Actes de CORIA-TALN 2023
Paris, France, 5 au 9 juin 2023

Attention sur les *spans* pour l'analyse syntaxique en constituants

RÉSUMÉ

Nous présentons une extension aux analyseurs syntaxiques en constituants neuronaux modernes qui consiste à doter les constituants potentiels d'une représentation vectorielle affinée en fonction du contexte par plusieurs applications successives d'un module de type transformer efficace (*pooling* par attention puis transformation non-linéaire). Nous appliquons cette extension à l'analyseur CRF de Zhang *et al.* (2020). Expérimentalement, nous testons cette extension sur deux corpus (PTB et FTB) avec ou sans vecteurs de mots dynamiques : cette extension permet d'avoir un gain constant dans toutes les configurations.

ABSTRACT

Attention over spans for syntactic parsing

We introduce an extension to recent neural constituency parsers that consists of providing potential constituents with a vector representation fine-tuned from the context by successive applications of an efficient transformer type module (pooling by attention, then a non-linear transformation). We implement this extension in the parser called CRF introduced by Zhang *et al.* (2020). We test this extension on two corpora, PTB and FTB, with or without dynamic word vectors : this extension achieves performance gains in all settings.

MOTS-CLÉS : Apprentissage profond, Attention, Analyse syntaxique en constituants.

KEYWORDS: Deep learning, Attention, Constituency Parsing.

1 Introduction

L'analyse syntaxique en constituants crée un arbre hiérarchique à partir d'une phrase d'entrée. Les feuilles de cet arbre sont les mots de la phrase et les nœuds internes sont les constituants. En tant que tâche fondamentale du TAL, l'analyse en constituants sert de base à de nombreuses applications en aval, notamment l'analyse sémantique ou l'extraction d'information, et il est donc crucial de produire correctement les structures syntaxiques nécessaires aux traitements ultérieurs. De nombreux modèles actuels (Stern *et al.*, 2017; Kitaev & Klein, 2018; Zhang *et al.*, 2020) fonctionnent en deux temps. Tout d'abord ils attribuent des scores aux différentes sous-chaînes (nous utiliserons le terme anglais *spans* dans la suite pour désigner les sous-chaînes) selon leur plausibilité d'être un constituant bien typé (groupe nominal, verbal, prépositionnel…). Ensuite un algorithme combinatoire, souvent CKY (Kasami, 1965; Baker, 1979), permet de construire l'arbre d'analyse de meilleure score. Ce score a en général une interprétation probabiliste, et l'analyse syntaxique revient donc dans ce cas à retourner l'arbre le plus probable (MAP, maximum a posteriori), ou qui combine les constituants les plus probables (MBR, *Minimum Bayes Risk*). La première étape, celle qui attribue les scores, est réalisée par une architecture neuronale construite à partir d'un module qui extrait des vecteurs caractéristiques des mots de la phrase (*extracteur*) puis un second module qui assemble les vecteurs de mots pour représenter les spans et qui leur attribue un score (*scoreur*). Un mécanisme de récurrence ou d'attention permet aux vecteurs de mots de prendre en compte le contexte, c'est-à-dire de s'affiner en fonction des autres mots dans la phrase.

Actes de la 30e Conférence sur le Traitement Automatique des Langues Naturelles (TALN), volume 2 : travaux de recherche originaux – articles courts, pages 37–45.

Dans cet article nous étendons cette prise en compte du contexte aux spans. Pour cela, il faut d'abord *représenter* les spans, dans le cas des réseaux de neurones leur attribuer un vecteur réel caractéristique. Puis nous ajoutons des *transformers* dotés d'un mécanisme d'attention linéaire pour faire évoluer ces représentations en fonctions des autres spans de la forêt d'analyse (sans toutefois calculer cette forêt explicitement). Enfin à partir des représentations finales nous calculons les scores des spans. Expérimentalement, cette addition de transformers sur les représentations des spans dans un analyseur récent au plus haut de l'état de l'art actuel permet d'améliorer les performances sur plusieurs corpus de référence.

Dans la Section 2 nous présentons le modèle d'analyseur CRF que nous utilisons ainsi que l'extension que nous proposons pour représenter les spans. En Section 3 nous présentons l'évaluation sur deux corpus dans différents scénarios. Puis en Section 4 nous présentons quelques travaux connexes avant de conclure.

2 Modèle

L'analyse syntaxique en constituants consiste à décomposer une phrase en ses parties constituantes, qui sont organisées dans une structure hiérarchique basée sur leurs relations syntaxiques. Formellement, étant donné une phrase $\boldsymbol{x} = w_1 \dots w_n$, un arbre d'analyse $\boldsymbol{t}$ est un ensemble de syntagmes $(i, j, l) \in \boldsymbol{t}$ couvrant les mots $w_i \dots w_j$ et ayant une étiquette syntaxique $l \in \mathcal{L}$. Ils forment une segmentation hiérarchique qui couvre toute la phrase. Dans cet article, nous proposons une nouvelle représentation des syntagmes basée sur l'attention que nous intégrons dans l'analyseur CRF à deux étapes de Zhang *et al.* (2020).

2.1 Analyseur CRF à deux étapes

Modèle d'analyse et algorithmes. Notre analyseur de base construit $\boldsymbol{t}$ en deux étapes : la première génère un arbre non étiqueté $\mathbf{y} = \{(i, j)\}_{i<j, i,j \in [1,n]}$ contraint à l'ensemble de toutes les segmentations hiérarchiques valides de $\boldsymbol{x}$, appelé $\mathcal{Y}_{\boldsymbol{x}}$. La deuxième étape attribue des étiquettes à ses spans. L'analyseur fonctionne sur des arbres en Forme Normale de Chomsky où un segment *non étiqueté* (i, j) peut être formé en joignant deux spans adjacents (i, k) et $(k + 1, j)$.

Afin de comparer des arbres d'analyse alternatifs pour une entrée $\boldsymbol{x}$, le modèle utilise une fonction de notation d'ordre zéro qui se factorise en terme des scores individuels de spans : $s(\boldsymbol{y}, \boldsymbol{x}) = \sum_{(i,j)\in \boldsymbol{y}} s_c(i, j; \boldsymbol{x})$. Ces scores sont globalement normalisés pour obtenir une distribution sur les arbres valides : $\log p(\boldsymbol{y}|\boldsymbol{x}) = s_c(\boldsymbol{y}, \boldsymbol{x}) - A(\boldsymbol{x})$. La constante de normalisation $A(\boldsymbol{x}) = \log \sum_{\boldsymbol{y}\in\mathcal{Y}_{\boldsymbol{x}}} \exp(s(\boldsymbol{y}, \boldsymbol{x}))$ est calculée à l'aide de l'algorithme *inside* en temps $O(n^3)$. Pendant le décodage, l'algorithme CKY est utilisé pour trouver l'arbre optimal étant donné le modèle : $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}\in\mathcal{Y}_{\boldsymbol{x}}} s(\boldsymbol{y}, \boldsymbol{x})$. CKY est presque identique à *inside* mais maximise au lieu de sommer sur les sous-arbres valides avec une complexité similaire. CKY peut également être utilisé avec le décodage MBR (Minimum Bayes-Risk) en remplaçant $s_c(i, j; \boldsymbol{x})$ par la probabilité marginale du segment $p(i, j|\boldsymbol{x})$. Ces probabilités peuvent être calculées efficacement avec l'algorithme *outside*.

Dans la deuxième étape, une étiquette est sélectionnée pour chaque segment dans l'arbre décodé $\forall(i, j) \in \hat{\boldsymbol{y}} : \hat{l} = \arg\max_{l\in\mathcal{L}} s_l(i, j, l; \boldsymbol{x})$. La complexité de cette étape est de $O(n|\mathcal{L}|)$, car $\hat{\boldsymbol{y}}$ ne contient que $2n - 1$ spans qui doivent être scorés parmi les $O(n^2)$ spans de $\boldsymbol{x}$. Les probabilités des étiquettes $p(l|i, j; \boldsymbol{x})$ sont obtenues en normalisant localement les scores des étiquettes à l'aide de la fonction softmax.

Représentations et fonctions de score La fonction de score s_c opère sur les représentations des bornes des spans i et j. Chaque mot $w_i \in \boldsymbol{x}$ peut jouer le rôle d'une borne *gauche* ou *droite* avec une sémantique différente. Par conséquent, nous construisons une représentation spécialisée pour chaque rôle à l'aide d'un MLP (*Multilayer Perceptron*, Perceptron multicouche), similaire aux travaux de Stern *et al.* (2017) et Zhang *et al.* (2020). Autrement dit, pour chaque mot w_i, nous calculons $\boldsymbol{g}_i = \text{MLP}^g(\boldsymbol{h}_i)$ et $\boldsymbol{d}_i = \text{MLP}^d(\boldsymbol{h}_i)$ où $\boldsymbol{h}_i \in \mathbb{R}^k$ est une représentation contextualisée de w_i et $\boldsymbol{g}_i, \boldsymbol{d}_i \in \mathbb{R}^d$. La fonction de score $s_c(i,j) = \text{Biaffine}(\boldsymbol{g}_i, \boldsymbol{d}_j; \boldsymbol{W})$ est une fonction biaffine (Dozat & Manning, 2017) paramétrée par $\boldsymbol{W} \in \mathbb{R}^{d \times d}$. La représentation du mot $\boldsymbol{h}_i \in \mathbb{R}^k$ est calculée en deux étapes : 1) un encodage initial $\boldsymbol{e}_i \in \mathbb{R}^{k'}$ est obtenu soit en combinant des *embeddings* statiques tel que fournis par GloVe (Pennington *et al.*, 2014), ou fastText (Grave *et al.*, 2018) avec un encodage BiLSTM de la séquence de caractères d'entrée ; soit en utilisant un modèle de langue pré-entraîné tel que BERT (Devlin *et al.*, 2019) ; 2) la deuxième étape fait passer l'encodage initial par un BiLSTM (Dozat & Manning, 2017) à trois couches pour obtenir l'encodage final $\boldsymbol{h}_i$.

Des calculs similaires sont effectués pour scorer les étiquettes : $s_l(i,j,l;\boldsymbol{x}) = \text{Biaffine}(\bar{\boldsymbol{g}}_i, \bar{\boldsymbol{d}}_j; \boldsymbol{W}[l])$ où $\boldsymbol{W} \in \mathbb{R}^{|\mathcal{L}| \times \bar{d} \times \bar{d}}$ regroupe tous les paramètres de s_l en un seul tenseur.

Apprentissage et implémentation Les paramètres des fonctions de scores sont appris pour minimiser la somme de deux fonctions de perte : la log-vraisemblance négative (NLL) des arbres de références dans une *treebank* $\mathcal{T}$ et la NLL des étiquettes individuelles des spans de ces arbres : $\mathcal{L}(x,y) = -\sum_{(\boldsymbol{y},\boldsymbol{x}) \in \mathcal{T}} \log p(\boldsymbol{y}|\boldsymbol{x}) - \sum_{\boldsymbol{y} \in \mathcal{T}; (i,j,l) \in \boldsymbol{y}} p(l|i,j;\boldsymbol{x})$.

L'apprentissage et le décodage sont particulièrement efficaces grâce à une implémentation *batchifiée* des algorithmes inside et CKY pour un calcul direct sur GPU. L'algorithme outside nécessaire au calcul des marginaux $p(i,j|\boldsymbol{x})$ est implémenté efficacement en utilisant la différenciation automatique car ces probabilités correspondent aux gradients du constant de normalisation $A(\boldsymbol{x})$ (Eisner, 2016).

2.2 Représentations des spans raffinées par attention

Notre objectif est d'affiner les représentations vectorielles biaffines des spans en permettant à leurs vecteurs initiaux d'interagir par le biais de l'attention. Les encodeurs basés sur le transformer (Devlin *et al.*, 2019) sont l'architecture de référence pour l'auto-attention mais ont une complexité quadratique puisque, pour chaque vecteur d'entrée, les poids d'attention doivent être calculés sur l'ensemble de l'entrée. Cette complexité quadratique est prohibitive dans notre cas puisque nous avons $O(n^2)$ spans pour n mots ce qui porte la complexité du transformer à $O(n^4)$. Pour résoudre ce problème, nous avons recours aux transformers linéaires (Katharopoulos *et al.*, 2020a; Choromanski *et al.*, 2021), une famille de transformers efficaces qui utilisent des noyaux pour calculer l'attention avec une complexité spatio-temporelle linéaire. Plus précisément, nous empruntons la couche NormAttention à TransNormer (Qin *et al.*, 2022), conçue pour résoudre le problème des gradients non bornés responsables d'instabilités dans l'apprentissage des transforers linéaires à base de noyaux.

Étant donné les représentations initiales des spans $\mathbf{H}$, nous commençons par calculer les requêtes $\mathbf{Q} = \mathbf{H}\boldsymbol{W}_Q$, les clés $\mathbf{K} = \mathbf{H}\boldsymbol{W}_K$ et les valeurs $\mathbf{V} = \mathbf{H}\boldsymbol{W}_V$, tous des vecteurs dans $\mathbb{R}^d$. Contrairement aux transformers standards qui calculent l'attention comme $\text{Softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d})\mathbf{V}$ qui a des gradients non bornés et une complexité quadratique (Qin *et al.*, 2022), nous calculons la version linéaire de l'attention et utilisons LayerNorm (Ba *et al.*, 2016) pour le borner : $\text{LayerNorm}(\mathbf{Q}(\mathbf{K}^\top\mathbf{V}))$. Les autres opérations du bloc transformer sont standards, y compris les opérations de *skip* connexion et

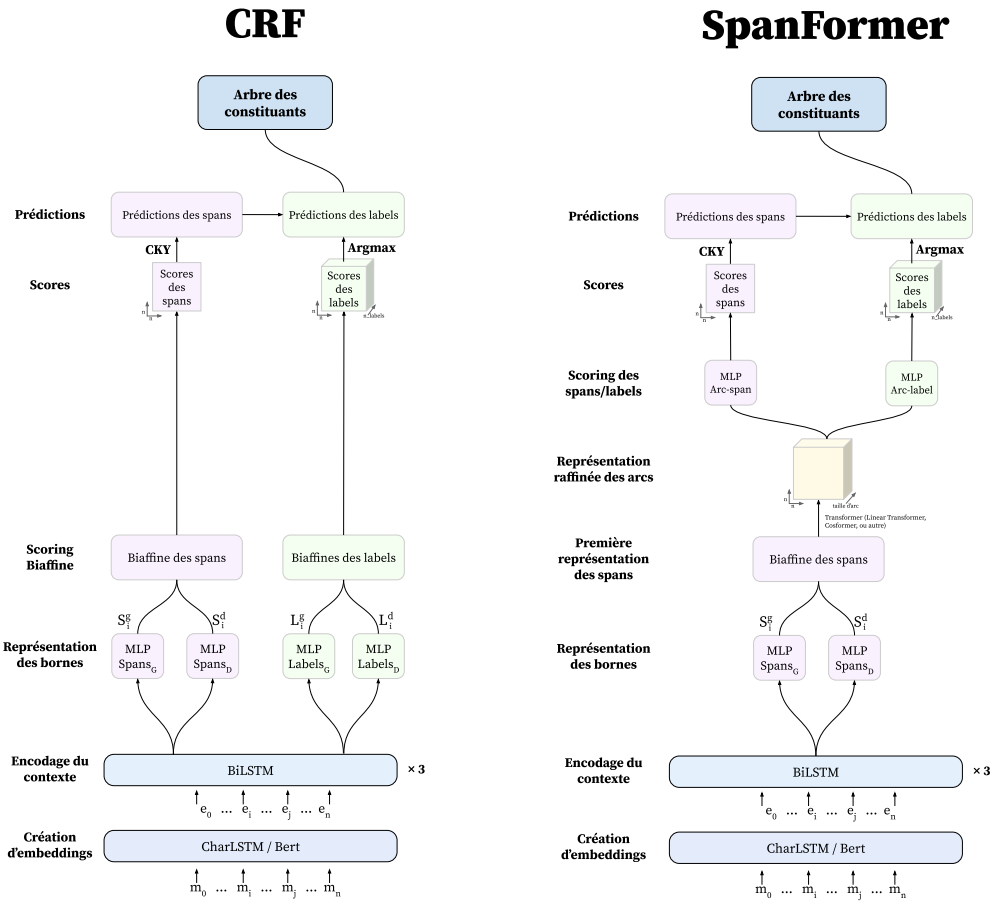

FIGURE 1 – Comparaison entre CRF et notre modèle SpanFormer, schéma adapté de la figure 2 de Zhang *et al.* (2020)

de perceptron multicouches. Nous utilisons plusieurs têtes d'attention et empilons plusieurs couches de transformer pour calculer les représentations finales des spans.

3 Expériences

Jeux de données Nos expériences sont effectuées sur le Penn TreeBank (PTB) en anglais et le corpus en français FTB (Abeillé *et al.*, 2000) de SPMRL (Seddah *et al.*, 2013) afin d'avoir des résultats sur plusieurs langues et sur des *treebanks* de différentes tailles. Nous suivons la séparation usuelle train/dev/test et nous entraînons nos modèles sur des GPUs Nvidia A40. Nous avons utilisé les vecteurs de mots anglais Glove (Pennington *et al.*, 2014) ou BERT large cased (Devlin *et al.*, 2018) sur le PTB et les vecteurs de mots français fastText (Grave *et al.*, 2018) ou XLM-RoBERTa large (Conneau *et al.*, 2019) sur SPMRL.

Évaluation Nous reprenons les conditions d'évaluation de Zhang *et al.* (2020), comme eux nous utilisons donc la précision, le recall et le F-score (P/R/F) des arbres étiquetés avec l'outil d'évaluation du parenthésage EVALB, avec les configurations standards d'évaulation pour le PTB.

	Dev				Test			
	P	R	F	σ(F)	P	R	F	σ(F)
PTB (CharLSTM + GloVe)								
CRF (Zhang *et al.*, 2020)	94.03	94.25	94.14	0.07	94.14	93.90	94.02	0.06
(Kitaev & Klein, 2018)∗	–	–	–		93.90	93.20	93.55	
SpanFormer	94.16	94.17	**94.17**	0.09	94.34	93.92	**94.13**	0.09
PTB (BERT large cased)								
CRF (Zhang *et al.*, 2020)	95.78	95.89	95.83	0.06	96.01	95.55	95.78	0.08
Kitaev *et al.* (2019)∗	–	–	–		95.73	95.46	95.59	
Yang & Deng (2020)	–	–	–		96.04	95.55	95.79	
SpanFormer	95.80	95.90	**95.85**	0.04	96.02	95.58	**95.80**	0.07
FTB (CharLSTM + fastText)								
CRF (Zhang *et al.*, 2020)	84.48	85.23	84.86	0.09	83.94	84.59	84.26	0.15
SpanFormer	84.79	85.12	**84.96**	0.07	84.29	84.48	**84.39**	0.08
FTB (XLM-RoBERTa large cased)								
CRF (Zhang *et al.*, 2020)	88.50	88.82	88.66	0.08	88.87	89.15	89.01	0.25
SpanFormer	88.70	88.73	**88.76**	0.15	89.06	89.00	**89.04**	0.08

TABLE 1 – Comparaison de nos résultats sur CRF (Zhang *et al.*, 2020), notre SpanFormer, et les résultats publiés d'autres modèles la métrique déterminant le meilleur modèle est le F-Score (moyennes et écart-types de 10 expériences avec différentes valeurs initiales des paramètres pour nos expériences).
∗ Résultats reportés par Zhang *et al.* (2020)

Paramètres Nous reprenons les paramètres de Zhang *et al.* (2020), nous utilisons aussi un charLSTM pour nos représentations initiales des mots, pour GloVe sur le PTB nous avons des *char embeddings* de dimension 50, les *word embeddings* et une sortie de charLSTM sont aussi de taille 100. (Pour fastText sur SPMRL, les *word embeddings* sont de taille 300). Les *dropout* valent 0.33 et nous utilisons aussi une taille de batch de 5000 tokens.

Pour les configurations avec GloVe/fastText, la patience est de 100, c'est à dire que notre entraînement s'arrête si le F-score du dev n'augmente pas pendant 100 époques. Avec BERT le nombre d'époques est fixé à 10. Pour BERT/XLM-RoBERTa, nous avons repris exactement les configurations données et nous avons aussi ajouté 3 arguments visiblement manquants au fichier de configuration, *weight_decay*, *decay* et *decay_steps* valant 0.01, 0.75 et 5000 respectivement. Le *learning rate* reste de 0.002 pour GloVe/fastText et 5×10^{-5} pour BERT/XLM-RoBERTa. Le *learning rate* du transformer est de 0.001.

3.1 Résultats

Comme on peut le voir dans la table 1, notre approche donne les meilleurs résultats pour chaque configuration sur une moyenne de 10 expériences, notamment avec une amélioration importante pour le FTB avec fastText.

Tailles des vecteurs de spans La taille des vecteurs représentant les spans doit être assez grande pour qu'ils soient porteurs d'information, mais assez petite pour limiter le coût en mémoire des $O(n^2)$

spans. Nos expériences ont montré qu'une taille entre 20 et 60 donne de bons scores, les résultats de la table 1 sont obtenus avec des tailles de 48, et 3 têtes d'attention pour le transformer. Nous avons obtenus de meilleurs résultats en produisant de grands spans de taille 480 avec la fonction biaffine, et en les projetant vers une taille plus petite de 48 avant de les raffiner avec le transformer.

Profondeur Nous avons testé nos modèles avec 1 à 4 couches de transformers et nous avons remarqué qu'en moyenne les scores n'augmentent pas significativement avec plus de couches, et qu'une suffit amplement.

Vitesse et mémoire Notre modèle SpanFormer est légèrement plus lent que CRF à l'entraînement, ce qui est dû au transformer, bien qu'efficace il a un coup conséquent qui fait qu'on traite 1080 phrases par seconde (p/s), contre 1319 pour CRF, soit 75% de la vitesse initiale. En revanche, lors de l'évaluation et du test, les vitesses sont plus similaires avec $607p/s$ pour SpanFormer et $624p/s$ pour CRF. La demande en mémoire est elle nettement accrue, typiquement un modèle avec une taille de span de 48 et 1 couche nécessitera environ 10Go de mémoire pour l'entraînement quand CRF n'avait besoin que de 5Go.

4 Travaux connexes

Notre méthode de calcul de représentation des spans via une attention globale est similaire aux *Edge Transformers* de Bergen *et al.* (2021) qui avaient proposé une attention globale sur les arcs pour les analyses en dépendances. Outre les formalismes d'analyse (constituants vs. dépendances), nous soulignons deux différences majeures : premièrement nous n'utilisons pas de masque d'attention particulier [1] ce qui, deuxièmement, nous oblige à utiliser des transformers efficaces vu que le nombre d'items en relation d'attention est quadratique dans la taille d'une phrase [2], ce qui crée un problème d'utilisation mémoire si l'on veut utiliser des co-processeurs spécialisés, de type GPU. Nous utilisons des Transformers linéaires (Katharopoulos *et al.*, 2020b) qui, en évitant l'opération de normalisation des compatibilité requête-clé par softmax, garantissent une complexité en espace linéaire dans le nombre d'items, donc quadratique dans la taille de la phrase d'entrée.

D'autres travaux tels que ceux de Zaratiana *et al.* (2022) montrent l'intérêt d'enrichir les représentations des spans afin d'obtenir de meilleures prédictions dans les tâches en TAL, ce que nous faisons aussi à l'aide d'un transformer.

L'attention est devenue une étape essentielle dans les analyseurs [3] depuis l'article de Dozat & Manning (2017). Nous notons que cette attention était déjà non-normalisée comme c'est le cas dans les transformers efficaces actuels. La plupart des analyseurs à l'état de l'art actuel, utilisent une attention particulière adaptée à l'analyse. Par exemple, Le Roux *et al.* (2019) utilisent une attention croisée spécialisée pour représenter les étapes d'une analyse en transitions, tandis que Kitaev & Klein (2018) travaillent sur la relation entre l'attention entre les contenus lexicaux et les contenus positionnels.

1. (Bergen *et al.*, 2021) utilisent l'attention *triangulaire* censée encourager la découverte de relations de transitivité logiques.
2. Il y a $2n^2 - n$ arcs en considérant une phrase de n mots complémentée d'une racine fictive.
3. Nous parlons de l'attention dans la partie des analyseurs propre à l'analyse syntaxique ou sémantique. La plupart des analyseurs utilisent comme la majorité des systèmes actuels en TAL des transformers pour plonger les mots dans des espaces vectoriels.

5 Conclusion

Nous proposons un modèle d'analyse syntaxique en constituants faisant usage de représentations vectorielles des spans raffinées à l'aide d'un transformer. Notre méthode mène à de meilleures performances sur toutes les configurations testées, ces améliorations sont conséquentes sur le FTB avec fastText. De plus, nous arrivons à ces performances avec un coût additionnel en temps raisonnable grâce à notre utilisation de transformers linéaires. Dans de futurs travaux nous essayerons de cibler l'attention pour mieux prendre en compte les incompatibilités entre certains spans et nous appliquerons notre approche dans différentes tâches du TAL.

Références

ABEILLÉ A., CLÉMENT L. & KINYON A. (2000). Building a treebank for French. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*, Athens, Greece : European Language Resources Association (ELRA).

BA L. J., KIROS J. R. & HINTON G. E. (2016). Layer normalization. *CoRR*, **abs/1607.06450**.

BAKER J. K. (1979). Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, **65**(S1), S132–S132. DOI : 10.1121/1.2017061.

BERGEN L., O'DONNELL T. J. & BAHDANAU D. (2021). Systematic generalization with edge transformers. In A. BEYGELZIMER, Y. DAUPHIN, P. LIANG & J. W. VAUGHAN, Éds., *Advances in Neural Information Processing Systems*.

CHOROMANSKI K. M., LIKHOSHERSTOV V., DOHAN D., SONG X., GANE A., SARLOS T., HAWKINS P., DAVIS J. Q., MOHIUDDIN A., KAISER L., BELANGER D. B., COLWELL L. J. & WELLER A. (2021). Rethinking attention with performers. In *International Conference on Learning Representations*.

CONNEAU A., KHANDELWAL K., GOYAL N., CHAUDHARY V., WENZEK G., GUZMÁN F., GRAVE E., OTT M., ZETTLEMOYER L. & STOYANOV V. (2019). Unsupervised cross-lingual representation learning at scale. *CoRR*, **abs/1911.02116**.

DEVLIN J., CHANG M., LEE K. & TOUTANOVA K. (2018). BERT : pre-training of deep bidirectional transformers for language understanding. *CoRR*, **abs/1810.04805**.

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 *International Conference on Learning Representations*.

EISNER J. (2016). Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP*, p. 1–17, Austin, TX : Association for Computational Linguistics. DOI : 10.18653/v1/W16-5901.

GRAVE E., BOJANOWSKI P., GUPTA P., JOULIN A. & MIKOLOV T. (2018). Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

KASAMI T. (1965). *An efficient recognition and syntax analysis algorithm for context-free languages*. Rapport interne, Air Force Cambridge Research Laboratory.

KATHAROPOULOS A., VYAS A., PAPPAS N. & FLEURET F. (2020a). Transformers are rnns : Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20 : JMLR.org.

KATHAROPOULOS A., VYAS A., PAPPAS N. & FLEURET F. (2020b). Transformers are rnns : Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20 : JMLR.org.

KITAEV N., CAO S. & KLEIN D. (2019). Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, p. 3499–3505, Florence, Italy : Association for Computational Linguistics. DOI : 10.18653/v1/P19-1340.

KITAEV N. & KLEIN D. (2018). Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 2676–2686, Melbourne, Australia : Association for Computational Linguistics. DOI : 10.18653/v1/P18-1249.

LE ROUX J., ROZENKNOP A. & LACROIX M. (2019). Representation learning and dynamic programming for arc-hybrid parsing. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, p. 238–248, Hong Kong, China : Association for Computational Linguistics. DOI : 10.18653/v1/K19-1023.

PENNINGTON J., SOCHER R. & MANNING C. D. (2014). Glove : Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, p. 1532–1543.

QIN Z., HAN X., SUN W., LI D., KONG L., BARNES N. & ZHONG Y. (2022). The devil in linear transformer. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, p. 7025–7041, Abu Dhabi, United Arab Emirates : 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. & VILLEMONTE DE LA CLERGERIE E. (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.

STERN M., ANDREAS J. & KLEIN D. (2017). A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 818–827, Vancouver, Canada : Association for Computational Linguistics. DOI : 10.18653/v1/P17-1076.

YANG K. & DENG J. (2020). Strongly incremental constituency parsing with graph neural networks. In H. LAROCHELLE, M. RANZATO, R. HADSELL, M. BALCAN & H. LIN, Éds., *Advances in Neural Information Processing Systems 33 : Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

ZARATIANA U., TOMEH N., HOLAT P. & CHARNOIS T. (2022). GNNer : Reducing overlapping in span-based NER using graph neural networks. In *Proceedings of the 60th Annual Meeting of*

the Association for Computational Linguistics : Student Research Workshop, p. 97–103, Dublin, Ireland : Association for Computational Linguistics. DOI : 10.18653/v1/2022.acl-srw.9.

ZHANG Y., ZHOU H. & LI Z. (2020). Fast and accurate neural crf constituency parsing. In C. BESSIERE, Éd., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, p. 4046–4053 : International Joint Conferences on Artificial Intelligence Organization. Main track, DOI : 10.24963/ijcai.2020/560.