points de Fscore étiqueté, pour respectivement le test en-domaine (ID) et hors-domaine (OOD) [1].

1. Avec l'architecture de D&M18, He & Choi (2020) obtiennent +2 et +3 points en ID et OOD, mais ces auteurs ont utilisé un pré-traitement qui ajoute des arcs étiquetés "orphan" aux tokens non connectés, créant ainsi une tâche plus facile (communication personnelle avec les auteurs et https://github.com/emorynlp/bert-2019/issues/1).

Dans ce travail, nous retenons l'architecture simple en $O(n^2)$ de D&M18, et nous investiguons si des tâches auxiliaires simples peuvent introduire un peu d'interdépendance dans les décisions de rattachement, dans le cadre d'un apprentissage multi-tâche (Caruana, 1997). Nous montrons que certaines tâches auxiliaires apportent une amélioration modeste mais systématique sur les 3 jeux de données en anglais SemEval-2015-18 cités supra. On s'appuie également sur une propriété de D&M18, qui est l'absence totale de contraintes sur les graphes sortants. On teste ainsi sur les graphes syntaxiques profonds français précités, qui contiennent de nombreux cycles, pour lesquels on montre également le bénéfice de nos tâches auxiliaires pour la tâche d'AAG.

2 L'analyseur biaffine de base

On réutilise l'architecture de D&M18 pour le calcul des scores des arcs et de leurs étiquettes, en introduisant des différences uniquement dans l'encodage des mots de la phrase.

Encodage des tokens : nous avons choisi d'investiguer l'utilisation de tâches auxiliaires, en partant de système de base assez haut, et donc utilisant des représentations contextuelles : une séquence d'entrée $w_{1:n}$ est passée dans un modèle de langue pré-entraîné. On encode chaque w_i en concaténant un vecteur non contextuel $\mathbf{e}_i^{(\text{word})}$ au vecteur contextuel $\mathbf{h}_i^{(\text{bert})}$ [2], comme fait par exemple par He & Choi (2020) pour l'anglais et le chinois.

$$\begin{aligned} \mathbf{h}_i^{(\text{bert})} &= (\text{BERT}(w_{1:n}))_i \\ \mathbf{v}_i &= \mathbf{h}_i^{(\text{bert})} \oplus \mathbf{e}_i^{(\text{word})} \end{aligned} \tag{1}$$

Alternativement, pour les expériences de comparaison à **F&G20**, on concatène également un plongement de catégorie morphosyntaxique et un plongement de lemme :

$$\mathbf{v}_i = \mathbf{h}_i^{(\text{bert})} \oplus \mathbf{e}_i^{(\text{word})} \oplus \mathbf{e}_i^{(\text{lemma})} \oplus \mathbf{e}_i^{(\text{POS})} \tag{2}$$

Ensuite, plusieurs couches biLSTM sont appliquées à la séquence : $\mathbf{r}_{1:n} = \text{biLSTM}(\mathbf{v}_{1:n})$.

Spécialisation de l'encodage et scorage biaffine : Ensuite 4 MLP distincts sont utilisés pour spécialiser la représentation récurrente, selon que w_i joue le rôle de tête ou dépendant, et selon que l'on score un arc versus une étiquette d'arc :

$$\begin{aligned} \mathbf{h}_i^{(\text{arc-head})} &= \text{MLP}^{(\text{arc-head})}(\mathbf{r}_i) & \mathbf{h}_i^{(\text{lab-head})} &= \text{MLP}^{(\text{lab-head})}(\mathbf{r}_i) \\ \mathbf{h}_i^{(\text{arc-dep})} &= \text{MLP}^{(\text{arc-dep})}(\mathbf{r}_i) & \mathbf{h}_i^{(\text{lab-dep})} &= \text{MLP}^{(\text{lab-dep})}(\mathbf{r}_i) \end{aligned}$$

Nous utilisons une transformation biaffine simplifiée pour les scores des arcs $s_{i\to j}^{(\text{arc})}$, et une autre pour le score de chaque étiquette de dépendance l, noté $s_{i\to j}^{(\text{l})}$:

$$\begin{aligned} s_{i\to j}^{(\text{arc})} &= \mathbf{h}_j^{(\text{arc-dep})} \mathbf{U}^{(\text{arc})} \mathbf{h}_i^{(\text{arc-head})\top} + \mathbf{b}^{(\text{arc})} \\ s_{i\to j}^{(\text{l})} &= \mathbf{h}_j^{(\text{lab-dep})} \mathbf{U}^{(\text{l})} \mathbf{h}_i^{(\text{lab-head})\top} + \mathbf{b}^{(\text{l})} \end{aligned} \tag{3}$$

2. Dans toutes nos expériences, nous utilisons les vecteurs de type BERT à la dernière couche, et plus précisément le vecteur caché du premier sous-mot de chaque mot.

Apprentissage : pour une phrase ayant comme graphe de référence $\mathcal{Y}$, la perte utilisée somme (i) une partie concernant l'existence ou inexistence d'un arc entre chaque paire de positions dans la phrase, et (ii) une partie concernant les étiquettes des arcs. Pour la partie (i), on utilise l'entropie croisée binaire : chaque paire de positions i, j donne lieu à un terme $-log(\sigma(s^{(\text{arc})}_{i\to j}))$ si l'arc existe effectivement dans la référence, et $-log(1-\sigma(s^{(\text{arc})}_{i\to j}))$ sinon. Pour la partie (ii), on utilise l'entropie croisée pour l'étiquette des arcs de référence uniquement. Ainsi, la perte complète pour la phrase complète de graphe $\mathcal{Y}$ est la suivante :

$$\mathcal{L}(\mathcal{Y}) = -\sum_{i,j/i\to j\in\mathcal{Y}} \log(\sigma(s^{(\text{arc})}_{i\to j})) - \sum_{i,j/i\to j\notin\mathcal{Y}} \log(1-\sigma(s^{(\text{arc})}_{i\to j})) - \sum_{i,j/i\xrightarrow{l} j\in\mathcal{Y}} \log(\frac{s^{(l)}_{i\to j}}{\sum_{l'\in L} s^{(l')}_{i\to j}})$$

Inférence : pour toute paire de positions i, j, l'arc l'arc $i \to j$ est prédit dès lors que son score est positif (ce qui équivaut à ce que la probabilité d'existence de cet arc dépasse un demi, soit $\sigma(s^{(\text{arc})}_{i\to j}) > 0.5$), et il reçoit l'étiquette de plus haut score pour cet arc.

3 Tâches auxiliaires sur ensembles d'arcs

Nos premières expériences sur les graphes sémantiques anglais (Oepen *et al.*, 2015) et sur des graphes syntaxiques profonds en français (Ribeyre *et al.*, 2014) ont donné de bons résultats avec l'analyseur biaffine de base, mais avec des incohérences parfois surprenantes étant donnée la bonne qualité générale, et clairement reliées à la localité des décisions.

Un exemple d'incohérence concerne les expressions polylexicales. Dans les graphes anglais DM et les graphes syntaxiques profonds français, celles-ci sont représentées via une structure plate : un composant est la tête de tous les autres composants de l'expression (par exemple *garde* $\xrightarrow{mwe}$ *du*, et *garde* $\xrightarrow{mwe}$ *corps*). Nous avons remarqué que l'analyseur ne résoud pas toujours la compétition entre analyser une séquence comme un mot composé ou bien avec une structure syntaxique régulière (qui donnerait *garde* $\xrightarrow{dep}$ *du* $\xrightarrow{dep}$ *corps*). L'analyseur rattache ainsi parfois un même dépendant à la fois comme composant de composé et comme dépendant régulier, ce qui n'est pas possible dans le schéma d'annotation, et n'arrive effectivement jamais dans ces jeux de données.

En outre, pour les graphes en français, nous avons constaté une légère tendance à prédire trop souvent qu'un token est déconnecté, i.e. ne prédire pour lui aucun arc entrant ni sortant. Plus généralement, si l'on se concentre sur le nombre de gouverneurs de chaque mot, dans les graphes prédits anglais, 95% des mots reçoivent le bon nombre de gouverneurs, alors que ce n'est le cas que pour 92% des mots dans les graphes prédits français.

D'où l'idée d'utiliser des tâches auxiliaires prenant en compte tous les gouverneurs ou tous les dépendants d'un token donné. Plus précisément, on expérimente un apprentissage multi-tâche sur les deux tâches principales (tâche **A** et **L** pour la prédiction des arcs et de leurs étiquettes), plus les tâches auxiliaires suivantes, qui prédisent pour chaque token w_j :

- Tâches **H** et **D** : le nombre de gouverneurs et nombre de dépendants.
  - Par exemple pour le token ***spoke*** de la Figure 1, if faut prédire H=2 et D=1.
- Les étiquettes de tous les arcs entrants, sous deux formes différentes :

— Tâche **S** : la concaténation des étiquettes des arcs entrants, en ordre alphabétique.
  — Par exemple pour ***spoke***, l'étiquette complexe sera `AND_C+ARG1`).
— Tâche **B** : un vecteur creux pour le "bag of labels" (BOL), dont les composantes est le nombre d'arcs entrants vers w_j pour chaque étiquette.
  — Par exemple pour ***spoke***, le vecteur BOL portera un 1 pour les composantes de `AND_C` et `ARG1` et 0 partout ailleurs.

Ces tâches auxiliaires capturent indirectement les deux exemples d'incohérences donnés supra. Premièrement, dans le cas d'un token w_j qui est un composant de composé, la représentation récurrente $\mathbf{r}_j$ de ce token va être optimisée à l'apprentissage pour aboutir à une seule étiquette entrante étiqueté `mwe` pour les tâches S ou B, et une valeur 1 pour la tâche H (cf. un seul gouverneur pour w_j). On fait l'hypothèse que quand $\mathbf{r}_j$ est par ailleurs utilisée pour les tâches A et L, celle-ci favorisera de ne scorer positivement qu'un seul arc entrant, avec l'étiquette `mwe`. L'autre problème mentionné supra, i.e. le fait de prédire trop fréquemment un token déconnecté, est capturé par les tâches H et D (et également moins directement par les tâches B et S). Par exemple, ne prédire aucun gouverneur pour un token ne sera cohérent qu'avec H=0 (et S="). Ainsi, on espère que prédire H > 0 pour un token w_j favorisera des scores plus hauts pour les arcs entrants vers w_j.

Pour chaque tâche auxiliaire, un MLP spécifique est utilisé pour spécialiser la représentation récurrente r_j de chaque token w_j. Les tâches H et D sont des tâches de régression, pour lesquelles on utilise un MLP avec un seul neurone en sortie, et comme fonction de perte, l'erreur au carré moyenne [3].

$$nbh_j = \mathrm{MLP}^{(\mathrm{H})}(\mathbf{r}_j) \qquad nbd_j = \mathrm{MLP}^{(\mathrm{D})}(\mathbf{r}_j)$$

La tâche S est une tâche de classification vers des catégories correspondant à des multiensembles d'étiquettes de dépendances. Pour w_j, le vector des scores de tous les multiensembles s'écrit $\mathbf{s}_j = \mathrm{MLP}^{(\mathrm{S})}(\mathbf{r}_j)$, et l'entropie croisée est utilisée à l'apprentissage.

Pour la tâche B, le MLP de spécialisation a une couche de sortie de taille $|L|$: $\mathbf{BOL}_j = \mathrm{MLP}^{(\mathrm{B})}(\mathbf{r}_j)$. La composante pour l'étiquette l, BOL_{jl}, est interprétée comme $1+$ le log du nombre d'arcs de la forme $* \xrightarrow{l} j$. La perte utilisée est la distance L2 entre le vecteur BOL_j prédit et celui de référence.

Propagation en pile : On teste deux modes d'apprentissage multi-tâche : le premier est un simple partage de paramètres jusqu'aux couches biLSTM (c'est-à-dire que pour un token w_j, tous les MLP de spécialisation s'appliquent à la même représentation récurrente $\mathbf{r}_j$). Le second mode est la "propagation en pile" que Zhang & Weiss (2016) ont expérimentée pour les tâches de tagging et parsing : la couche cachée du réseau réalisant le tagging est utilisée dans l'encodage des tokens en entrée du réseau pour l'analyse syntaxique. Dans notre cas, cela revient à utiliser les couches cachées des MLP de spécialisation des tâches auxiliaires, dans la représentation d'entrée des tokens avant calcul biaffine des scores des arcs et des étiquettes.

Par exemple, pour utiliser la tâche H en mode propagation en pile, on modifie le calcul du score des arcs (cf. équation 3 supra). Soit $\mathrm{hidden}_j^{(\mathrm{H})}$ la couche cachée de $\mathrm{MLP}^{(\mathrm{H})}$ pour le dépendant j, et $c^{(\mathrm{H})}$ un coefficient hyperparamètre, le calcul de $s_{i\to j}^{(\mathrm{arc})}$ en mode propagation en pile s'écrit :

3. Plus précisément, pour la tâche H (resp. D), on prédit $1+$ le log des nombres de gouverneurs (resp. dépendants), de manière à moins pénaliser les erreurs sur les plus grands nombres. Par exemple prédire 1 gouverneur au lieu de 0 sera plus pénalisé que prédire 3 gouverneurs au lieu de 4.

$$\mathbf{sp}_j^{(\text{arc-dep})} = \mathbf{h}_j^{(\text{arc-dep})} \oplus c^{(\text{H})}\text{hidden}_j^{(\text{H})}$$
$$s_{i\to j}^{(\text{arc})} = \mathbf{sp}_j^{(\text{arc-dep})}\mathbf{U}^{(\text{arc})}\mathbf{h}_i^{(\text{arc-head})\top} + \mathbf{b}^{(\text{arc})}$$

4 Expériences et discussion

Données : On expérimente sur les trois jeux en anglais de Semeval-2015-18 (Oepen *et al.*, 2015) (DM, PAS and PSD), qui sont des graphes acycliques représentant principalement des relations prédicat-argument, ainsi que sur les graphes syntaxiques profonds (Ribeyre *et al.*, 2014) obtenus pour le corpus French Treebank (Abeillé & Barrier, 2004). Il s'agit de graphes "pseudo-gold", de bonne qualité cependant, obtenus par application de règles de transformation sur les arbres de dépendances gold. Ils capturent du partage d'arguments (en cas de montée, contrôle obligatoire ou arbitraire, partage de sujets en cas de coordination de VP etc...). Leurs étiquettes restent syntaxiques, mais l'utilisation de fonctions canoniques permet de neutraliser des alternances comme le passif. Des cycles peuvent apparaître, par exemple en cas de relatives.

Protocole expérimental : Nous avons choisi d'investiguer l'impact de nos tâches auxiliaires en partant d'une architecture état de l'art, i.e. intégrant modèles de langue pré-entraînés à base de "transformers". Nous utilisons notre propre implémentation de l'analyseur biaffine [4], et les modèles pré-entraînés $\text{BERT}_{\text{base-uncased}}$ (Devlin *et al.*, 2019) et $\text{FlauBERT}_{\text{base-cased}}$ (Le *et al.*, 2020) pour l'anglais et le français, via la librairie Huggingface (Wolf *et al.*, 2020).

Nous avons utilisé deux modes :

— **mode BERT_tuned** : ce premier mode est destiné à utiliser les représentations contextuelles comme seule source de paramètres pré-entraînés, et définit les vecteurs $\mathbf{v}_i$ comme dans l'équation (1) (pas de plongements de lemme ni de catégorie morpho-syntaxique), les plongements de mots étant initialisés au hasard, et les paramètres BERT étant modifiés à l'apprentissage (mode "réglage fin").

— **mode BERT_froz+POS+lem** : ce second mode est utilisé pour la comparaison aux travaux antérieurs, en particulier F&G20, sur les jeux de données anglais SemEval2015-Task18 : les paramètres BERT sont gelés, et on rajoute pour l'encodage des plongements de lemmes et catégories (cf. l'équation (2). Notez que cette configuration utilise les catégories et lemmes de référence, et ne constitue donc pas un scénario réaliste.

Nous avons réglé la plupart des hyperparamètres sur les données françaises. Après quelques tests nous avons fixé une combinaison d'hyperparamètres [5], et cherché la meilleure combinaison de tâches auxiliaires. Pour chaque configuration, nous reportons la Fmesure des arcs étiquetés (incluant les arcs étiquetés ROOT, cf. Figure 1), macro-moyennée sur 9 apprentissages.

4. https://github.com/mcandito/aux-tasks-biaffine-graph-parser-FindingsACL22

5. **Mode BERT_tuned :** Taille plongement de mots et lemmes : 100 ; dropout lexical : 0,4 ; biLSTM : 3 couches (600) avec dropout 0,33 ; MLPs pour arcs et étiquettes : cachée (600) dropout 0,33, sortie (600) ; MLPs pour tâches auxiliaires : cachée (300) dropout 0,25, sortie (300) ; Optimiseur : Adam ($\beta_1 = \beta_2$=0,9), taux d'apprentissage : 2×10^{-5} ; batchs : 8. L'apprentissage est arrêté lorsque tous les Fscores étiquetés diminuent sur l'ensemble dev (cf. en plus du LF principal, les tâches H et B donnent lieu à un Fscore, calculé en utilisant le nombre de têtes tel que prédit par ces tâches). **Mode BERT_froz+POS+lem :** Les plongements de mots et de lemmes sont ceux utilisés par F&G20, fournis par Ma *et al.* (2018) (taille 100) ; taux d'apprentissage : $0,0001$; MLP pour arcs (500,500), MLP pour étiquettes (100,100) (pour imiter (He & Choi, 2020)).

Résultats sur les graphes profonds français : Nous donnons Table 1 les résultats pour plusieurs combinaisons de tâches auxiliaires [6]. Utilisée de manière isolée, aucune tâche auxiliaire utilisée isolément ne donne de gain significatif. En revanche, en étudiant les résultats des combinaisons plus complexes, on observe que B+H, B+H+S et B+D+H+S donnent un gain statistiquement significatif [7]. On retient pour la suite comme meilleure configuration sans propagation en pile, la combinaison la plus simple, B+H, avec en moyenne un gain de +0,53 point de Fscore.

mode FlauBERT_tuned	Tâches aux.	Propagation en pile	LF moyen	écart-type
Version de base	$\emptyset$	NA	86,79	0,19
sans propag. en pile	H	non	86,82	0,54
	D	non	86,83	0,40
	S	non	86,98	0,30
	B	non	87,05	0,49
	B+H	non	87,32***	0,18
	D+H	non	86,61	0,71
	H+S	non	87,04	0,17
	D+H+S	non	86,86	0,39
	B+H+S	non	87,14**	0,39
	B+D+H S	non	**87,35*****	0,26
avec propag. en pile	B+H	$c^{(\mathrm{B})}$=1 $c^{(\mathrm{H})}$=1	87,49	0,06
	B+H	$c^{(\mathrm{B})}$=1 $c^{(\mathrm{H})}$=10	**87,66**$^{+++}$	0,18

TABLE 1 – Résultats sur l'ensemble **dev** français, en mode FlauBERT_tuned, pour diverses combinaisons de tâches, avec et sans propagation en pile. Col3-4 : Fscore étiqueté moyen et écart-type (sur 9 lancers). $^{***}/^{**}$: différence significative avec 1ère ligne (p <0.001/0.01). $^{+++}$: différence significative avec ligne B+H sans propagation en pile (p <0.001).

Nous avons retenu la combinaison B+H (plus simple que B+D+H+S), et testé l'impact de la propagation en pile. On observe un gain de $+0,34$, léger mais significatif, avec les poids $c^{(\mathrm{B})}$=1 et $c^{(\mathrm{H})}$=10 (bloc de lignes du bas de la table 1). Les résultats sur l'ensemble de test (table 2) confirment la tendance observée sur dev.

On peut également évaluer l'impact des tâches auxiliaires en comparant la précision, au sein des graphes prédits, du nombre de têtes de chaque mot : pour le système de base (sans tâches auxiliaires), en moyenne sur les 9 lancers, dans les graphes prédits environ $92,2\%$ des mots reçoivent le bon nombre de gouverneurs. Pour la configuration B+H sans propagation, cette proportion monte à $93,5\%$. Si ceci confirme bien l'impact des tâches auxiliaires, cette précision dans les graphes prédits est nettement en-dessous de la précision obtenue directement pour la tâche H, qui vaut en moyenne $96,5\%$. On en conclut qu'il reste une marge de progression dans la manière d'utiliser cette tâche.

6. L'état de l'art précédent sur ces données date d'un système non-neuronal : Ribeyre *et al.* (2016) obtenaient LF=80.86, et montaient à LF=84.91 grâce à des traits de l'analyseur à grammaire riche FrMG (Villemonte De La Clergerie, 2010).

7. Toutes les significativités sont évaluées avec un test de permutation exact de Fisher-Pitman (comme fait par ex. par Bernard (2021)). Ici nous considérons deux échantillons de Fscores, 9 pour la configuration A, et 9 pour la configuration B, avec en moyenne la configuration B meilleure que A. L'hypothèse nulle est que les deux échantillons suivent la même distribution. La valeur p correspond à la probabilité que la séparation de l'ensemble des Fscores en deux échantillons A' et B' de même taille que les deux initiaux (ici 9 et 9) donne une différence de moyenne au moins aussi grande que la différence observée. Avec le test exact, la valeur p est calculée exactement, sur toutes les séparations possibles en échantillons A' et B'.

mode FlauBERT_tuned	Tâches aux.	Propagation en pile	LF moyen	écart-type
Version de base	$\emptyset$	NA	86,76	0,34
sans propag. en pile	H	non	87,42***	0,38
avec propag. en pile	B+H	$c^{(B)}$=1 $c^{(H)}$=1	87,66***	0,19

TABLE 2 – Résultats sur l'ensemble de **test** français, en mode FlauBERT_tuned, dans trois configurations : sans tâche auxiliaire, avec tâches B+H sans propagation, avec tâches B+H et propagation en pile. Col3-4 : Fscore étiqueté moyen et écart-type (sur 9 lancers). *** : différence significative avec 1ère ligne (p <0.001).

Résultats sur les graphes anglais : Pour l'anglais, nous avons retenu la configuration B+H sans propagation en pile, considérant que le gain de la propagation est minime au regard du désavantage qui est qu'avec propagation, les tâches auxiliaires doivent être prédites également à l'inférence. Table 3 donne les résultats sur les jeux de test anglais, pour les 3 formalismes (DM, PAS, PSD) et les deux genres (en domaine (ID) et hors domaine (OOD)). On observe un gain modeste mais systématique sur les 6 corpus de test, ce qui tend à montrer une certaine robustesse de la méthode.

Tâches		DM	PAS	PSD	Moy.		DM	PAS	PSD	Moy.
$\emptyset$	ID	93,7	93,9	80,7	89,4	OOD	90,3	92,0	79,8	87,4
B+H		94,2	94,3	81,2	89,9		91,0	92,8	80,2	88,0

TABLE 3 – Fscore étiqueté moyen (sur 9 lancers) pour les jeux de test anglais "en domaine" (ID) et "hors domaine" (OOD), en utilisant soit aucune tâche auxiliaire ($\emptyset$) soit les tâches B et H (B+H), sans propagation en pile. Les résultats B+H sont significativement plus hauts que $\emptyset$ pour DM ID, DM OOD, PAS ID, PAS OOD (p <0.001) et PSD ID (p <0.01).

Dans les travaux antérieurs, l'analyseur le plus performant et plus comparable en termes d'encodage des tokens d'entrée est le parser par transitions de F&G20 (FL=$90,7$ en ID, et FL=$88,8$ en OOD). Pour s'y comparer, on passe au mode BERT_froz+POS+lem. On obtient FL_ID=$90,2$ et FL_OOD=$87,9$ des résultats donc clairement en-dessous. Mais signalons que la méthode que nous proposons peut être utilisée avec tout système neuronal, et potentiellement apporter des gains orthogonaux au changement d'algorithme d'analyse.

5 Conclusion

Un analyseur biaffine pour l'analyse en graphes de dépendances prédit les arcs indépendamment les uns des autres. Nous avons proposé des tâches auxiliaires introduisant une forme d'interdépendance dans les décisions de rattachement. Nous avons montré qu'entraîner des représentations récurrentes de mots à prédire à la fois les graphes cibles, et les nombres de têtes et étiquettes entrantes de chaque token donne des gains certes modestes mais systématiques, sur des graphes syntaxiques profonds français et des graphes sémantiques anglais, en- et hors-domaine. Ce multi-apprentissage est peu coûteux et peut être utilisé pour tout système utilisant un encodage récurrent, quel que soit l'algorithme d'analyse.

Références

ABEILLÉ A. & BARRIER N. (2004). Enriching a French treebank. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC'04)*, Lisbon, Portugal : European Language Resources Association (ELRA).

BERNARD T. (2021). Multiple tasks integration : Tagging, syntactic and semantic parsing as a single task. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics : Main Volume*, p. 783–794, Online : Association for Computational Linguistics.

CANDITO M., PERRIER G., GUILLAUME B., RIBEYRE C., FORT K., SEDDAH D. & DE LA CLERGERIE É. (2014). Deep syntax annotation of the sequoia French treebank. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, p. 2298–2305, Reykjavik, Iceland : European Language Resources Association (ELRA).

CARUANA R. (1997). Multitask learning. *Machine learning*, **28**, 41–75. DOI : https ://link.springer.com/article/10.1023%2FA%3A1007379606734.

DEVLIN J., CHANG M.-W., LEE K. & TOUTANOVA K. (2019). BERT : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, p. 4171–4186, Minneapolis, Minnesota : Association for Computational Linguistics. DOI : 10.18653/v1/N19-1423.

DOZAT T. & MANNING C. D. (2017). Deep Biaffine Attention for Neural Dependency Parsing. In *5th International Conference on Learning Representations, ICLR 2017, Conference Track Proceedings*, Toulon, France : OpenReview.net.

DOZAT T. & MANNING C. D. (2018). Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2 : Short Papers)*, p. 484–490, Melbourne, Australia : Association for Computational Linguistics. DOI : 10.18653/v1/P18-2077.

FERNÁNDEZ-GONZÁLEZ D. & GÓMEZ-RODRÍGUEZ C. (2020). Transition-based semantic dependency parsing with pointer networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, p. 7035–7046, Online : Association for Computational Linguistics. DOI : 10.18653/v1/2020.acl-main.629.

HE H. & CHOI J. (2020). Establishing strong baselines for the new decade : Sequence tagging, syntactic and semantic parsing with BERT. In *Proceedings of Florida Artificial Intelligence Research Society Conference FLAIRS-33*, p. 228–233.

KURITA S. & SØGAARD A. (2019). Multi-task semantic dependency parsing with policy gradient for learning easy-first strategies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 2420–2430, Florence, Italy : Association for Computational Linguistics. DOI : 10.18653/v1/P19-1232.

LE H., VIAL L., FREJ J., SEGONNE V., COAVOUX M., LECOUTEUX B., ALLAUZEN A., CRABBÉ B., BESACIER L. & SCHWAB D. (2020). FlauBERT : Unsupervised language model pre-training for French. In *Proceedings of the 12th Language Resources and Evaluation Conference*, p. 2479–2490, Marseille, France : European Language Resources Association.

MA X., HU Z., LIU J., PENG N., NEUBIG G. & HOVY E. (2018). Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computatio-*

nal Linguistics (Volume 1 : Long Papers), p. 1403–1414, Melbourne, Australia : Association for Computational Linguistics. DOI : 10.18653/v1/P18-1130.

OEPEN S., KUHLMANN M., MIYAO Y., ZEMAN D., CINKOVÁ S., FLICKINGER D., HAJIČ J. & UREŠOVÁ Z. (2015). SemEval 2015 task 18 : Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, p. 915–926, Denver, Colorado : Association for Computational Linguistics. DOI : 10.18653/v1/S15-2153.

RIBEYRE C., CANDITO M. & SEDDAH D. (2014). Semi-Automatic Deep Syntactic Annotations of the French Treebank. In *Proceedings of the 13th International Workshop on Treebanks and Linguistic Theories (TLT13)*, p. 184–197, Tübingen, Germany.

RIBEYRE C., VILLEMONTE DE LA CLERGERIE E. & SEDDAH D. (2016). Accurate deep syntactic parsing of graphs : The case of French. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, p. 3563–3568, Portorož, Slovenia : European Language Resources Association (ELRA).

VILLEMONTE DE LA CLERGERIE É. (2010). Convertir des dérivations TAG en dépendances. In *Actes de la 17e conférence sur le Traitement Automatique des Langues Naturelles. Articles longs*, p. 91–100, Montréal, Canada : ATALA.

WANG X., HUANG J. & TU K. (2019). Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 4609–4618, Florence, Italy : Association for Computational Linguistics. DOI : 10.18653/v1/P19-1454.

WOLF T., DEBUT L., SANH V., CHAUMOND J., DELANGUE C., MOI A., CISTAC P., RAULT T., LOUF R., FUNTOWICZ M., DAVISON J., SHLEIFER S., VON PLATEN P., MA C., JERNITE Y., PLU J., XU C., LE SCAO T., GUGGER S., DRAME M., LHOEST Q. & RUSH A. (2020). Transformers : State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing : System Demonstrations*, p. 38–45, Online : Association for Computational Linguistics. DOI : 10.18653/v1/2020.emnlp-demos.6.

ZHANG Y. & WEISS D. (2016). Stack-propagation : Improved representation learning for syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 1557–1566, Berlin, Germany : Association for Computational Linguistics. DOI : 10.18653/v1/P16-1147.