Old School vs. New School: Comparing Transition-Based Parsers with and without Neural Network Enhancement

Miryam de Lhoneux, Sara Stymne and Joakim Nivre

Department of Linguistics and Philology
Uppsala University
E-mail: {miryam.de_lhoneux,sara.stymne,joakim.nivre}@lingfil.uu.se

Abstract

In this paper, we attempt a comparison between "new school" transition-based parsers that use neural networks and their classical "old school" counterpart. We carry out experiments on treebanks from the Universal Dependencies project. To facilitate the comparison and analysis of results, we only work on a subset of those treebanks. However, we carefully select this subset in the hope to have results that are representative for the whole set of treebanks. We select two parsers that are hopefully representative of the two schools; MaltParser and UDPipe and we look at the impact of training size on the two models. We hypothesize that neural network enhanced models have a steeper learning curve with increased training size. We observe, however, that, contrary to expectations, neural network enhanced models need only a small amount of training data to outperform the classical models but the learning curves of both models increase at a similar pace after that. We carry out an error analysis on the development sets parsed by the two systems and observe that overall MaltParser suffers more than UDPipe from longer dependencies. We observe that MaltParser is only marginally better than UDPipe on a restricted set of short dependencies.

1 Introduction

Treebanks have recently been released for a large number of languages in a consistent annotation within the framework of the Universal Dependencies (UD) project [14]. In line with work on dependency parsing following the CoNLL shared task from 2006 [3], and in contrast with most work on constituency parsing which has focused on one language and one domain (the English Penn Treebank), this project may help reshape the field of syntactic parsing by using a wide variety of languages and domains. Simultaneously to the development of this project, syntactic parsing has seen a significant boost in accuracy in the last couple of years with methods

that make use of neural networks to learn dense vectors for words, POS tags and dependency relations [4, 18, 1] or stacks [5].

Motivated by the success of neural network models on the PTB and the Chinese treebank, Straka et al. [17] trained Parsito, a neural network model, on UD treebanks and obtained good results, improving over their 'classical' counterpart MaltParser [15]. There was, however, no systematic comparison between the classical and the neural network approach and it may be that one approach is more suitable than another for specific settings. Moreover, the results they reported for MaltParser are obtained using default settings but results can be much higher with optimised settings.

In this paper, we propose to compare the performance of these two types of parsers on UD treebanks. We first propose to select a sample of the UD treebanks to ease the comparison between parsing models in general.

2 Treebank Sampling for Comparative Parser Evaluation

Having a large and varied data set has the advantage that our parsing models will generalize better. A disadvantage is that it is expensive to train models for all the languages, especially with the new neural network models that need a search over a large hyperparameter space in order to be optimised. As UD grows, it may become more and more prohibitive to train models for all the languages when we want to evaluate how a parser does as opposed to another or as opposed to a modified version of itself.

We therefore suggest that it might be wise to examine their behavior in a small-scaled setting before training them for the large number of treebanks in UD. Parsing models can first be evaluated on a small sample of UD treebanks. Subsequently, depending on the observations on the small set, we can move to a medium sample before finally testing on all the treebanks if evidence points towards a clear direction. We have come up with a set of criteria to select the small sample which we now turn to.

The objective was to have a sample as representative of the whole treebank set as possible. To ensure typological variety we divided UD languages into coarse-grained and fine-grained language families. This led to a total of 15 different fine-grained families and 8 coarse-grained. We made it a requirement to not select two languages from the same fine-grained family and ensured to have some variety in coarse-grained families. We made sure to have at least one isolating, one morphologically-rich and one inflecting language. We additionally ensured a variability of treebank sizes and domains. Since parsing non-projective trees is notoriously harder than parsing projective trees, we also made sure to have at least one treebank with a large amount of non-projective trees. The quality of treebanks was also considered in the selection, in particular, there are known issues[1] about inconsistency in the annotation. We selected languages that had as little of those

[1] http://universaldependencies.org/svalidation.html

Table 1: Treebank Sample

	coarse	fine	main argument for inclusion
Czech	indo-european	balto-slavic	the largest UD treebank
Chinese	sino-tibetan	sinitic	the only isolating language (without copyrights)
Finnish	uralo-altaic	finno-ungric	morphologically rich; has many different domains
English	indo-european	germanic	largest treebank with full manual check of the data
Ancient_Greek-PROIEL	indo-european	hellenic	large percentage of non-projective trees
Kazakh	uralo-altaic	turkic	smallest treebank; full manual check of the annotations
Tamil	dravidian	tamil	small treebank; language family considerations
Hebrew	afro-asiatic	semitic	morphologically rich/language family considerations

as possible. To ensure comparability, we finally made sure to select only treebanks with morphological features (with one exception for Kazakh). This resulted in a selection of 8 treebanks. The selection is given in Table 1 together with main arguments for inclusion for each.

3 Comparing Parsing Accuracy

As said in the introduction, a systematic comparison of classical models for transition-based parsing as opposed to models helped by neural network training (henceforth NN parsers) is lacking. With our selection of treebanks just presented, we will now compare those two model types. More specifically, we compare MaltParser with Parsito. As was also said, when compared with NN parsers, MaltParser results were reported using default settings but results can be much better with optimised settings. For this reason, we optimised models with the help of MaltOptimizer [2]. In an attempt to keep the models somewhat similar across languages, we chose to use the same parsing algorithm for all the languages and hence only optimised options specific to the data set (like the root label for example) (MaltOptimizer's phase 1) and the feature model (MaltOptimizer's phase 3). We selected the arc-standard swap system with a lazy oracle for its ability to deal with non-projectivity which was crucial at least for Ancient Greek. Additionally, it was one of the popular algorithms suggested by MaltOptimizer for our selection of languages, together with its projective version.

For Parsito, we used the pretrained models that the authors made available. They are trained on UD version 1.2 but we tested them on version 1.3 since that is the version used for the other parsers. We additionally optimised models for the languages for which there was no pretrained model available as well as for Tamil because the results were too low on version 1.3 as compared to version 1.2 (probably due to significant differences in the 2 versions). In order to optimise those models, we first made sure that we could reproduce results from the pretrained models on one language (Hebrew): we experimented with the transition system and oracle and used the random hyperparameter search provided by UDPipe to tune the hyperparameters.

We add the best reported results for SyntaxNet [1] because they are currently on

Table 2: Parsing models comparison (LAS, excluding punctuation).

language	tokens	UDPipe	Malt	*SyntaxNet*
Ancient_Greek-PROIEL	206K	**68.3**	67.8	*73.15*
Chinese	123K	**68.9**	67.7	*71.24*
Czech	1,503K	**82.6**	80.3	*85.93*
English	254K	**81.3**	79.9	*80.38*
Finnish	181K	**75.7**	74.2	*79.60*
Hebrew	115K	**77.2**	76.2	*78.71*
Kazakh	4K	41.2	**44.4**	*43.95*
Tamil	8K	58.1	**58.8**	*55.35*

average the best reported results for UD. Parsito and SyntaxNet are both transition-based parsers that use neural networks to learn vector representations of words, POS tags and dependency relations in a similar way to Chen and Manning [4]. Parsito adds a search-based oracle and SyntaxNet adds beam search and global normalization.

Parsito has been integrated into the recent UDPipe [16] parsing pipeline that performs tokenisation, morphological analysis and POS tagging. Beam search was added to Parsito in the version used in UDPipe. We will refer to that parser as the UDPipe parser in the remainder of this paper. UDPipe taggers and morphological analysers were trained for all treebanks and both MaltParser and the UDPipe parser were tested on the test sets that used those models to predict POS tags (universal and language specific) as well as morphological features. Note that SyntaxNet results are not directly comparable as they use their own POS tagger and morphological analyser.

Labeled Attachment Scores (LAS) are given in Table 2. As appears from the table, UDPipe largely outperforms MaltParser. However, MaltParser performs better than UDPipe on very small data sets. SyntaxNet is even better than UDPipe in most cases but is also outperformed by MaltParser on very small treebanks.

4 Impact of Training Size on Neural Network Parsing

Looking at these results, we can hypothesize that the superior performance of the NN parsers depend on having reasonably large training sets. We hypothesize that neural network parsers improve more steeply with increased training size than MaltParser does. We tested this hypothesis with a learning curve experiment. We trained several parsers for our selection of languages, varying the size of the training data. In order to prevent sequential effects that may come from the structure of treebanks from impacting the results, we randomly shuffled the sentences of the training set before splitting it to different sample sizes. We compared the effect of doing that using MaltParser and UDPipe. We first split the training data sets into one sample of 1K and samples of 50K. Additionally, to zoom in on what hap-

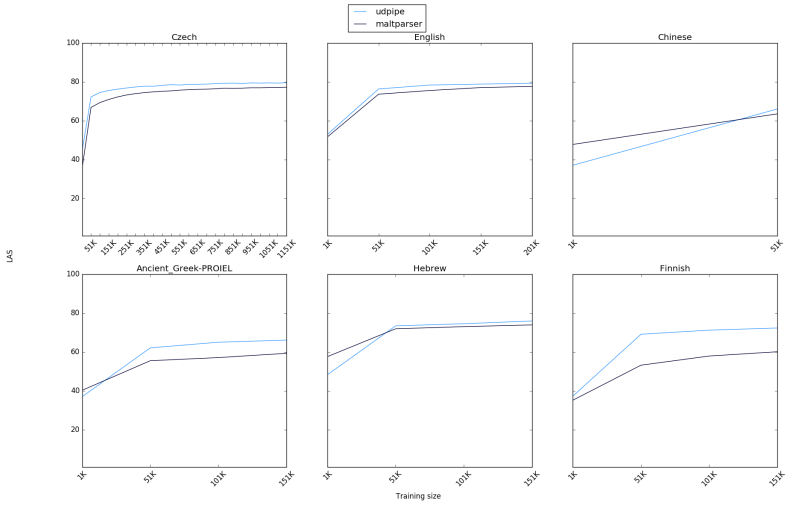

Figure 1: Learning curve experiment

pens with very small data sizes, we also ran the experiment with splits of 1K from 1K to 15K. Because it would have been unreasonable to optimise models for each split size, we used unoptimised version of both MaltParser and UDPipe. For MaltParser, we used the arc-standard swap algorithm again with a lazy oracle and an extended feature model that uses morphological features, in the same way as was done by Nivre [13]. The use of morphological features is important so as to have a model that is comparable to UDPipe but also because morphological features are crucial for parsing morphologically rich languages like Finnish and Hebrew. For UDPipe, we also used the swap transition system as well as a lazy oracle so that we have a comparable system for both parsers. We used UDPipe's default hyperparameters which they report to be the hyperparameters that worked best across their experiments on UD treebanks[2].

As can be seen in Figures 1 and 2, the hypothesis that neural network parsers increase more steeply than MaltParser with increased training size seems to hold only to some extent. The learning curve for UDPipe is very steep for only the first few thousands of tokens but flattens out quite quickly after that and continues improving at a similar pace as MaltParser. It is interesting to note also that MaltParser does not even outperform UDPipe on all treebanks with a training size of 1K.

[2] http://ufal.mff.cuni.cz/udpipe/users-manual

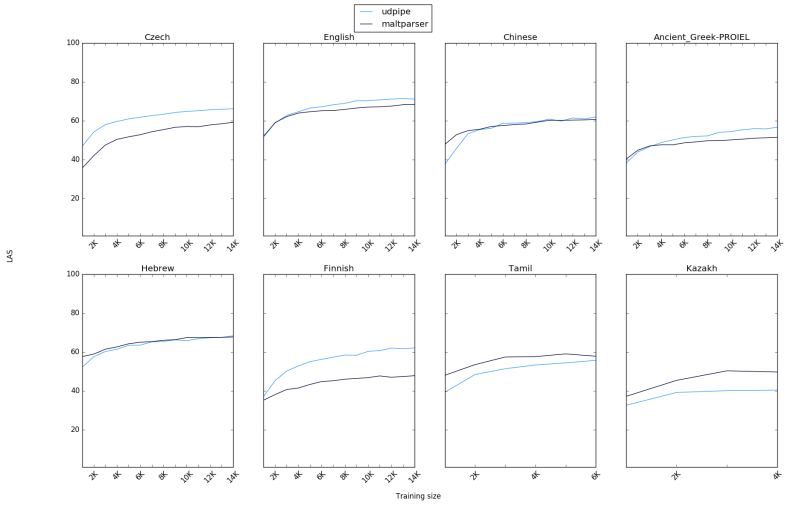

Figure 2: Learning curve experiment: zoom on tiny training sizes

5 Error Analysis

5.1 Error Analysis Approaches

Since training size does not seem to be the only factor explaining the variation in results in the comparison of the parsers, we want to gain more insights into the strengths and weaknesses of each parser.

There are many ways of comparing different parsers which, as described in Kirilin and Versley [7], can be placed on a coarse-grained to fine-grained scale where the coarsest level just consists in comparing the attachment scores and the finest level consists in manually looking at output parse trees of both systems. There are many other possibilities in between these two levels. McDonald and Nivre [11] look at the impact of different properties of trees on accuracy of a transition-based and a graph-based parser. Goldberg and Elhadad [6] follow up on this and look more in detail at the over- and underproductions of specific constructions of those two systems, showing that it is possible to train a classifier that predicts which system was used to parse some output data. Kummerfeld et al. [8] also look at some fine-grained phenomena such as PP and NP attachment and compare the behaviour of many constituency parsers on those phenomena. Kirilin and Versley [7] also look at fine-grained error patterns made by different parsers on UD treebanks.

There has not been much work on characterizing the errors of neural network parsers, at least the feedforward neural network ones. Nguyen et al. [12] compared the performance of two graph-based and two transition-based parsers, each pair of which contains a neural network parser, on the Vietnamese treebank. The parsers they used, however, use Recurrent Neural Network (RNN) models. Such a study has not been done for feedforward neural network parsers, as far as we are aware.

Similarly, recent work has started investigating what neural network parsers learn [10, 9], but these again use RNNs. For this reason, the approach by McDonald and Nivre [11] seems particularly suited here to get an overview of the strengths and weaknesses of feedforward neural network parsers compared to their classical counterpart. It would be interesting to follow up on this study by looking at more fine-grained phenomena.

5.2 MaltParser vs UDPipe

As just mentioned, McDonald and Nivre [11] conducted an extensive error analysis on two parsers, MaltParer and MSTParser, coming from two different parsing frameworks: the transition-based and the graph-based parsing framework respectively. They analyse the effect of different properties of dependency trees on accuracy which they divide into two different types: graph and linguistics factors. For the first type, they look at the length of dependencies, length of the sentence, and a few other things. For the second type, they divide accuracy across different POS tags and dependency relations. This allows them to observe a tradeoff between the rich representation of MaltParser that allows it to do well on frequent dependencies and the problem of error propagation from which MaltParser suffers more than MSTParser. They argue that these results can be explained by the properties of the two parsers and the tradeoff between rich representation combined with local greedy inference and less rich representation combined with global exact inference. We attempted to carry out a similar study to compare the two parsers that we are investigating in this paper.

McDonald and Nivre [11] worked on the CoNLL shared-task data [3], that is, data from 13 different languages. They concatenated the test sets parsed by the two systems they compared and conducted their analysis on that concatenated test set. They were able to do that because the test sets all had a very similar size. In this study, we are working on the development sets. We could also have concatenated those development sets but their sizes vary by orders of magnitude ($\sim$500 tokens to $\sim$150K tokens) so we unfortunately cannot do this. We could theoretically create a balanced test set using only as many tokens as there are in the smallest development set. The 2 smallest development sets are, however, very small ($\sim$500 and $\sim$1200 tokens), which means that doing this would make the data set very small and sparse. In order to reach a compromise between a perfectly balanced small data set and a very unbalanced big data set, we concatenated a portion of each treebank of the size of the smallest data size of the development sets excluding Tamil and Kazakh (which is Finnish and has $\sim$9K tokens). We added the full development set of Tamil and Kazakh but we keep in mind the fact that they have a small impact on the results.

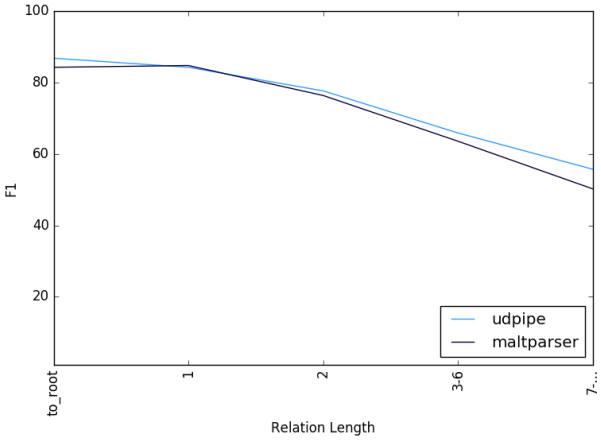

Figure 3: Accuracy relative to dependency Length

Graph Factors

Our overall results are not as clear-cut as they were in McDonald and Nivre [11] but we observe somewhat similar tendencies: the advantages of MSTParser over MaltParser seem to carry over to UDPipe to some extent. For example, as can be seen in Figure 3, MaltParser seems to suffer from dependency length more than UDPipe does. Note that we use the F1 harmonic mean of precision and recall here instead of LAS. This is because we cannot assume a one-to-one correspondence between the predicted and gold dependencies.

The picture is not so clear for sentence length as can be seen from Figure 4 because although accuracy for MaltParser seems to be decreasing more than for UDPipe between 1-10 and 20-30 token sentences, they both perform similarly on sentences between 40 and 50 tokens and UDPipe is then again better on sentences longer than 50 tokens. Note however that the max length of sentences for Kazakh and Tamil are 27 and 49 respectively which might provide some explanation of why MaltParser outperforms UDPipe for those.

It is interesting to point out that Nguyen et al. [12] reported a similar tendency between the results of RNN graph-based and transition-based parsers and their classical counterpart, where the RNN ones suffer less from sentence and dependency length than the classical ones.

Linguistic Factors

McDonald and Nivre [11] created a taxonomy of POS tags and dependency relations so as to group similar ones together and have consistent labels across treebanks. Luckily, in our case, we already have consistent dependency labels and POS tags. In Figure 5 we give accuracies (in terms of F1 again for the same reason as before) for the 15 most frequent dependency relations and in Figure 6 we give

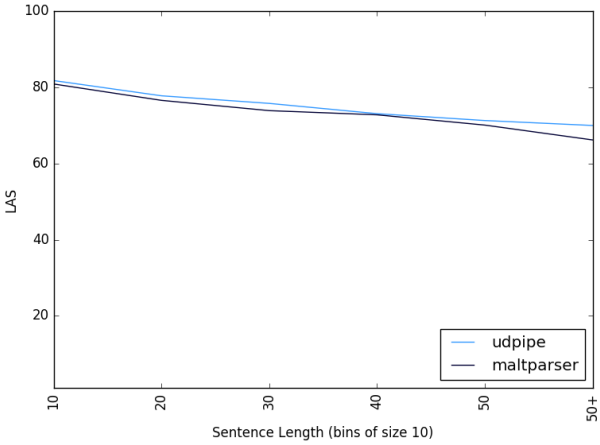

Figure 4: Accuracy relative to sentence Length

accuracies for the POS tags. The picture seems somewhat consistent with what we have observed so far: UDPipe is better than MaltParser on dependencies that may be distant such as *conj* and *advcl*. MaltParser is better, although not by far, for dependencies that are expected to be short such as *nummod*.

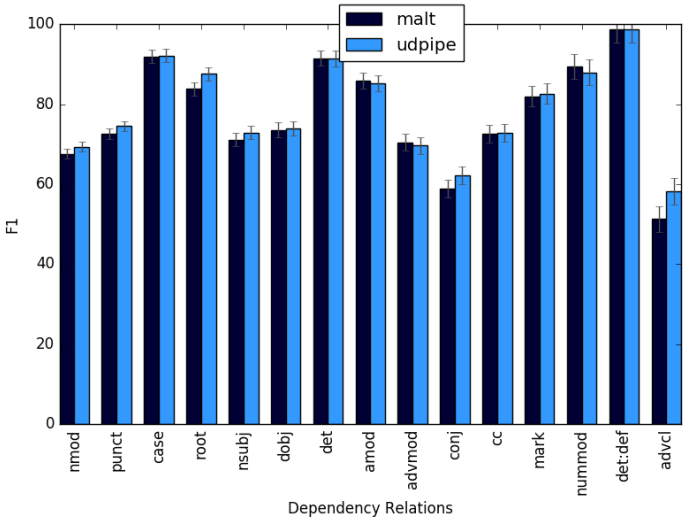

Figure 5: Accuracy for different dependency relations

The accuracies for POS tags show a similar picture as dependency relations. UDPipe outperforms MaltParser on most POS tags except the ones that are expected to be close to their head such as *NUM*.

Overall then, UDPipe outperforms MaltParser on most dependencies and tags which shows that in general, their representation is better. MaltParser outperforms

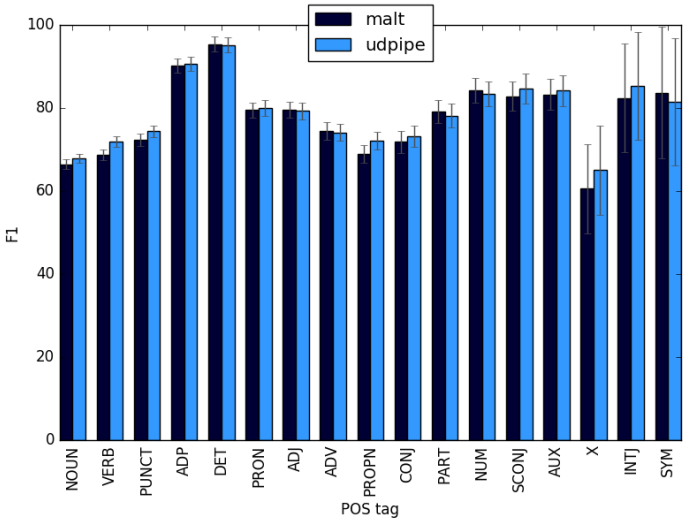

Figure 6: Accuracy for different POS tags

UDPipe only by a small margin on a limited number of phenomena which all seem to involve short dependencies.

It is important to note however that an important factor at play here is the beam search used by UDPipe. It has been shown that beam search helps with long dependencies [19]. It would be interesting to isolate the beam search factor from the neural network classifier one by comparing UDPipe with and without it.

6 Conclusion and Future Work

In this paper, we presented a comparison of the performance of neural network parsers with their classical counterpart. We saw that on small treebanks, MaltParser outperforms UDPipe. We investigated the effect of training size on both types of parsers and observed that UDPipe is better than MaltParser even on tiny data sizes. We observed that UDPipe has a steep learning curve with very small data sizes but flattens out to a smaller increase rate in a similar way as MaltParser which stays only a few percentages below with increased training sizes in most cases. We carried out an error analysis and observed that MaltParser suffers more from increased dependency length than UDPipe does. MaltParser only does marginally better on a restricted set of short dependencies. Doing more fine-grained error analysis could lead to a more in-depth understanding of the strengths and weaknesses of the two models and it would be interesting to investigate the effect of beam search on neural network models.

References

[1] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

[2] Miguel Ballesteros and Joakim Nivre. 2016. MaltOptimizer: Fast and effective parser optimization. *Natural Language Engineering* 22(2):187–213.

[3] Sabine Buchholz and Erwin Marsi. 2006. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pages 149–164.

[4] Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Empirical Methods in Natural Language Processing (EMNLP)*.

[5] 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)*. Association for Computational Linguistics, Beijing, China, pages 334–343.

[6] Yoav Goldberg and Michael Elhadad. 2010. Inspecting the structural biases of dependency parsing algorithms. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, pages 234–242.

[7] Angelika Kirilin and Yannick Versley. 2015. What is hard in Universal Dependency Parsing? In *6th Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2015)*. pages 31–38.

[8] Jonathan K Kummerfeld, David Hall, James R Curran, and Dan Klein. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, pages 1048–1059.

[9] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. 2016. What do recurrent neural network grammars learn about syntax? *arXiv preprint arXiv:1611.05774* .

[10] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *arXiv preprint arXiv:1611.01368* .

[11] Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. pages 122–131.

[12] Dat Quoc Nguyen, Mark Dras, and Mark Johnson. 2016. An empirical study for vietnamese dependency parsing. In *Proceedings of the Australasian Language Technology Association Workshop 2016*. Melbourne, Australia, pages 143–149.

[13] Joakim Nivre. 2016. Universal dependency evaluation. *Unpublished paper* .

[14] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. 2016. Universal dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*.

[15] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryiğit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering* 13(2):95–135.

[16] Milan Straka, Jan Hajič, and Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. European Language Resources Association (ELRA), Paris, France.

[17] Milan Straka, Jan Hajič, Jana Straková, and Jan Hajič jr. 2015. Parsing universal dependency treebanks using neural networks and search-based oracle. In *Proceedings of Fourteenth International Workshop on Treebanks and Linguistic Theories (TLT 14)*.

[18] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. 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)*. Association for Computational Linguistics, Beijing, China, pages 323–333.

[19] Yue Zhang and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *COLING (Posters)*. pages 1391–1400.