Linear Neural Parsing and Hybrid Enhancement for Enhanced Universal Dependencies

Giuseppe Attardi, Daniele Sartiano, Maria Simi
Department of Computer Science,
University of Pisa
{attardi, sartiano, simi}@di.unipi.it

Abstract

To accomplish the shared task on dependency parsing we explore the use of a linear transition-based neural dependency parser as well as a combination of three of them by means of a linear tree combination algorithm. We train separate models for each language on the shared task data. We compare our base parser with two biaffine parsers and also present an ensemble combination of all five parsers, which achieves an average UAS 1.88 point lower than the top official submission. For producing the enhanced dependencies, we exploit a hybrid approach, coupling an algorithmic graph transformation of the dependency tree with predictions made by a multitask machine learning model.

1 System Overview

The shared task is aimed at performing all the levels of linguistic analysis according to the UD guidelines, starting from raw text all the way to enhanced dependency graphs. All this in a multi-language setting for seventeen languages (Bouma et al., 2020).

In this endeavor, we concentrate on the syntactic parsing and enhancement stages, by exploiting existing tools for tokenization, sentence splitting, POS tagging and morphological analysis.

For syntactic parsing we make experiments exploring different ideas, in an attempt to improve state-of-the-art parsers with linear complexity. A parser combination is then used for our official submission, exploiting the linear tree combination algorithm by Attardi and Dell'Orletta (2009), resulting in an overall linear algorithm.

For the enhancement step, we build on previous work in writing an enhancer for UD, based on algorithmic graph transformation, that was used to produce the Italian version of the enhanced dependencies (Simi and Montemagni, 2018). The script used language specific heuristics and lexical information, achieving a good degree of accuracy for Italian and English. In this multi-language challenge, we have to deal with partial implementations of the expected enhancement types as well as with varying degree of compliance with the guidelines in the different languages. In order to address this additional complexity, we implement a new version of the script for making it modular, parametric, and language independent. For specific enhancement tasks, we integrate the output of machine learning classifiers, in an attempt to learn from the training data and make the heuristics more robust and general.

2 Syntactic parsing

State of the art dependency parsers currently often adopt the graph-based model, based on neural networks for the choice of arcs and labels.

We consider as current SoTA on the English PTB the graph dependency parsers listed in Table 1.

In particular the Bi-LSTM-based deep biaffine neural dependency parser by Dozat and Manning (2017) has been quite popular and used in three out of five of the top submissions to the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies (Zeman et al., 2018), in particular in the top non-ensemble submission (Kanerva et al., 2018).

The preference for such models leads to systems with high accuracy but possibly slower due to their $O(n^2)$ complexity. For example, the original implementation of the Dozat parser is rated at about 400 sents/sec on GPUs, while for example the neural transition-based parser by Chen and Manning (2014) is rated at 640 sents/sec just on CPUs. Our experiments attempt to find a parser with linear complexity and hence good speed performance. Indeed the linear transition parser that we choose for our experiments (UUParser) is twice as fast as the

Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task, pages 206–214
Virtual Meeting, July 9, 2020.

Parser	UAS	LAS
HPSG (Zhou and Zhao, 2019)	**96.09**	**94.68**
BIST-Graph (Kiperwasser and Goldberg, 2016)	93.10	91.00
Biaffine (Dozat and Manning, 2017)	95.74	94.08
Pointer-TD (Ma et al., 2018)	95.87	94.19
Pointer-LR (Fernández-González and Gómez-Rodríguez, 2019)	**96.04**	**94.43**
UUParser (de Lhoneux et al., 2017)	94.63	92.77
BIST-Transition (Kiperwasser and Goldberg, 2016)	93.9	91.9
CM (Chen and Manning, 2014)	91.80	89.60

Table 1: SoTA dependency parsers, grouped into graph-based (top) and transition-based (bottom).

latest version of the biaffine parser from Stanford (Stanza). However, after submission, we discovered a new implementation of the biaffine parser in PyTorch (Zhang, 2019), which is 5 times faster by better exploiting GPU acceleration.

We trained our own models for each language on the shared task treebanks for UUParser, UDPipe and Zysite, while we used a pretrained multilanguage model for UDify and pretrained individual language models for Stanza.

2.1 UUParser

We choose `UUParser` as our base parser. UUParser (de Lhoneux et al., 2017) is a transition-based parser model, derived from the parser by (Kiperwasser and Goldberg, 2016): the (bidirectional) LSTM's recurrent output vector for each word is concatenated with each possible head's recurrent vector, and the result is used as input to a MLP that scores each resulting arc. The predicted tree structure at training time is the one where each word depends on its highest-scoring head. Labels are generated analogously, with each word's recurrent output vector and its gold or predicted head word's recurrent vector being used in a multi-class MLP. We ported the Kiperwasser parser to Python 3. UUParser was further extended to deal with non-projectivity by means of a swap transition and to support ELMo embeddings as an input to the LSTM.

We further extended UUParser in order to exploit BERT and AlBERT embeddings. Words are first tokenized with their specific tokenizer and then the embeddings for words split into wordpieces obtained as the average of the wordpiece embeddings.

The code for the extended version is available on GitHub[1].

On development experiments, using the English and Italian train and development sets provided for the task, we obtained the results in Table 2.

Treebank	Embeddings	UAS	LAS
en-ewt	BERT	91.24	89.33
en-ewt	AlBERT	91.36	89.39
fr-sequoia	BERT	91.44	89.55
cs-pdt	BERT	93.87	91.94
it-isdt	BERT	94.67	93.11

Table 2: Parser accuracy on the development set.

For BERT we use the base-uncased model and for AlBERT the large-v2 model, which we keep frozen during training. Given the minor difference between using BERT and AlBERT, in our experiments we choose to use the BERT model.

We explored the idea to provide hints to the parser, obtained from structural syntax probes (Hewitt and Mannings, 2019). We use a syntax probe to estimate the parse tree path distance between two tokens. The transition-based parser needs to decide at each step which transition to apply to the pair of words on the top of the stack (s0) and on the input buffer (b0). The parser computes a distance matrix for each pair of tokens in a sentence. The parser is provided as additional features the estimated distances between b0 and the top k (default 3) tokens on the stack. These distances should help the parser in deciding whether to perform a Shift transition rather than a premature Reduce.

The results we obtained with such an extension on the English development corpus where 92.21 UAS and 90.31 LAS, using ELMo embeddings for word representations and BERT for syntax probes, a small improvement with respect to 91.32 UAS and 89.33 LAS without using these features.

We also tested two biaffine parsers: the implementation by `Zysite` (Zhang, 2019) and `Stanza` (Qi et al., 2020) which augments the bi-

[1] https://github.com/attardi/uuparser

Parser	GPU sents/s	CPU sents/s	UAS
UUParser	16.62	0.81	83.93
Stanza	7.77	0.43	84.51
Zysite	**84.82**	**2.11**	**86.67**

Table 3: Speed performance of parsers: average user time on all test set.

affine parser with features to predict the *linearization* order of two words in a given language, and to predict the typical distance in linear order between them.

We report in table 3 the average speed performance on all the 17 test sets of the challenge obtained by the linear parser and the two quadratic biaffine graph parsers.

The `Zysite` biaffine parser turns out to be both the most accurate and the fastest. It is also worth mentioning the significant training time, as for example `Zysite` takes more than 39 hours to train it on the Czech treebank, with 68,495 sentences. The experiments were performed on a Dell server using a single NVIDIA Tesla T4 GPU.

2.2 Tokenization, Tagging

UUParser does not provide tokenization nor tagging capabilities, so we have to rely on another set of tools to accomplish these tasks. We choose to use UDPipe (Straka and Straková, 2017) to perform sentence splitting, tokenization and tagging. This gives us a common tagged representation to use also with alternative parsers.

Some of the parsers tested provide the ability to perform end-to-end parsing from raw text, in particular UDify (Kondratyuk and Straka, 2019) and Stanza. However, they turn out not to be very effective: the pretrained model of UDify does not support all the task languages and Stanza has a weird behavior: for example, it would split a word like "GoogleOS" not just into two tokens, "Google" and "OS", but into two separate sentences.

So eventually we decided to use the same tokenization provided by UDPipe as input to all parsers. This enables us also to produce an ensemble version combining the outputs of three parsers.

2.3 Ensemble of Parsers

In the official submission, we exploit the linear tree combination algorithm by Attardi and Dell'Orletta (2009) to combine the outputs of an ensemble of dependency parsers. The algorithm is greedy and works by combining the trees top down. It has been shown to outperform more complex algorithms based on computing the Maximum Spanning Tree.

The parsers used are UDify, UUParser and UDPipe.

In a later unofficial submission labeled comb5, we included also `Zysite` and `Stanza` in the ensemble. Table 7 presents the results of this submission compared to the best performing official submission in the challenge.

These scores are within 1% UAS to the results of the top submission by Jenna Kanerva of the University of Turku, except on the Baltic languages (-3.37% Latvian, -4.91% Lithuanian, -4.98% Estonaian), Finnish (-4.42% UAS) and Arabic (-8.22%) and better on Tamil (+3.63%).

3 Enhanced Dependencies

For producing the enhanced dependencies we follow a "hybrid" approach, using a combination of an algorithmic graph transformation of the syntactic dependency tree coupled with predictions made by three machine learning classifiers. The basic enhancing script is an evolution of the work presented in (Simi and Montemagni, 2018) to bootstrap enhanced dependencies for the Italian treebank, also used for experiments in (Nivre et al. 2018).

One classifier is used to recognize the external subjects in `xcomp` constructions. The second classifier detects when a head should be propagated in conjunctions. The third classifier detects the case of propagation of dependents in conjunctions. The classifiers are trained jointly on the three tasks and produce three binary predictions.

The script that adds the enhanced dependencies is modular, so that it can be adapted to perform just the required analysis depending on the kind of enhanced dependencies present in each language and to bypass those that were not implemented. In addition, the script is parametric with respect to predictions coming from machine learning classifiers, which can be taken into account or ignored. We describe below how the different kinds of enhancements are addressed.

3.1 Controlled/Raised Subjects

This type of enhancement applies to subordinate infinitive clauses introduced by the `xcomp` relation and consists in adding an extra `nsubj` dependency to the embedded or controlled verb. The difficult aspect of this enhancement is to predict the correct

subject for the dependent clause among the different dependents of the main verb. In fact, this extra subject can be the subject, object or an oblique complement, as the following examples testify:

1. Mary wants to buy a book. *Mary* is the subject of *buy*.
2. Mary asked John to buy a book. *John*, the object, is the subject of *buy*.
3. Maria ha chiesto a Giovanni di comprare un libro. [Mary asked John to buy a book]. In Italian, the buyer, *Giovanni*, is an indirect complement (`obl`) of the main verb *chiesto*.

We train a neural binary classifier to predict which of the dependents of the main verb should be chosen to play the role of the extra subject for the dependent verb, if any. If more than one token is predicted as an external subject of the subordinate clause, currently all of them are added.

The classifier is applied to tokens that have a sibling in a `xcomp` relation, which are either a noun or a pronoun and whose `deprel` is one of the following: `nsubj`, `csubj`, `obj`, `iobj`, `obl`, `nsubj:pass`, `csubj:pass`.

Such tokens are represented by the following features: the form, the `upos` and the `deprel` of the token, the form, the `upos` and the `deprel` of the token's head, the form of the `xcomp` sibling, the form of the `case` or `mark` which introduces the subordinate phrase. A training example for the first classifier has the features for a token as input and a binary value as output depending on whether the sibling is indeed a `nsubj` for the subordinate clause.

3.2 Propagation over Conjuncts

The classifiers for propagation over conjuncts act in a similar way. We train two distinct classifiers for recognizing candidates for head propagation and for dependents propagation over conjuncts. Candidates for head propagation are conjoined subjects and objects, that should each be attached to their head as in "Paul and Mary are running" or "Paul bought apples and oranges".

Candidates for dependent propagation are subjects, objects and other complements of conjoined verbs, as it is the case of *she* in "She was reading and watching a movie".

The model is trained to predict whether a candidate for propagation should be safely propagated, by making the implicit relations explicit.

3.3 Model Architecture

The three classifiers share the same neural network architecture. The first layer collects the embeddings for each form, upos or deprel in the input vector. The embeddings for the forms are obtained from FastText (Bojanowski et al. 2017). The embeddings for `upos` and `deprel` are learned as vectors of size 20 each.

The second layer of the classifier concatenates the embeddings from the first layer. The third layer is a flatten layer, which is followed by a fully connected layer with a hidden dimension of 100. This is followed by a dropout with a probability of 50% (chosen by tuning experiments) and finally there is a fully connected layer with a sigmoid activation.

The classifiers are trained jointly with a binary cross entropy loss function and an Adam optimizer (Kingma and Ba, 2015) on the training set of each language. The training is run for up to four epochs, even though in most cases the loss stops decreasing after the second epoch. Validation accuracies during training range around 97-98%. The code is written in Keras on a Tensorflow backend.

3.4 Relative Clauses

The treatment of enhancements for relative clauses is quite straightforward. It consists in attaching the relative pronoun to its antecedent with the special `ref` relation and attaching the referred antecedent as an argument to the main predicate of the relative clause. This enhancement may create circularities in the enhanced graph.

3.5 Label Augmentation with Case/Mark Information

The most difficult sub-task turned out to be guessing the right case/mark information for augmenting the relation name of non-core dependents, due to the different interpretations and varying degree of compliance with the guidelines in the various treebanks. Given the high frequency of occurrence of this type of enhancements, doing this task right has high impact on the overall performance.

As it turned out, the differences concern all the following aspects, and their combinations:

1. the type of deprels considered for the augmentation (e.g. `conj` is not specialized in Arabic, Bulgarian, Estonian, Finnish, French, Latvian, Lithuanian, Polish etc.)
2. the case/mark information used (either the lemma or form of the case/mark dependent)

3. the strategy adopted in presence of multiple marks/cases dependents (whether their concatenation or the last one as in English)

4. the strategy adopted when cases/marks are fixed multi-word expressions (whether forms or lemmas are combined)

5. the use or not of morphological case information and to what extent

6. the presence of non canonical keywords in some languages (for example `agentxoxnsubj` and `enh` introduced in the French treebank to encode diathesis normalization as described by Candito et al. (2017)).

In this sense, the inclusion/exclusion of type specialization depending on the language is a too coarse strategy, since it does not account of all these variations; moreover the differences are treebank-wise (as opposed to language-wise) in the sense that different subparts of the test set for a specific language may be coming from different treebanks following different approaches.

In order to address these issues, we adopted a very simplistic data driven approach to adjust the result of a rule-based algorithm, which implements the guidelines. We computed a mapping from the label predicted from our enhancer to the gold label found in the training data set and filtered out correspondences whose frequency was less than a fixed threshold, in order to be tolerant to sporadic errors. As a final "patch", we applied the resulting transformation to produce the final augmented label.

This strategy is far from perfect and clean, but it does take care of systematic differences among languages, such as the use of case features (`gen`, tt nom, `dat`, tt ins etc.) in some of the languages with morphological cases. However, it provides no solution to issues related to non-conventional label completions, nor solves the problem of selecting the correct `mark` or `case` when multiple ones are present (e.g. *about whether*, *along with* in English), or to address the non-canonical use, with respect to the guidelines, of lemma vs form in augmentations[2].

[2]For case information the guidelines suggest the use of forms in multi-word expressions and lemmas for single words. English apparently adopts the inverse convention

Language	Parameters
Arabic	-e=4; ml; patch
Bulgarian	ml; patch
Czech	patch
Dutch	ml
English	
Estonian	ml; patch
Finnish	ml; patch
French	-e=156; ml
Italian	
Latvian	patch
Lithuanian	ml
Polish	ml
Russian	-e=3; patch
Slovak	patch
Swedish	
Tamil	-e=145; ml; patch
Ukrainian	patch

Table 4: Parameters resulting form tuning: see the text for their meaning.

3.6 Tuning Parameters

The machine learning modules and the "patch" strategy were not equally effective for all languages. On the basis of the performance on the development set, we selected for each language the best choice of parameters for the enhancement script. These were consistently applied in producing the enhanced version of the parser results in all submissions.

Table 4 summarizes the choice of parameters for the different languages, where the values for the parameters "-e" represent the types of enhancement to be excluded, since not implemented for the language (consistently with the parameters of the evaluation script), `ml` means that we used the predictions from the machine learning classifiers, `patch` means that we used the mappings strategy for fixing label augmentation. The lack of parameters means that only the basic enhancement script was used, and all enhancement types were performed.

4 Results

The official results are those labeled UNIPI-003 in our submission, obtained through the combination of the parsers UDify, UUParser and UDPipe.

Table 5 shows the official results obtained in tokenization and tagging on the test sets. Table 6 shows our team official results on parsing and en-

hancement.

After the submission deadline, we experimented with the biaffibe parsers Stanza and Zysite.

Stanza improves over UUParser by an average of 0.58 UAS, 0.64 LAS, 1.05 CLAS, 4.26 MLAS, 4.26 BLEX, 0.60 EULAS, 0.42 ELAS, with notable improvements of +16.82 LAS on Estonian, while a decrease of -28.52 LAS on Lithuanian, and -9.95 LAS on Polish is observed.

Zysite improves over UUParser by an average of 1.77 UAS, 1.84 LAS, 1.97 CLAS, 0.27 MLAS, 1.28 BLEX, 1.83 EULAS and 1.63 ELAS, with notable improvements of +17.49 LAS on Estonian, +5.02 on Dutch, +3.23 on Svedish, but with a significant drop of -14.48 LAS on Arabic.

These are encouraging results that show that a transition-based parser can be competitive with graph-based ones.

We then produced a new run `comb5` (UNIPI-comb5), as an ensemble of five parsers: UUParser, UDify, UDPipe, Stanza and Zysite. We report these unofficial results in Table 7.

The improvements on parsing by the ensemble of five parsers with respect to the single parser UUParser are summarized in Table 8.

The most significant improvements from the ensemble combinations are +13.07 UAS on Estonian, +5.31 on Tamil, +5.11 on Dutch, +3.59 on Lithuanian, +4.37 on Finnish.

Estonian, Finnish, Latvian, Lithuanian turned out as the most difficult for our dependency parsers, with a difference between 4.2 and 6.5 points of UAS with respect to the submission by Kanerva and even 10.7 point lower on Arabic.

If we consider the average UAS excluding the Baltic languages, the average UAS of the ensemble parsers is 89.82.

As for the enhancement task, its difficulty, besides what we discussed in section 3.5, seems to be confirmed by a significant drop from our EULAS score (restricted to UD relations) to the ELAS score, which also takes into account label enhancements. The average drop is 6.26 points and for some languages more than 10 points. The effectiveness of our 'patch' strategy had been carefully assessed with the development data, but did not provide analogous results on the test set. Our algorithm was poor in predicting the label extended with case information. Perhaps a machine learning approach would have provided better results in this case.

5 Conclusions

We experimented with both linear transition-based parsers and two implementations of graph-based biaffine parsers. All parsers have difficulties with Baltic languages, Finnish and Arabic which somehow we were able to mitigate by combining them into an ensemble, except for Arabic, which remains 8.2 points UAS lower than the top submission. Our enhanced version of UUParser, using BERT embeddings, performs competitively well with respect to the biaffine Zysite parser, except on Estonian, Tamil and Dutch, while it outperforms it by +14 LAS on Arabic. Since all the parsers use the same base model, multilingual uncased of BERT, it might be worthwhile to investigate how such models affect the performance on Baltic languages.

The implementation of the biaffine parser by Zysite was a surprising discovery, since it is capable to outperform in speed all other parsers, possibly due to its use of a more efficient biaffine operation via `torch.einsum()`.

For adding the enhanced relations to the output of the parser we opted for a hybrid approach, where for some languages, which appear to be more conforming to the guidelines, we applied an algorithmic solution, while for the rest we exploited machine learning classifiers.

In principle the algorithmic approach should be sufficient as soon as languages adhere more strictly to the guidelines. In the meanwhile, we wonder whether it is worthwhile to develop techniques which are language specific in order to obtain better results, unless there are ways to devise a language agnostic solution.

Acknowledgments

Simonetta Montemagni contributed to the design of a preliminary version of the enhancement script.

The experiments were run on a Dell server with four NVIDIA Tesla P100 GPUs partly funded through the grant "Grandi Attrezzature 2016" by the University of Pisa.

References

Giuseppe Attardi and Felice Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short*

Language	Tokens	Words	Sentences	UPOS	XPOS	UFeats	AllTags	Lemmas
Arabic	99.98	94.58	82.09	88.53	84.00	84.16	81.97	88.46
Bulgarian	99.91	99.91	94.17	97.62	94.34	95.39	93.78	94.60
Czech	99.88	99.88	93.18	97.83	90.88	90.77	89.71	97.42
Dutch	99.74	99.74	70.64	92.58	89.86	92.02	88.97	94.43
English	99.22	99.22	83.82	93.63	92.77	94.09	90.70	95.41
Estonian	99.37	99.37	76.33	82.96	85.77	78.78	75.43	76.11
Finnish	99.70	99.68	88.65	94.83	54.52	93.02	51.82	87.09
French	99.78	99.36	95.46	93.94	99.36	76.02	73.29	96.07
Italian	99.93	99.84	98.76	97.18	97.04	97.10	96.17	97.38
Latvian	99.33	99.33	98.74	93.48	84.29	89.55	83.93	92.73
Lithuanian	99.91	99.91	87.87	90.33	80.69	81.20	79.33	88.75
Polish	99.40	99.83	97.52	96.43	84.87	83.62	80.35	95.60
Russian	99.60	99.60	98.80	97.78	99.60	85.34	84.97	96.55
Slovak	100.00	100.00	85.28	92.93	77.06	80.34	76.71	86.56
Swedish	98.95	98.95	94.07	90.92	0.00	76.18	0.00	88.16
Tamil	99.16	94.51	97.52	81.31	76.35	80.45	75.59	84.14
Ukrainian	99.85	99.81	96.61	94.91	84.03	84.28	83.32	93.56
Average	**99.63**	**99.03**	**90.56**	**92.78**	**80.91**	**86.02**	**76.83**	**91.35**

Table 5: Official results on tagging the test set, produced by UDPipe.

Papers, pages 261–264, Boulder, Colorado. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2020. Overview of the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, Seattle, US. Association for Computational Linguistics.

Marie Candito, Bruno Guillaume, Guy Perrie, and Djamé Seddah. 2017. Enhanced UD dependencies with neutralized diathesis alternation. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 42–53.

Danqi Chen and Christopher 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.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. 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)*, pages 710–716, Minneapolis, Minnesota. Association for Computational Linguistics.

Jenna Kanerva, Filip Ginter, Niko Miekka, Akseli Leino, and Tapio Salakoski. 2018. Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 133–142, Brussels, Belgium. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

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.

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally. 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 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Language	UAS	LAS	CLAS	MLA	BLEX	EULAS	ELAS
Arabic	76.46	71.48	67.41	57.26	62.70	68.66	57.79
Bulgarian	91.66	88.26	84.82	79.11	77.98	86.77	84.93
Czech	91.95	89.64	87.99	76.21	85.25	86.19	75.99
Dutch	84.43	80.53	73.42	65.08	67.75	78.95	77.62
English	88.22	85.10	82.21	73.74	77.79	84.54	83.95
Estonian	70.57	63.18	60.04	46.69	42.79	62.45	57.24
Finnish	85.17	81.25	78.79	72.53	66.33	79.04	72.13
French	88.09	82.58	75.37	41.81	71.93	81.84	78.85
Italian	93.04	90.69	86.57	82.55	83.04	89.77	89.14
Latvian	85.47	81.25	78.25	66.47	72.02	78.44	68.23
Lithuanian	76.99	70.76	67.48	51.75	58.40	67.16	61.06
Polish	90.97	87.53	85.24	64.92	80.26	84.83	70.61
Russian	92.44	90.52	89.20	69.95	85.45	88.34	76.90
Slovak	91.24	88.95	87.02	62.99	71.95	85.93	81.40
Swedish	84.80	80.92	78.63	49.09	66.77	79.90	78.73
Tamil	62.79	54.69	51.62	42.50	45.29	54.59	48.50
Ukrainian	88.96	85.23	82.16	63.78	74.89	82.51	73.90
Average	**84.90**	**80.74**	**77.42**	**62.73**	**70.03**	**78.82**	**72.76**

Table 6: UNIPI Official results on parsing the test set: ensemble of UUParser, UDify and UDPipe.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Arc-hybrid non-projective dependency parsing with a static-dynamic oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104, Pisa, Italy. 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, Melbourne, Australia. Association for Computational Linguistics.

Joakim Nivre, Paola Marongiu, Filip Ginter, Jenna Kanerva, Simonetta Montemagni, Sebastian Schuster, and Maria Simi. 2018. Enhancing universal dependency treebanks: A case study. In *Proceedings of the Second Workshop on Universal Dependencies, UDW@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 102–107. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. *ArXiv*, abs/2003.07082.

Maria Simi and Simonetta Montemagni. 2018. Bootstrapping Enhanced Universal Dependencies for Italian. In *Proceedings of the Fifth Italian Conference on Computational Linguistics CLiC-it 2018, 10-12 December 2018, Torino*, pages 348–353. Torino: Accademia University Press.

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.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Y. Zhang. 2019. A pytorch implementation of "deep biaffine attention for neural dependency parsing".

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on Penn treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

	UniPI run comb5					University of Turku				
Language	**UAS**	**LAS**	**CLAS**	**MLA**	**BLEX**	**UAS**	**LAS**	**CLAS**	**MLA**	**BLEX**
Arabic	77.14	72.97	69.41	58.22	64.31	85.36	81.17	78.81	72.15	75.60
Bulgarian	93.72	90.73	87.72	81.10	80.36	95.07	92.48	89.94	85.96	87.85
Czech	92.89	90.79	89.35	77.26	86.53	92.94	90.83	89.39	81.25	87.53
Dutch	89.54	86.71	81.57	70.29	74.92	90.02	87.20	82.33	75.95	78.84
English	89.09	86.70	84.14	74.93	79.56	91.13	88.97	87.15	81.45	84.71
Estonian	83.64	80.18	78.88	53.55	53.20	88.62	85.86	84.61	79.01	81.40
Finnish	89.54	86.99	85.20	76.87	70.86	93.96	92.50	91.65	87.11	87.60
French	91.54	87.40	83.09	45.72	78.86	91.26	87.85	82.94	70.36	80.07
Italian	94.47	92.66	89.04	84.09	85.08	94.71	93.31	90.34	86.91	88.45
Latvian	88.01	84.54	82.10	68.75	75.16	91.38	88.53	86.64	77.43	82.86
Lithuanian	80.58	74.88	72.03	53.17	61.65	85.49	81.85	79.88	66.23	74.04
Polish	93.42	90.66	88.79	66.76	83.48	94.38	91.82	90.36	77.36	88.11
Russian	93.86	92.45	91.23	71.35	87.30	94.06	92.74	91.84	88.10	89.97
Slovak	92.65	90.43	88.85	63.80	72.74	93.40	91.57	90.60	77.76	86.76
Swedish	88.16	85.18	83.45	51.43	70.60	90.81	88.31	87.29	71.65	80.37
Tamil	68.10	61.32	58.46	51.39	53.86	64.47	59.66	57.72	47.18	53.84
Ukrainian	90.38	87.71	85.23	65.59	77.67	91.65	89.68	87.41	76.92	84.81
Average	**88.04**	**84.84**	**82.27**	**65.55**	**73.89**	**89.92**	**87.31**	**85.23**	**76.63**	**81.93**

Table 7: Unofficial results on parsing: on the left our submission, on the right the best submission.

Language	**UAS**	**LAS**	**CLAS**	**MLAS**	**BLEX**	**EULAS**	**ELAS**
Arabic	0.68	1.49	2.00	0.96	1.61	1.49	0.95
Bulgarian	2.06	2.47	2.90	1.99	2.38	2.44	2.41
Czech	0.94	1.15	1.36	1.05	1.28	1.10	0.93
Dutch	5.11	6.18	8.15	5.21	7.17	6.08	5.92
English	0.87	1.60	1.93	1.19	1.77	1.61	1.72
Estonian	13.07	17.00	18.84	6.86	10.41	16.72	14.50
Finnish	4.37	5.74	6.41	4.34	4.53	5.60	4.97
French	3.45	4.82	7.72	3.91	6.93	4.89	4.58
Italian	1.43	1.97	2.47	1.54	2.04	2.04	2.09
Latvian	2.54	3.29	3.85	2.28	3.14	3.22	2.53
Lithuanian	3.59	4.12	4.55	1.42	3.25	4.03	3.83
Polish	2.45	3.13	3.55	1.84	3.22	3.14	2.48
Russian	1.42	1.93	2.03	1.40	1.85	1.90	1.52
Slovak	1.41	1.48	1.83	0.81	0.79	1.46	0.97
Swedish	3.36	4.26	4.82	2.34	3.83	4.20	4.42
Tamil	5.31	6.63	6.84	8.89	8.57	6.53	5.53
Ukrainian	1.42	2.48	3.07	1.81	2.78	2.31	2.37
Average	**3.14**	**4.10**	**4.85**	**2.82**	**3.86**	**4.04**	**3.63**

Table 8: Improvements by parser combination on unofficial run.