Viable Dependency Parsing as Sequence Labeling

Michalina Strzyz **David Vilares** **Carlos Gómez-Rodríguez**
Universidade da Coruña, CITIC
FASTPARSE Lab, LyS Research Group, Departamento de Computación
Campus de Elviña, s/n, 15071 A Coruña, Spain
{michalina.strzyz,david.vilares,carlos.gomez}@udc.es

Abstract

We recast dependency parsing as a sequence labeling problem, exploring several encodings of dependency trees as labels. While dependency parsing by means of sequence labeling had been attempted in existing work, results suggested that the technique was impractical. We show instead that with a conventional BILSTM-based model it is possible to obtain fast and accurate parsers. These parsers are conceptually simple, not needing traditional parsing algorithms or auxiliary structures. However, experiments on the PTB and a sample of UD treebanks show that they provide a good speed-accuracy tradeoff, with results competitive with more complex approaches.

1 Introduction

The application of neural architectures to syntactic parsing, and especially the ability of long short-term memories (LSTMs) to obtain context-aware feature representations (Hochreiter and Schmidhuber, 1997), has made it possible to parse natural language with conceptually simpler models than before. For example, in dependency parsing, the rich feature models with dozens of features used in transition-based approaches (Zhang and Nivre, 2011) can be simplified when using feedforward neural networks (Chen and Manning, 2014), and even more with BiLSTM architectures (Kiperwasser and Goldberg, 2016), where in fact two positional features can suffice (Shi et al., 2017). Similarly, in graph-based approaches, Dozat and Manning (2017) have shown that an arc-factored model can achieve state-of-the-art accuracy, without the need for the higher-order features used in systems like (Koo and Collins, 2010).

In the same way, neural feature representations have made it possible to relax the need for structured representations. This is the case of sequence-to-sequence models that translate sentences into linearized trees, which were first applied to constituent (Vinyals et al., 2015) and later to dependency parsing (Wiseman and Rush, 2016; Zhang et al., 2017b; Li et al., 2018). Recently, Gómez-Rodríguez and Vilares (2018) have shown that sequence labeling models, where each word is associated with a label (thus simpler than sequence to sequence, where the mapping from input to output is not one to one) can learn constituent parsing.

Contribution We show that sequence labeling is useful for dependency parsing, in contrast to previous work (Spoustová and Spousta, 2010; Li et al., 2018). We explore four different encodings to represent dependency trees for a sentence of length n as a set of n labels associated with its words. We then use these representations to perform dependency parsing with an off-the-shelf sequence labeling model. The results show that we produce models with an excellent speed-accuracy tradeoff, without requiring any explicit parsing algorithm or auxiliary structure (e.g. stack or buffer). The source code is available at https://github.com/mstrise/dep2label

2 Parsing as sequence labeling

Sequence labeling is a structured prediction problem where a single output label is generated for every input token. This is the case of tasks such as PoS tagging, chunking or named-entity recognition, for which different approaches obtain accurate results (Brill, 1995; Ramshaw and Marcus, 1999; Reimers and Gurevych, 2017).

On the contrary, previous work on dependency parsing as sequence labeling is vague and reports results that are significantly lower than those provided by transition-, graph-based or sequence-to-sequence models (Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017; Zhang et al., 2017a). Spoustová and Spousta

Proceedings of NAACL-HLT 2019, pages 717–723
Minneapolis, Minnesota, June 2 - June 7, 2019.

(2010) encoded dependency trees using a relative PoS-based scheme to represent the head of a node, to then train an averaged perceptron. They did not provide comparable results, but claimed that the accuracy was between 5-10% below the state of the art in the pre-deep learning era. Recently, Li et al. (2018) used a relative positional encoding of head indexes with respect to the target token. This is used to train Bidirectional LSTM-CRF sequence-to-sequence models (Huang et al., 2015), that make use of sub-root decomposition. They compared their performance against an equivalent BiLSTM-CRF labeling model. The reported UAS for the sequence labeling model was 87.6% on the Penn Treebank, more than 8 points below the current best model (Ma et al., 2018), concluding that sequence-to-sequence models are required to obtain competitive results.

In this work, we show that these results can be clearly improved if simpler architectures are used.

3 Encoding of trees and labels

Given a sentence $w_1 \dots w_n$, we associate the words with nodes $\{0, 1, \dots, n\}$, where the extra node 0 is used as a dummy root for the sentence. A dependency parser will find a set of labeled relations encoded as edges of the form (h, d, l), where $h \in \{0, 1, \dots, n\}$ is the head, $d \in \{1, \dots, n\}$ the dependent, and l a dependency label. The resulting dependency graph must be acyclic and such that each node in $\{1, \dots, n\}$ has exactly one head, so it will be a directed tree rooted at node 0.

Thus, to encode a dependency tree, it suffices to encode the unique head position and dependency label associated with each word of $w_1 \dots w_n$. To do so, we will give each word w_i a discrete label of the form (x_i, l_i), where l_i is the dependency label and x_i encodes the position of the head in one of the following four ways (see also Figure 1):

1. Naive positional encoding: x_i directly stores the position of the head, i.e., a label (x_i, l_i) encodes an edge (x_i, i, l_i). This is the encoding used in the CoNLL file format.

2. Relative positional encoding: x_i stores the difference between the head index minus that of the dependent, i.e., (x_i, l_i) encodes an edge $(i + x_i, i, l_i)$. This was the encoding used for the sequence-to-sequence and sequence labeling models in (Li et al., 2018), as well as for the sequence-to-sequence model in (Kiperwasser and Ballesteros, 2018).

3. Relative PoS-based encoding: x_i is a tuple p_i, o_i. If $o_i > 0$, the head of w_i is the o_ith closest among the words to the right of w_i that have PoS tag p_i. If $o_i < 0$, the head of w_i is the $-o_i$th closest among the words to the left of w_i that have PoS tag p_i. For example, $(V, -2)$ means "the second verb to the left" of w_i. This scheme is closer to the notion of valency, and was used by Spoustová and Spousta (2010).

4. Bracketing-based encoding: based on (Yli-Jyrä, 2012; Yli-Jyrä and Gómez-Rodríguez, 2017). In each label (x_i, l_i), the component x_i is a string following the regular expression $(<)?((\backslash)*|(/)*)(>)?$ where the presence of character $<$ means that w_{i-1} has an incoming arc from the right, k copies of character $\backslash$ mean that w_i has k outgoing arcs towards the left, k copies of $/$ mean that w_{i-1} has k outgoing arcs towards the right, and the presence of $>$ means that w_i has an incoming arc from the left. Thus, each right dependency from a word i to j is encoded by a $(/, >)$ pair in the label components x_{i+1} and x_j, and each left dependency from j to i by a $(<, \backslash)$ pair in the label components x_{i+1} and x_j. Note that the intuition that explains why information related to a word is encoded in a neighboring node is that each x_i corresponds to a fencepost position (i.e., x_i represents the space between w_{i-1} and w_i), and the character pair associated to an arc is encoded in the most external fencepost positions covered by that arc. These pairs act as pairs of matching brackets, which can be decoded using a stack to reconstruct the dependencies.

The first three encodings can represent any dependency tree, as they encode any valid head position for each node, while the bracketing encoding only supports projective trees, as it assumes that brackets are properly nested. All the encodings are total and injective, but they are not surjective: head indexes can be out of range in the first three encodings, brackets can be unbalanced in encoding 4, and all the encodings can generate graphs with cycles. We will deal with ill-formed trees later.

4 Model

We use a standard encoder-decoder network, to show that dependency parsing as sequence label-

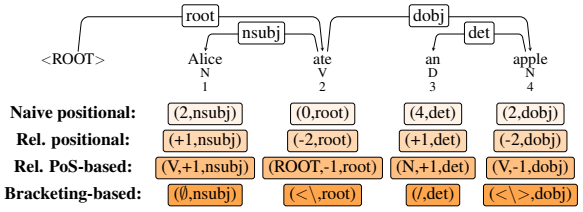

Figure 1: Types of encoding on an example tree.

ing works without the need of complex models.

Encoder We use bidirectional LSTMs (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997). Let $\text{LSTM}_\theta(\mathbf{x})$ be an abstraction of a long short-term memory network that processes the sequence of vectors $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_{|\mathbf{x}|}]$, then output for $\mathbf{x}_i$ is defined as $\mathbf{h}_i = \text{BiLSTM}_\theta(\mathbf{x}, i) = \text{LSTM}^l_\theta(\mathbf{x}_{[1:i]}) \circ \text{LSTM}^r_\theta(\mathbf{x}_{[|\mathbf{x}|:i]})$. We consider stacked BiLSTMs, where the output $\mathbf{h}^m_i$ of the mth BiLSTM layer is fed as input to the m+1th layer. Unless otherwise specified, the input token at a given time step is the concatenation of a word, PoS tag, and another word embedding learned through a character LSTM.

Decoder We use a feed-forward network, which is fed the output of the last BiLSTM. The output is computed as $P(y_i|\mathbf{h}_i) = \text{softmax}(W \cdot \mathbf{h}_i + \mathbf{b})$.

Well-formedness (i) Each token must be assigned a head (one must be the dummy root), and (ii) the graph must be acyclic. If no token is the real root (no head is the dummy root), we search for candidates by relying on the three most likely labels for each token.[1] If none is found, we assign it to the first token of the sentence. The single-head constraint is ensured by the nature of the encodings themselves, but some of the predicted head indexes might be out of bounds. If so, we attach those tokens to the real root. If a cycle exists, we do the same for the leftmost token in the cycle.

5 Experiments

We use the English Penn Treebank (PTB) (Marcus et al., 1993) and its splits for parsing. We transform it into Stanford Dependencies (De Marneffe et al., 2006) and obtain the predicted PoS tags using Stanford tagger (Toutanova et al., 2003). We also select a sample of UDv2.2 treebanks (Nivre et al., 2018): Ancient-Greek$_{\text{PROIEL}}$, Czech$_{\text{PDT}}$, Chinese$_{\text{GSD}}$, English$_{\text{EWT}}$, Finnish$_{\text{TDT}}$, Hebrew$_{\text{HTB}}$, Kazakh$_{\text{KTB}}$ and Tamil$_{\text{TTB}}$, as a representative sample, following (de Lhoneux et al., 2017). As evaluation metrics, we use Labeled (LAS) and Unlabeled Attachment Score (UAS). We measure speed in sentences/second, both on a single core of a CPU[2] and on a GPU[3].

[1] If single-rooted trees are a prerequisite, the most probable node will be selected among multiple root nodes.

Encoding	UAS	LAS
Li et al. (2018) (sequence labeling)	87.58	83.81
Li et al. (2018) (seq2seq)	89.16	84.99
Li et al. (2018) (seq2seq+beam+subroot)	93.84	**91.86**
Naive positional	45.41	42.65
Rel. positional	91.05	88.67
Rel. PoS-based	**93.99**	91.76
Bracketing-based	93.45	91.17

Table 1: Performance of our encodings on the PTB dev set with hyperparameters from Gómez-Rodríguez and Vilares (2018). We compare against previous sequence labeling and seq2seq models with more complex architectures, beam search and subroot decomposition.

Setup We use NCRFpp as our sequence labeling framework (Yang and Zhang, 2018). For PTB, we use the embeddings by Ling et al. (2015), for comparison to BIST parser (Kiperwasser and Goldberg, 2016), which uses a similar architecture, but also needs a parsing algorithm and auxiliary structures. For UD, we follow an end-to-end setup and run UDPipe[4] (Straka and Straková, 2017) for tokenization and tagging. We use the pretrained word embeddings by Ginter et al. (2017). Appendix A contains additional hyperparameters.

5.1 Encoding evaluation and model selection

We first examine the four encodings on the PTB dev set. Table 1 shows the results and also compares them against Li et al. (2018), who proposed seq2seq and sequence labeling models that use a relative positional encoding.

As the relative PoS-based encoding and bracketing-based encoding provide the best results, we will conduct the rest of our experiments with these two encodings. Furthermore, we perform a small hyperparameter search involving encoding, number of hidden layers, their dimension and presence of character embeddings, as these parameters influence speed and accuracy. From now on, we write $P^{\text{z}}_{\text{x,y}}$ for a PoS-based encoding model and $B^{\text{z}}_{\text{x,y}}$ for a bracketing-based encoding

[2] Intel Core i7-7700 CPU 4.2 GHz.

[3] GeForce GTX 1080.

[4] The pretrained models from the CoNLL18 Shared Task.

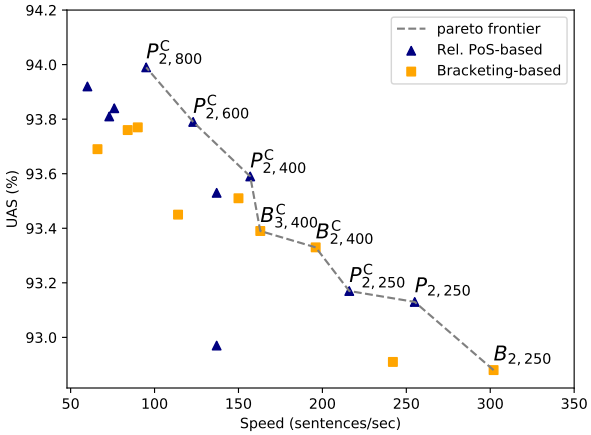

Figure 2: UAS/speed Pareto front on the PTB dev set.

model, where z indicates whether character representation was used in the model, x the number of BiLSTM layers, and y the word hidden vector dimension. We take as starting points (1) the hyperparameters used by the BIST parser (Kiperwasser and Goldberg, 2016), as it uses a BiLSTM architecture analogous to ours, with the difference that it employs a transition-based algorithm that uses a stack data structure instead of plain sequence labeling without explicit representation of structure, and (2) the best hyperparameters used by Gómez-Rodríguez and Vilares (2018) for constituent parsing as sequence labeling, as it is an analogous task for a different parsing formalism.

From there, we explore different combinations of parameters and evaluate 20 models on the PTB development set, with respect to accuracy (UAS) and speed (sentences/second on a single CPU core), obtaining the Pareto front in Figure 2. The two starting models based on previous literature ($P_{2,250}$ and $P^{C}_{2,800}$, respectively) happen to be in the Pareto front, confirming that they are reasonable hyperparameter choices also for this setting. In addition, we select two more models from the Pareto front (models $P^{C}_{2,400}$ and $B_{2,250}$) for our test set experiments on PTB, as they also provide a good balance between speed and accuracy.

5.2 Results and discussion

Table 2 compares the chosen models, on the PTB test set, against state-of-the-art models. Contrary to previous dependency-parsing-as-sequence-labeling attempts, we are competitive and provide a good speed-accuracy tradeoff. For instance, the $P^{C}_{2,800}$ model runs faster than the BIST parser (Kiperwasser and Goldberg, 2016) while being almost as accurate (-0.18 LAS). This comes in spite of its simplicity. While our BiLSTM architecture is similar to that of BIST, the sequence labeling approach does not need a stack, a specific transition system or a dynamic oracle. Using the BIST hyperparameters for our model ($P_{2,250}$) yields further increases in speed, at some cost to accuracy: 3.34x faster and -0.04 LAS score than the graph-based model, and 3.51x faster and

Model	sent/s CPU	sent/s GPU	UAS	LAS
$P_{2,250}$	$267_{\pm 1}$	$777_{\pm 24}$	92.95	90.96
$P^{C}_{2,400}$	$165_{\pm 1}$	$700_{\pm 5}$	93.34	91.34
$P^{C}_{2,800}$	$101_{\pm 2}$	$648_{\pm 20}$	**93.67**	**91.72**
$B_{2,250}$	$310_{\pm 30}$	$730_{\pm 53}$	92.64	90.59
KG (transition-based)	$76_{\pm 1}$		93.90	91.90
KG (graph-based)	$80_{\pm 0}$		93.10	91.00
CM	$654^{\diamond}$		91.80	89.60
DM		$411^{\diamond}$	95.74	94.08
Ma et al. (2018)		$10_{\pm 0}$	95.87	94.19

Table 2: Comparison of models on the PTB test set. KG refers to Kiperwasser and Goldberg (2016), CM to Chen and Manning (2014) and DM to Dozat and Manning (2017). $\diamond$ indicates the speed is taken from their paper.

	UPoS-based			XPoS-based		
Treebank	UAS	LAS	# UPoS	UAS	LAS	# XPoS
Ancient Greek	76.58	71.70	14	**77.00**	**72.14**	23
Chinese	**61.01**	**57.28**	15	60.98	57.14	42
Czech	**89.82**	**87.63**	17	88.33	85.46	1417
English	**82.22**	**78.96**	17	82.05	78.70	50
Finnish	**80.31**	**76.39**	15	80.19	76.28	12
Hebrew[5]	67.23	62.86	17	67.23	62.86	17
Kazakh[6]	32.14	17.03	15	**32.93**	**17.07**	26
Tamil	**73.24**	**66.51**	13	59.70	52.57	210

Table 3: Performance of the $P^{C}_{2,800}$ model with UPoS- and XPoS-based encoding for each language on the dev set. # UPoS/XPoS represents the number of distinct UPoS/XPoS tags in the training set for each language.

Treebank	PoS type	$P^{C}_{2,800}$ (sent/s) CPU	UAS	LAS	KG (transition-based) (sent/s) CPU	UAS	LAS
Ancient Greek	XPOS	$123_{\pm 1}$	75.31	70.87	$116_{\pm 4}$	69.43	64.41
Chinese	UPOS	$105_{\pm 0}$	63.20	59.12	$73_{\pm 1}$	64.69	60.45
Czech	UPOS	$125_{\pm 1}$	89.10	86.68	$94_{\pm 3}$	89.25	86.11
English	UPOS	$139_{\pm 1}$	81.48	78.64	$120_{\pm 2}$	82.22	79.00
Finnish	UPOS	$168_{\pm 0}$	80.12	76.22	$127_{\pm 3}$	80.99	76.63
Hebrew	equal PoS	$120_{\pm 0}$	63.04	58.66	$70_{\pm 1}$	63.56	58.80
Kazakh	XPOS	$283_{\pm 3}$	32.93	17.07	$178_{\pm 5}$	23.09	12.73
Tamil[7]	UPOS	$150_{\pm 2}$	71.59	64.00	$127_{\pm 3}$	75.41	68.58

Table 4: Comparison on UD-CoNLL18 test sets.

[5] In Hebrew, UPoS and XPoS tags are the same.

[6] Kazakh is missing a development set. The scores are based on the test set.

[7] Tamil was run on gold segmented and tokenized inputs, as there is no pretrained UDpipe model. We did not use pretrained word embeddings either.

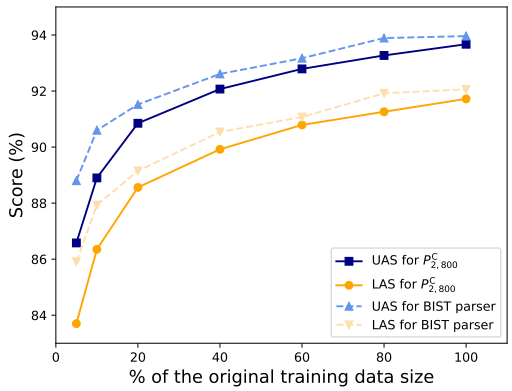

Figure 3: Impact of the PTB data size available for parsers during training on the results from the test set.

-0.94 LAS score than their transition-based one.

We now extend our experiments to the sample of UD-CoNLL18 treebanks. To this end, we focus on the $P^C_{2,800}$ model and since our PoS tag-based encoding can be influenced by the specific PoS tags used, we first conduct an experiment on the development sets to determine what tag set (UPoS, the universal PoS tag set, common to all languages, or XPoS, extended language-specific PoS tags) produces the best results for each dataset.

Table 3 shows how the number of unique UPoS and XPoS tags found in the training set differs in various languages. The results suggest that the performance of our system can be influenced by the size of the tag set. It appears that a very large tag set (for instance the XPoS tag set for Czech and Tamil) can hurt the performance of the model and significantly slow down the system, as it results into a large number of distinct labels for the sequence labeling model, increasing sparsity and making the classification harder. In case of Ancient Greek and Kazakh, the best performance is achieved with the XPoS-based encoding. In these corpora, the tag set is slightly bigger than the UPoS tag set. One can argue that the XPoS tags in this case were possibly more fine-grained and hence provided additional useful information to the system facilitating a correct label prediction, without being so large as to produce excessive sparsity.

Table 4 shows experiments on the UD test sets, with the chosen PoS tag set for each corpus. $P^C_{2,800}$ outperforms transition-based BIST in LAS in 3 out of 8 treebanks,[8] and is clearly faster in all analyzed languages. We believe that the variations between languages in terms of LAS difference with respect to BIST can be largely due to differences in the accuracy and granularity of predicted PoS tags, since our chosen encoding relies on them to encode arcs. The bracketing-based encoding, which does not use PoS tags, may be more robust to this. On the other hand, finding the optimal granularity of PoS tags for the PoS-based encoding can be an interesting avenue for future work.

In this work, we have also examined the impact of the training data size on the performance of our system compared to the performance of BIST parser. The results in Figure 3 suggest that our model requires more data during the training than BIST parser in order to achieve similar performance. The performance is slightly worse when little training data is available, but later on our model reduces the gap when increasing the training data size.

6 Conclusion

This paper has explored fast and accurate dependency parsing as sequence labeling. We tested four different encodings, training a standard BiLSTM-based architecture. In contrast to previous work, our results on the PTB and a subset of UD treebanks show that this paradigm can obtain competitive results, despite not using any parsing algorithm nor external structures to parse sentences.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the TELEPARES-UDC project (FFI2014-51978-C2-2-R) and the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia (ED431B 2017/01). We gratefully acknowledge NVIDIA Corporation for the donation of a GTX Titan X GPU.

[8]For Ancient Greek, this may be related to the large amount of non-projectivity (BIST is a projective parser). For extra comparison, a non-projective variant of BIST (Smith et al., 2018) obtains 71.58 LAS with mono-treebank training, but from better segmentation and morphology than used here. UDpipe (Straka and Straková, 2017) obtains 67.57 LAS. Czech and Kazakh have a medium amount of non-projectivity.

References

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational linguistics*, 21(4):543–565.

Danqi Chen and Christopher D. Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Lrec*, volume 6, pages 449–454.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of the 5th International Conference on Learning Representations*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343. Association for Computational Linguistics.

Filip Ginter, Jan Hajič, Juhani Luotolahti, Milan Straka, and Daniel Zeman. 2017. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Eliyahu Kiperwasser and Miguel Ballesteros. 2018. Scheduled multi-task learning: From syntax to translation. *Transactions of the Association for Computational Linguistics*, 6:225–240.

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.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *TLT*, pages 99–110.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1299–1304. Association for Computational Linguistics.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414. Association for Computational Linguistics.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational linguistics*, 19(2):313–330.

Joakim Nivre et al. 2018. Universal dependencies 2.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Lance A Ramshaw and Mitchell P Marcus. 1999. Text chunking using transformation-based learning. In *Natural language processing using very large corpora*, pages 157–176. Springer.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348. Association for Computational Linguistics.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Tianze Shi, Liang Huang, and Lillian Lee. 2017. Fast(er) exact decoding and global training for transition-based dependency parsing via a minimal feature set. In *Proceedings of the 2017 Conference*

on Empirical Methods in Natural Language Processing, pages 12–23, Copenhagen, Denmark. Association for Computational Linguistics.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018. 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. Association for Computational Linguistics.

Drahomíra Spoustová and Miroslav Spousta. 2010. Dependency parsing as a sequence labeling task. *The Prague Bulletin of Mathematical Linguistics*, 94(1):7–14.

Milan Straka and Jana Straková. 2017. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas. Association for Computational Linguistics.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia. Association for Computational Linguistics.

Anssi Yli-Jyrä. 2012. *On Dependency Analysis via Contractions and Weighted FSTs*, pages 133–158. Springer Berlin Heidelberg, Berlin, Heidelberg.

Anssi Yli-Jyrä and Carlos Gómez-Rodríguez. 2017. Generic axiomatization of families of noncrossing graphs in dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1745–1755, Vancouver, Canada. Association for Computational Linguistics.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017a. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 665–676.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017b. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark. Association for Computational Linguistics.

A Model parameters

During the training we use Stochastic Gradient Descent (SGD) optimizer with a batch size of 8, and the model is trained for up to 100 iterations. We keep the model that obtains the highest UAS on the development set. Additional hyperparameters are shown in Table 5.

Word embedding dimension	100
Char embedding dimension	30
PoS tag embedding dimension	25
Word hidden vector dimension	250,400,600, 800,1000,1200
Character hidden vector dimension	50
Initial learning rate	0.02
Time-based learning rate decay	0.05
Momentum	0.9
Dropout	0.5

Table 5: Common hyperparameters for the sequence labeling models.