Revisiting Higher-Order Dependency Parsers

Erick Fonseca
Instituto de Telecomunicações
Lisbon, Portugal
erick.fonseca@lx.it.pt

André F. T. Martins
Instituto de Telecomunicações & Unbabel
Lisbon, Portugal
andre.t.martins@tecnico.ulisboa.pt

Abstract

Neural encoders have allowed dependency parsers to shift from higher-order structured models to simpler first-order ones, making decoding faster and still achieving better accuracy than non-neural parsers. This has led to a belief that neural encoders can implicitly encode structural constraints, such as siblings and grandparents in a tree. We tested this hypothesis and found that neural parsers may benefit from higher-order features, even when employing a powerful pre-trained encoder, such as BERT. While the gains of higher-order features are small in the presence of a powerful encoder, they are consistent for long-range dependencies and long sentences. In particular, higher-order models are more accurate on full sentence parses and on the exact match of modifier lists, indicating that they deal better with larger, more complex structures.

1 Introduction

Before the advent of neural networks in NLP, dependency parsers relied on higher-order features to better model sentence structure (McDonald and Pereira, 2006; Carreras, 2007; Koo and Collins, 2010; Martins et al., 2013, *inter alia*). Common choices for such features were siblings (a head word and two modifiers) and grandparents (a head word, its own head and a modifier).

Kiperwasser and Goldberg (2016) showed that even without higher order features, a parser with an RNN encoder could achieve state-of-the-art results. This led folk wisdom to suggest that modeling higher-order features in a neural parser would not bring additional advantages, and nearly all recent research on dependency parsing was restricted to first-order models (Dozat and Manning, 2016; Smith et al., 2018a). Kulmizev et al. (2019) further reinforced this belief comparing transition and graph-based decoders (but none of which higher order); Falenska and Kuhn (2019) suggested that higher-order features become redundant because the parsing models encode them implicitly.

However, there is some evidence that neural parsers still benefit from structure modeling. Zhang et al. (2019) showed that a parser trained with a global structure loss function has higher accuracy than when trained with a local objective (i.e., learning the head of each word independently). Falenska and Kuhn (2019) examined the impact of consecutive sibling features in a neural dependency parser. While they found mostly negative results in a transition-based setting, a graph-based parser still showed significant gains on two out of 10 treebanks.

In this paper, we test rigorously the hypothesis of the utility of second-order features. In particular, we experiment with **consecutive sibling** and **grandparent** features in a **non-projective**, graph-based dependency parser. We found that without a pretrained encoder, these features are only useful for large treebanks; however, when using BERT, they can improve performance on most treebanks we tested on — especially true for longer sentences and long-distance dependencies, and full sentence parses[1]. This challenges the hypothesis that encoders can single-handedly improve parsers, or more generally, structured models in general.

2 Model

2.1 Notation

We use $\mathbf{x}$ to refer to a sentence with tokens $(x_1, x_2, \ldots, x_n)$, plus the `ROOT` pseudo-token, and $\mathbf{y}$ to refer to a valid tree composed of n arcs (h, m).

We overload the notation $s_\theta(\cdot)$ to indicate the model score for a part or complete sentence, de-

[1] Our code is available at https://github.com/deep-spin/pyturbo/

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 8795–8800
July 5 - 10, 2020.

pending on its arguments.

2.2 Encoding

We encode a $\mathbf{x}$ with a bidirectional LSTM, producing hidden states $(\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_n)$, with $\mathbf{h}_0$ corresponding to `ROOT`. Each token is represented by the concatenation of its pretrained word embeddings, a character-level left-to-right LSTM and, optionally, BERT embeddings.

Similar to Straka et al. (2019), when using BERT, we take the mean of its last four layers. When the BERT tokenizer splits a token into more than one, we take the first one and ignore the rest, and we use the special token `[CLS]` to represent `ROOT`. The word embeddings we use are the ones provided in the CoNLL 2018 shared task.

2.3 First-Order Model

We start with a first-order model, which is used as a pruner before running the second-order parser as in Martins et al. (2013). It uses biaffine attention to compute arc and label scores (Dozat and Manning, 2016), and similarly to Qi et al. (2018), we also add distance and linearization terms.[2]

We want our pruner to be capable of estimating arc probabilities, and thus we train it with a marginal inference loss, maximizing the log probability of the correct parse tree $\mathbf{y}$:

$$\begin{aligned}\mathcal{L}_\theta(\mathbf{x}, \mathbf{y}) &= -\log p_\theta(\mathbf{y} \mid \mathbf{x}) \\ &= -s_\theta(\mathbf{y}) + \log \sum_i \exp(s_\theta(\mathbf{y}_i)).\end{aligned}$$

We can compute the partition function over all possible trees $\mathbf{y}_i$ efficiently using the Matrix-Tree Theorem (Koo et al., 2007), which also gives us arc marginal probabilities. The sentence score $s_\theta(\mathbf{x}, \mathbf{y})$ is computed as the sum of the score of its parts.

Additionally, we try first-order models trained with a hinge loss, as Zhang et al. (2019) (also used with our second-order models; see §2.4), maximizing the margin between the correct parse tree $\mathbf{y}$ and any other tree $\hat{\mathbf{y}}$:

$$\mathcal{L}_\theta(\mathbf{x}, \mathbf{y}) = \max_{\hat{\mathbf{y}}} [s_\theta(\mathbf{x}, \hat{\mathbf{y}}) - s_\theta(\mathbf{x}, \mathbf{y}) + \Delta(\mathbf{y}, \hat{\mathbf{y}})],$$

where $\Delta(\mathbf{y}, \hat{\mathbf{y}})$ is the Hamming cost between $\mathbf{y}$ and $\hat{\mathbf{y}}$, i.e., the number of arcs in which they differ.

[2] We refer the reader to Qi et al. (2018) for further definition of the distance and linearization terms. Also, like them, we only backpropagate error for these scores for the gold arcs.

2.4 Second-Order Model

We train second-order models with a hinge loss. It is computed in the same way as in the first-order case, except now the sentence scores include second-order parts. Notice that the Hamming cost still only considers differing arcs.

Consecutive siblings A consecutive sibling part is a tuple (h, m, s) such that h is the parent of both m and s, which are both to the left or to the right of h, and no other child of h exists between them. Additionally, we consider tuples $(h, m, \emptyset)$ to indicate that m is the first child (if to the left of h) or the last child (if to the right).

Grandparents A grandparent part is a tuple (h, m, g) such that g is the parent of h and h is the parent of m. There are no grandparent parts such that h is `ROOT`.

Scoring The score for a higher order part (h, m, r) of type ρ (in our case, either grandparent or consecutive sibling) is computed as:

$$\begin{aligned}s_\theta(h, m, r) = \mathbf{w}^{\rho\top} \cdot (&\lambda_1^\rho \tanh(\mathbf{h}_h^\rho + \mathbf{h}_r^\rho) \\ &+ \lambda_2^\rho \tanh(\mathbf{h}_m^\rho + \mathbf{h}_r^\rho) \\ &+ \lambda_3^\rho \tanh(\mathbf{h}_h^\rho + \mathbf{h}_m^\rho + \mathbf{h}_r^\rho)),\end{aligned}$$

$$\mathbf{h}_h^\rho = f_h^\rho(\mathbf{h}_h), \mathbf{h}_m^\rho = f_m^\rho(\mathbf{h}_m), \mathbf{h}_r^\rho = f_r^\rho(\mathbf{h}_r).$$

where λ_1^ρ, λ_2^ρ and λ_3^ρ are learnable scalars, $\mathbf{w}^\rho$ is a learnable vector, $f_h^\rho(\cdot)$, $f_m^\rho(\cdot)$ and $f_r^\rho(\cdot)$ are learnable affine transforms. There is a set of these parameters for consecutive siblings and another for grandparents. The factors that compose the score represent different combinations of a second-order part with h, m, or both. There is no factor combining h and m only, since they are already present in the first-order scoring. We also introduce a parameter vector $\mathbf{h}_\emptyset$ to account for $\emptyset$.

Decoding The drawback of higher-order feature templates is that exact decoding is intractable for the non-projective case. Classically, researchers have resorted to approximate decoding as well as using a first-order parser to eliminate unlikely arcs and their respective higher-order parts. We employ both of these techniques; specifically, we use the dual decomposition algorithm AD^3 (Martins et al., 2011, 2013) for decoding, which often arrives at the exact solution. We use head automata factors

to handle sibling and grandparent structures (Koo et al., 2010), and the traditional Chu-Liu-Edmonds algorithm to handle the tree constraint factor (McDonald et al., 2005).

2.5 Additional Training Details

Multitask Learning Our models also predict UPOS, XPOS and morphology tags (UFeats), as training for these additional objectives increases parsing performance. They are implemented via softmax layers on top of the BiLSTM output, and have a cross-entropy loss. Parser and tagger share two BiLSTM layers, with an additional layer for each one (similar to Straka, 2018). We only consider UFeats singletons in the training data, i.e., we do not decompose them into individual features.

Perturb and MAP During training with a hinge loss, we add noise sampled from a standard Gumbel distribution to the arc scores, as in Papandreou and Yuille (2011). This effectively makes decoding behave as sampling from the tree space.

3 Experiments

Data We evaluate our models on 19 treebanks from Universal Dependencies 2.3: Afrikaans (AfriBooms), Ancient Greek (Perseus), Arabic (PADT), Basque (BDT), Chinese (GSD), Czech (PDT), Finnish (TDT), Hebrew (HTB), Hindi (HDTB), Hungarian (Szeged), Italian (ISDT), Japanese (GSD), Korean (GSD), Persian (Seraji), Portuguese (Bosque), Russian (SynTagRUS), Swedish (Talbanken) and Turkish (IMST). In all cases, we use gold tokenization. They represent varied language families, writing systems and typology, inspired by Smith et al. (2018b).

Hyperparameters All LSTM cells have 400 units in each direction, as well as arc and label biaffine projections. Second-order layers have 200 units, and character embeddings have 250. We apply dropout with $p = 0.5$ to all linear layers, and we use word dropout (replacing an encoded word vector with a trainable vector) with $p = 0.33$ in models without BERT and 0.2 in the ones with it. We use Adam with $\beta_1 = 0.9$, $\beta_2 = 0.99$, and constant learning rate of 10^{-3} for the first-order models without BERT and $5 \cdot 10^{-4}$ for all others. We used `bert-chinese` for Chinese and Japanese, and `bert-base-multilingual-cased` for other languages; and did not fine-tune its weights. We run the AD^3 decoder for up to 500 iterations with a step size of 0.05. We use batches of 1,000 tokens for first-order models and 800 for second-order, and train for up to 100k batches. We evaluate on the dev set each 200 batches and stop early after 50 evaluations without improvement.

Pruning Before training or evaluating a second-order parser, we run a first-order model trained with marginal inference to prune unlikely arcs and any second-order parts including them. When using BERT in the main parser, we also use a pruner trained with BERT. We keep up to 10 candidate heads for each token, and further prune arcs with posterior probability lower than a threshold t times the probability of the most likely head. Without BERT, $t = 10^{-6}$, and with it $t = 10^{-8}$, as we found BERT makes the pruner overconfident. The lowest pruner recall on the dev set was 98.91% (on Turkish); all other treebanks are above 99%. During training, we never prune out gold arcs.

3.1 Results

Table 1 shows the test set UAS and LAS for our models. Parsers with BERT and hinge loss achieve the best performance in most datasets; second-order models are generally better at UAS. An interesting case is Ancient Greek, which is not in BERT's pretraining data. First-order models with BERT perform worse than the ones without it in UAS and LAS, but the second-order model achieves the highest UAS.

Without BERT, second-order features are only beneficial in some medium-to-large treebanks. In the smallest ones, as Turkish and Hungarian, they actually lead to a performance drop; when using BERT, however, they increase accuracy in these datasets. On the other hand, large treebanks such as Russian and Czech have improvements from second-order features even without BERT. This suggests that in order for them to be beneficial, either large amounts of annotated training data are needed (which not all UD treebanks have) or a powerful encoder such as BERT.

Considering first-order models, Zhang et al. (2019) found no particular advantage of a hinge loss objective over a cross-entropy one or vice-versa. In our experiments, this is mostly the case for models trained with small-to-medium treebanks and without BERT. When more training data or a pretrained encoder is available, the hinge loss objective tends to reach higher accuracy than the cross-entropy one.

		First Order Marginal		First Order Hinge		Second Order Hinge		FO + BERT Marginal		FO + BERT Hinge		SO + BERT Hinge	
	Tokens	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
AF	33.8k	88.08	85.24	88.38	85.15	87.93	84.85	90.54	87.99	**90.96**	**88.22**	90.66	88.03
AR	223.8k	88.07	83.51	88.36	83.62	88.37	83.71	88.37	83.79	88.78	84.16	**88.97**	**84.29**
CS	1.1M	92.35	89.88	92.91	90.44	93.25	90.89	93.61	91.49	**93.96**	**91.79**	93.90	91.71
EN	204.6k	89.82	87.15	90.02	87.29	90.20	87.53	92.51	90.22	**92.80**	**90.53**	92.63	90.31
EU	72.9k	86.32	83.02	86.35	83.02	86.24	82.66	**87.42**	**84.11**	87.34	83.93	**87.42**	84.03
FA	121.1k	90.76	87.15	90.59	87.33	90.60	86.97	91.95	88.79	**92.27**	**89.14**	91.91	88.83
FI	162.6k	90.51	88.20	90.97	88.69	91.07	88.90	**91.84**	**89.81**	91.66	89.38	91.72	89.47
GRC	159.8k	79.81	74.40	80.11	74.61	80.12	**74.74**	79.72	73.94	78.61	72.51	**80.33**	74.33
HE	137.7k	89.65	86.86	89.89	87.10	89.56	86.67	91.00	88.25	**91.44**	**88.59**	91.25	88.43
HI	281k	94.79	91.52	95.12	91.97	95.03	91.86	95.26	92.00	95.30	**92.24**	**95.34**	92.11
HU	20.2k	83.02	77.78	83.66	78.26	82.30	76.97	87.71	83.21	**87.90**	83.21	86.62	**82.38**
IT	276k	93.35	91.27	93.63	91.65	93.65	91.64	94.98	93.28	95.23	**93.53**	**95.25**	93.42
JA	160.4k	94.82	93.21	94.76	93.25	94.19	92.56	95.14	**93.62**	95.07	**93.62**	**95.18**	**93.62**
KO	56.6k	86.89	82.97	87.69	84.00	88.02	84.16	89.06	85.69	**89.71**	**86.33**	89.62	86.26
PT	206.7k	91.76	89.37	91.59	88.95	92.09	89.64	92.55	90.20	92.63	90.14	**92.97**	**90.58**
RU	870.4k	93.02	91.14	93.43	91.51	93.87	92.06	94.47	93.01	94.51	92.98	**94.70**	**93.16**
SV	66.6k	89.50	86.62	89.31	86.16	87.00	83.95	91.49	88.93	91.79	**89.31**	**91.82**	89.08
TR	37.9k	74.48	67.63	72.42	65.22	73.30	65.86	74.59	67.96	75.43	68.72	**75.66**	**68.88**
ZH	98.6k	85.06	80.94	84.98	80.65	84.97	80.40	90.08	87.32	90.03	87.17	**90.43**	**87.53**

Table 1: Results on 19 UD treebanks. **FO**: first order, **SO**: second order.

Figures 1, 2 and 3 show LAS by sentence length, dependency length and depth in the tree (distance to root). While BERT reduces the gap between first and second-order models, the latter are consistently more accurate in sentences longer than 10 tokens, and in dependencies longer than four tokens. Varying distance to root shows a somewhat irregular pattern (similar to what Kulmizev et al., 2019 found); the three BERT models are close to each other, but among the other three, the second-order parser is clearly best for depths 2–9.

Table 2 shows complete sentence matches and head words with exact match of their modifier set, over all treebanks. Second-order models are better on both metrics.

Table 3 shows results for models that do not employ multitask learning (in our case, jointly learning UPOS, XPOS and morphological features) on the development set for a subset of the treebanks, and the results for the models that employ it on the same data. All models are first order with a probabilistic loss function. MTL parsers performed better except for Arabic UAS, and even then only by a small difference, which motivated us to use MTL in all our experiments.

Runtime Our first-order parsers without BERT process 2,000 tokens per second on average, and the second-order ones around 600 (averaged across all treebanks). For models with BERT, the figures

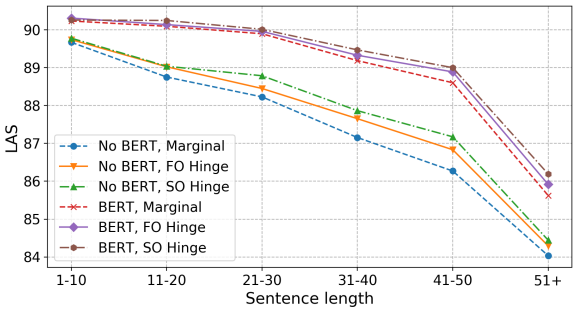

Figure 1: LAS by sentence length.

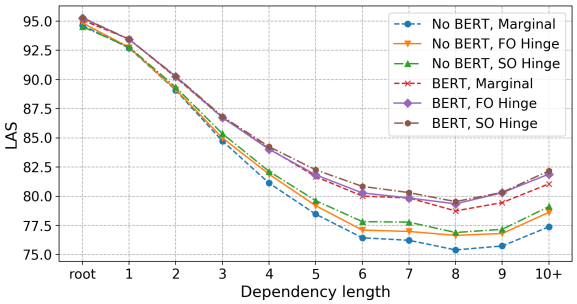

Figure 2: LAS by dependency distance.

are 1,600 and 460, respectively.[3] This slowdown of 3.5x for second-order models is even smaller than the ones reported by Martins et al. (2013).

4 Conclusion

We compared second-order dependency parsers to their more common, first-order counterparts.

[3]Runtime on an NVidia Titan Xp GPU.

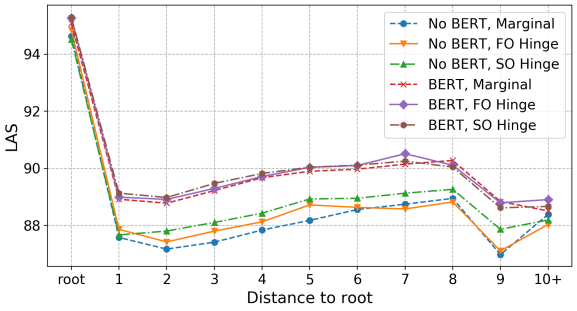

Figure 3: LAS by distance to root.

Model	Full Sent	All Mod
FO, Marginal	47.36/37.25	75.38/71.41
FO, Hinge	49.05/38.34	76.51/72.42
SO, Hinge	51.14/39.79	77.90/73.75
FO+BERT, Marg.	51.87/41.63	78.82/75.11
FO+BERT, Hinge	53.23/42.42	79.34/75.50
SO+BERT, Hinge	**54.39/42.88**	**80.14/76.13**

Table 2: Unlabeled/labeled full correct sentences and head words with full correct set of modifiers per model.

While their overall performance gain was small, they are distinctively better for longer sentences and long-range dependencies. Considering the exact match of complete parse trees or all modifiers of a word, second-order models exhibit an advantage over first-order ones. Our results indicate that even a powerful encoder as BERT can still benefit from explicit output structure modelling; this would be interesting to explore in other NLP tasks as well. Another interesting line of research would be to evaluate the contribution of higher-order features in a cross-lingual setting, leveraging structure learned from larger treebanks to underresourced languages.

Acknowledgments

This work was supported by the European Research Council (ERC StG DeepSPIN 758969), and by the Fundação para a Ciência e Tecnologia through contracts UID/EEA/50008/2019 and CMUPERI/TIC/0046/2014 (GoLocal).

References

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 957–961, Prague, Czech Republic. Association for Computational Linguistics.

	Non-MTL		MTL	
	UAS	LAS	UAS	LAS
AR	**87.23**	82.86	87.17	**82.94**
CS	92.51	90.25	**92.93**	**90.77**
EN	90.17	87.48	**90.61**	**87.97**
GRC	78.43	72.92	**79.09**	**73.72**
ZH	83.49	79.18	**83.65**	**79.70**

Table 3: Results on the development set for models with and without multitask learning (MTL; with UPOS, XPOS and morphological tagging objectives besides parsing).

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734.

Agnieszka Falenska and Jonas Kuhn. 2019. The (non-)utility of structural features in BiLSTM-based dependency parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, Prague, Czech Republic. Association for Computational Linguistics.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA. Association for Computational Linguistics.

Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. Deep contextualized word embeddings in transition-based and graph-based dependency parsing - a tale of two parsers revisited. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2755–2768, Hong Kong, China. Association for Computational Linguistics.

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics.

André Martins, Noah Smith, Mário Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

G. Papandreou and A. Yuille. 2011. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, pages 193–200, Barcelona, Spain.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium. Association for Computational Linguistics.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018a. 82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 113–123, Brussels, Belgium. Association for Computational Linguistics.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018b. An investigation of the interactions between pre-trained word embeddings, character models and POS tags in dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720, Brussels, Belgium. Association for Computational Linguistics.

Milan Straka. 2018. UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium.

Milan Straka, Jana Straková, and Jan Hajič. 2019. Evaluating contextualized embeddings on 54 languages in pos tagging, lemmatization and dependency parsing.

Zhisong Zhang, Xuezhe Ma, and Eduard Hovy. 2019. An empirical investigation of structured output modeling for graph-based neural dependency parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5592–5598, Florence, Italy. Association for Computational Linguistics.