Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe

Milan Straka and **Jana Straková**
Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
{straka,strakova}@ufal.mff.cuni.cz

Abstract

We present an update to UDPipe 1.0 (Straka et al., 2016), a trainable pipeline which performs sentence segmentation, tokenization, POS tagging, lemmatization and dependency parsing. We provide models for all 50 languages of UD 2.0, and furthermore, the pipeline can be trained easily using data in CoNLL-U format.

For the purpose of the *CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, the updated UDPipe 1.1 was used as one of the baseline systems, finishing as the 13th system of 33 participants. A further improved UDPipe 1.2 participated in the shared task, placing as the 8th best system, while achieving low running times and moderately sized models.

The tool is available under open-source Mozilla Public Licence (MPL) and provides bindings for C++, Python (through `ufal.udpipe` PyPI package), Perl (through `UFAL::UDPipe` CPAN package), Java and C#.

1 Introduction

The Universal Dependencies project (Nivre et al., 2016) seeks to develop cross-linguistically consistent treebank annotation of morphology and syntax for many languages. The latest version of UD (Nivre et al., 2017a) consists of 70 dependency treebanks in 50 languages. As such, the UD project represents an excellent data source for developing multi-lingual NLP tools which perform sentence segmentation, tokenization, POS tagging, lemmatization and dependency tree parsing.

The goal of the *CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* (*CoNLL 2017 UD Shared Task*) is to stimulate research in multi-lingual dependency parsers which process raw text only. The overview of the task and the results are presented in Zeman et al. (2017).

This paper describes UDPipe (Straka et al., 2016)[1] – an open-source tool which automatically generates sentence segmentation, tokenization, POS tagging, lemmatization and dependency trees, using UD version 2 treebanks as training data.

The contributions of this paper are:

- Description of *UDPipe 1.1 Baseline System*, which was used to provide baseline models for *CoNLL 2017 UD Shared Task* and preprocessed test sets for the *CoNLL 2017 UD Shared Task* participants. UDPipe 1.1 provided a strong baseline for the task, placing as the 13$^{\text{th}}$ (out of 33) best system in the official ranking. The *UDPipe 1.1 Baseline System* is described in Section 3.
- Description of *UDPipe 1.2 Participant System*, an improved variant of UDPipe 1.1, which was used as a contestant system in the *CoNLL 2017 UD Shared Task*, finishing 8$^{\text{th}}$ in the official ranking, while keeping very low software requirements. The *UDPipe 1.2 Participant System* is described in Section 4.
- Evaluation of search-based oracle and several transition-based system on UD 2.0 dependency trees (Section 5).

2 Related Work

There is a number of NLP pipelines available, e.g., Natural Language Processing Toolkit[2] (Bird et al.,

[1] http://ufal.mff.cuni.cz/udpipe
[2] NLTK, http://nltk.org

Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, pages 88–99,
Vancouver, Canada, August 3-4, 2017.

2009) or OpenNLP[3] to name a few. We designed yet another one, UDPipe, with the aim to provide extremely simple tool which can be trained easily using only a CoNLL-U file without additional resources or feature engineering.

Deep neural networks have recently achieved remarkable results in many areas of machine learning. In NLP, end-to-end approaches were initially explored by Collobert et al. (2011). With a practical method for precomputing word embeddings (Mikolov et al., 2013) and routine utilization of recurrent neural networks (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), deep neural networks achieved state-of-the-art results in many NLP areas like POS tagging (Ling et al., 2015), named entity recognition (Yang et al., 2016) or machine translation (Vaswani et al., 2017). The wave of neural network parsers was started recently by Chen and Manning (2014) who presented fast and accurate transition-based parser. Many other parser models followed, employing various techniques like stack LSTM (Dyer et al., 2015), global normalization (Andor et al., 2016), biaffine attention (Dozat and Manning, 2016) or recurrent neural network grammars (Kuncoro et al., 2016), improving LAS score in English and Chinese dependency parsing by more than 2 points in 2016.

3 UDPipe 1.1 Baseline System

UDPipe 1.0 (Straka et al., 2016)[4] is a trainable pipeline performing sentence segmentation, tokenization, POS tagging, lemmatization and dependency parsing. It is fully trainable using CoNLL-U version 1 files and the pretrained models for UD 1.2 treebanks are provided.

For the purpose of the *CoNLL 2017 UD Shared Task*, we implemented a new version UDPipe 1.1 which processes CoNLL-U version 2 files. UDPipe 1.1 was used as one of the baseline systems in the shared task. *UDPipe 1.1 Baseline System* was trained and tuned in the training phase of *CoNLL 2017 UD Shared Task* on the UD 2.0 training data and the trained models and outputs were available to the participants.

In this Section, we describe the *UDPipe 1.1 Baseline System*, focusing on the differences to the previous version described in (Straka et al., 2016): the tokenizer (Section 3.1), the tagger (Section 3.2), the parser (Section 3.3), the hyperparameter search support (Section 3.4), the training details (Section 3.5) and evaluation (Section 3.6).

3.1 Tokenizer

In UD and in CoNLL-U files, the text is structured on several levels – a *document* consists of *paragraphs* composed of (possibly partial) *sentences*, which are sequences of *tokens*. A token is also usually a *word* (unit used in further morphological and syntactic processing), but a single token may be composed of several syntactic words (for example, token *zum* consists of words *zu* and *dem* in German). The original text can be therefore reconstructed as a concatenation of tokens with adequate spaces, but not as a concatenation of words.

Sentence Segmentation and Tokenization

Sentence segmentation and tokenization is performed jointly (as it was in UDPipe 1.0) using a single-layer bidirectional GRU network which predicts for each character whether it is the last one in a sentence, the last one in a token, or not the last one in a token. Spaces are usually not allowed in tokens and therefore the network does not need to predict end-of-token before a space (it only learns to separate adjacent tokens, like for example *Hi!* or *cannot*).

Multi-Word Token Splitting

In UDPipe 1.0, a case insensitive dictionary was used to split tokens into words. This approach is beneficial if there is a fixed number of multi-word tokens in the language (which is the case for example in German).

In *UDPipe 1.1 Baseline System* we also employ automatically generated suffix rules – a token with a specific suffix is split, using the non-matching part of the token as prefix of the first words, and a fixed sequence of first word suffix and other words (e.g, in Polish we create a rule $\star łem \rightarrow \star ł + em$). The rules are generated automatically by keeping all such rules present in the training data, which do not trigger incorrectly too often. The contribution of suffix rules is evaluated in Section 5.

Documents and Paragraphs

We use an improved sentence segmenter in *UDPipe 1.1 Baseline System*. The segmenter learns sentence boundaries in the text in a standard way as in *UDPipe 1.1 Baseline System*, but it omits the sentence breaks at the end of a paragraph or a document. The reason for excluding these boundaries

[3] https://opennlp.apache.org
[4] http://ufal.mff.cuni.cz/udpipe

from the training data is that the ends of paragraphs and documents are frequently recognized by layout (e.g. newspaper headlines) and if the recognizer is trained to recognize these sentence breaks, it tends to erroneously split regular sentences.

Additionally, we now also mark paragraph boundaries (recognized by empty lines) and document boundaries (corresponding to files being processed, storing file names as document ids) when running the segmenter.

Spaces in Tokens

Additional feature allowed in CoNLL-U version 2 files is presence of spaces in tokens. If spaces in tokens are allowed, the GRU tokenizer network must be modified to predict token breaks in front of spaces. On the other side, many UD 2.0 languages do not allow spaces in tokens (and in such languages a space in a token might confuse the following systems in the pipeline), therefore, it is configurable whether spaces in tokens are allowed, with the default being to allow spaces in tokens if there is any token with spaces in the training data.

Precise Reconstruction of Spaces

Unfortunately, neither CoNLL-U version 1 nor version 2 provide a standardized way of storing inter-token spaces which would allow reconstructing the original plain text. Therefore, *UDPipe 1.1 Baseline System* supports several UDPipe-specific MISC fields that are used for this purpose.

CoNLL-U defines `SpaceAfter=No` MISC feature which denotes that a given token is not followed by a space. We extend this scheme in a compatible way by introducing `SpacesAfter=`*`spaces`* and `SpacesBefore=`*`spaces`* fields. These fields store the spaces following and preceding this token, with `SpacesBefore` by default empty and `SpacesAfter` being by default empty or one space depending on `SpaceAfter=No` presence. Therefore, these fields are not needed if tokens are separated by no space or a single space. The spaces are encoded by a means of a C-like escaping mechanism, with escape sequences `\s`, `\t`, `\r`, `\n`, `\p`, `\\` for space, tab, CF, LF, `|` and `\` characters, respectively.

If spaces in tokens are allowed, these spaces cannot be represented faithfully in the FORM field which disallows tabs and new line characters. Therefore, UDPipe utilizes an additional MISC field `SpacesInToken=`*`token_with_spaces`* representing the token with original spaces. Once again, with the default value being the value of the FORM field, the field is needed only if the token spaces cannot be represented in the FORM field.

All described MISC fields are generated automatically by *UDPipe 1.1 Baseline System* tokenizer, with `SpacesBefore` used only at the beginning of a sentence.

Furthermore, we also provide an optional way of storing the document-level character offsets of all tokens, using `TokenOffset` MISC field. The values of this field employ Python-like `start:end` format.

Detokenization

To train the tokenizer, the original plain texts of the CoNLL-U files are required. These plain texts can be reconstructed using the `SpaceAfter=No` feature. However, very little UD version 1 corpora contains this information. Therefore, UDPipe 1.0 offers a way of generating these features using a different raw text in the concerned language (Straka et al., 2016).

Fortunately, most UD 2.0 treebanks do include the `SpaceAfter=No` feature. We perform detokenization only for Dannish, Finnish-FTB and Slovenian-SST.

Inference

When employing the segmenter and tokenizer GRU network during inference, it is important to normalize spaces in the given text. The reason is that during training, tokens were either adjacent or separated by a single space, so we need to modify the network input during inference accordingly.

During inference, we precompute as much network operations on character embeddings as possible[5] (to be specific, we cache 6 matrix products for every character embedding in each GRU). Consequently, the inference is almost twice as fast.

3.2 Tagger

The tagger utilized by *UDPipe 1.1 Baseline System* is nearly identical to the previous version in UDPipe 1.0. A guesser generates several (UPOS, XPOS, FEATS) triplets for each word according to its last four characters, and an averaged perceptron tagger with a fixed set of features disambiguates the generated tags (Straka et al., 2016; Straková et al., 2014).

[5] Similarly to Devlin et al. (2014).

The lemmatizer is analogous. A guesser produces (lemma rule, UPOS) pairs, where the lemma rule generates a lemma from a word by stripping some prefix and suffix and prepending and appending new prefix and suffix. To generate correct lemma rules, the guesser generates the results not only according to the last four characters of a word, but also using word prefix. Again, the disambiguation is performed by an averaged perceptron tagger.

We prefer to perform lemmatization and POS tagging separately (not as a joint task), because we found out that utilization of two different guessers and two different feature sets improves the performance of our system (Straka et al., 2016).

The only change in *UDPipe 1.1 Baseline System* is a possibility to store lemmas not only as lemma rules, i.e., relatively, but also as "absolute" lemmas. This change was required by the fact that some languages such as Persian contain a lot of empty lemmas which are difficult to encode using relative lemma rules, and because Latin-PROIEL treebank uses `greek.expression` lemma for all Greek forms.

3.3 Dependency Parsing

UDPipe 1.0 utilizes fast transition-based neural dependency parser. The parser is based on a simple neural network with just one hidden layer and without any recurrent connections, using locally-normalized scores.

The parser offers several transition systems – a projective arc-standard system (Nivre, 2008), partially non-projective link2 system (Gómez-Rodríguez et al., 2014) and a fully non-projective swap system (Nivre, 2009). Several transition oracles are implemented – static oracles, dynamic oracle for the arc-standard system (Goldberg et al., 2014) and a search-based oracle (Straka et al., 2015). Detailed description of the parser architecture and transition systems and oracles can be found in Straka et al. (2016) and Straka et al. (2015).

The parser makes use of FORM, UPOS, FEATS and DEPREL embeddings. The form embeddings are precomputed with `word2vec` using the training data, the other embeddings are initialized randomly, and all embeddings are updated during training.

We again precompute as much network operations as possible for the input embeddings. However, to keep memory requirements and loading times reasonable, we do so only for 1000 most frequent embeddings of every type.

Because the *CoNLL 2017 UD Shared Task* did not allow sentences with multiple roots, we modified all the transition systems in UDPipe 1.1 to generate only one root node and to use the `root` dependency relation only for this node.

3.4 Hyperparameter Search Support

All three described components employ several hyperparameters which can improve performance if tuned correctly. To ease up the process, UDPipe offers random hyperparameter search for all the components – the `run=`*`number`* option during training generates pseudorandom but deterministic values for predefined hyperparameters. The hyperparameters are supposed to be tuned for every component individually, and then merged.

3.5 Training the UDPipe 1.1 Baseline System

When developing the *UDPipe 1.1 Baseline System* in the training phase of *CoNLL 2017 UD Shared Task*, the testing data were not yet available for the participants. Therefore a new data split was created from the available training and development data: the performance of the models was evaluated on the development data, and part of the training data was put aside and used to tune the hyperparameters. This *baselinemodel-split* of the UD 2.0 data is provided together with the baseline modes from Straka (2017).

The following subsections describe the details of training the *UDPipe 1.1 Baseline System*.

Tokenizer

The segmenter and tokenizer network employs character embeddings and GRU cells of dimension 24. The network was trained using dropout both before and after the recurrent units, using the Adam optimization algorithm (Kingma and Ba, 2014). Suitable batch size, dropout probability, learning rate and number of training epochs was tuned on the *tune* set.

Tagger

The tagger and the lemmatizer do not use any hyperparameters which require tuning. The guesser hyperparameter were tuned on the tune set.

Parser

The parser network employs form embeddings of dimension 50, and UPOS, FEATS and DEPREL

embeddings of dimension 20. The hidden layer has dimension 200, batch consists of 10 words and the network was trained for 10 iterations. The suitable transition system, oracle, learning rate and L2 regularization was chosen to maximize the accuracy on the tune set.

3.6 Evaluation of the UDPipe 1.1 Baseline System

There are three testing collections in *CoNLL 2017 UD Shared Task*: UD 2.0 test data, new parallel treebank (PUD) sets, and four surprise languages.

The *UDPipe 1.1 Baseline System* models were completely trained, released and "frozen" on the UD 2.0 training and development data with a new split (see the previous Section 3.5) already in the training phase of the *CoNLL 2017 UD Shared Task* on the UD 2.0 training data, unlike the participant systems, which could use the full training data for training and development data for tuning.

We used the *UDPipe 1.1 Baseline System* models for evaluation of the completely new parallel treebank (PUD) set and completely new surprise languages in the following way:

For the new parallel treebank sets we utilized the "main" treebank for each language (e.g., for Finish `fi` instead of `fi_ftb`). This arbitrary decision was a lucky one – after the shared task evaluation, the performance on the parallel treebanks was shown to be significantly worse if different treebanks than the "main" were used (even if they were larger or provided higher LAS on their own test set). The reason seem to be the inconsistencies among the treebanks of the same language – the Universal Dependencies are yet not so universal as everyone would like.

To parse the surprise languages, we employed a baseline model which resulted in highest LAS F1-score on the surprise language sample data – resulting in Finnish FTB, Polish, Finnish FTB and Slovak models for the surprise languages Buryat, Kurmanji, North Sámi and Upper Sorbian, respectively. Naturally, most words of a surprise language are not recognized by a baseline model for a different language. Conveniently, the UPOS tags and FEATS are shared across languages, allowing the baseline model to operate similarly to a delexicalized parser.

4 UDPipe 1.2 Participant System

We further updated the *UDPipe 1.1 Baseline System* to participate in *CoNLL 2017 UD Shared Task* with an improved *UDPipe 1.2 Participant System*.

As participants of the shared task, we trained the system using the whole training data and searched for hyperparameters using the development data (instead of using the *baselinemodel-split* described in Section 3.5). Although the data size increase is not exactly a change in the system itself, it improves performance, especially for smaller treebanks.

4.1 Hyperparameter Changes

While tokenization and segmentation is straightforward in some languages, it is quite complex in others (notably in Japanese and Chinese, which do not use spaces for word separation, or in Vietnamese, in which many tokens contain spaces). In order to improve the performance on these languages we increased the embedding dimension and GRU cell dimension in the tokenizer from 24 to 64.

We increased form embedding dimension in the parser from 50 to 64 (larger dimensions showed no more improvements on the development set) and also trained the parser for 20 iterations over the training data instead of 10.

Furthermore, instead of using beam of size 5 during parsing as in *UDPipe 1.1 Baseline System*, we tuned the beam size individually for each treebank, choosing 5, 10, 15 or 20 according to resulting LAS on a development set.

4.2 Merging Treebanks of the Same Language

For several languages, there are multiple treebanks available in the UD 2.0 collection. Ideally, one would merge all training data of all treebanks of a given language. However, according to our preliminary experiments, the annotation is not perfectly consistent even across treebanks of the same language. Still, additional training data, albeit imperfect, could benefit small treebanks.

We therefore attempt to exploit these multiplex treebanks by enriching each treebank's training data with training data from other treebanks of the same language. Given a treebank for which another treebanks of the same language exist, we evaluate performance of several such expansions

Treebank	Maximum sentence length	Changed sentence boundary log-probability	Every sentence log-probability
Gothic	20	-0.5	-0.9
Latin-PROIEL	25	-0.4	-0.7
Slovenian-SST	15	-0.7	-0.9

Table 1: Hyperparameters for joint segmentation and parsing.

and choose the best according to LAS score on the development data of the treebank in question. We extend the original training data by adding random sentences from the additional treebanks of the same language – we consider subsets containing $\frac{1}{4}$, $\frac{1}{2}$, 1 and 2 times the size of the original treebank.

4.3 Joint Sentence Segmentation and Parsing

Some treebanks are very difficult to segment into sentences due to missing punctuation, which harms the parser performance. We segment three smallest treebanks of this kind (namely Gothic, Latin-PROIEL and Slovenian-SST) jointly with the parser, by choosing such sentence segmentation which maximizes likelihood of their parse trees.

In order to determine the segmentation with maximum parsing likelihood, we evaluate every possible segmentation with sentences up to a given maximum length L. Because likelihoods of parse trees are independent, we can utilize dynamic programming and find the best segmentation in polynomial time by parsing sentences of lengths 1 to L at every location in the original text. Therefore, the procedure has the same complexity as parsing text which is circa $L^2/2$ times longer than the original one.

Additionally, we incorporate the segmentation suggested by the tokenizer in the likelihood of the parse trees – we multiply the tree likelihood by a fixed probability for each sentence boundary different than the one returned by the tokenizer.

However, if a transition-based parser is used, the optimum solution for the algorithm described so far would probably be to segment the text into one-token sentences, due to the fact that for a single word there is only one possible sequence of transitions (to make the word a root node), which has therefore probability one. Consequently, we introduce a third hyperparameter, which is an additional "cost" for every sentence.

We tuned the three described hyperparameters for every treebank independently to maximize LAS score on development set. The chosen hyperparameter values are shown in Table 1.

We expect graphical parsing models to benefit even more from this kind of joint segmentation – for every word, one can compute the probability distribution of attaching it as a dependent to all words within a distance of L (including the word itself, which represents the word being a root node). Then, the likelihood of a single-word sentence would not be one, but would take into account the possibility of attaching the word as a dependent to every near word.

5 Experiments and Results

The official *CoNLL 2017 UD Shared Task* evaluation was performed using a TIRA platform (Potthast et al., 2014), which provided virtual machines for every participants' systems. During test data evaluation, the machines were disconnected from the internet, and reset after the evaluation finished – this way, the entire test sets were kept private even during the evaluation.

In addition to official results, we also report results of supplementary experiments. These were evaluated after the shared task, using the released test data (Nivre et al., 2017b). All results are produced using the official evaluation script.

Because only plain text (and not gold tokenization) is used as input, all results are in fact F1-scores and always take tokenization performance into account.

The complete *UDPipe 1.2 Participant System* scores are shown in Table 2. We also include LAS F1-score of the *UDPipe 1.1 Baseline System* for reference. Note that due to time constraints, some *UDPipe 1.2 Participant System* submitted models did not generate any XPOS and lemmas. In these cases, we show XPOS and lemmatization results using post-competition models and typeset them in italic.

Treebank	UDPipe 1.2 Participant System										Baseline
	Tokens	Words	Sents	Words	UPOS	XPOS	Feats	AllTags	Lemmas	UAS	LAS	LAS
Ancient Greek	99.96	99.96	98.73	99.96	85.55	*43.69*	73.30	*43.67*	*82.89*	65.37	**57.39**	56.04
Ancient Greek-PROIEL	100.00	100.00	47.09	100.00	95.60	*93.34*	87.66	*84.85*	*92.73*	71.72	**66.51**	65.22
Arabic	99.98	93.71	81.77	93.71	88.26	83.27	83.40	82.08	87.34	71.69	**66.06**	65.30
Basque	99.96	99.96	99.50	99.96	92.33	99.96	87.25	84.66	93.49	75.59	**70.45**	69.15
Bulgarian	99.92	99.92	92.85	99.92	97.72	94.57	95.55	94.01	94.60	88.82	**84.92**	83.64
Catalan	99.97	99.97	99.03	99.97	98.00	98.00	97.20	96.56	97.87	88.69	**85.53**	85.39
Chinese	89.55	89.55	98.20	89.55	83.47	83.38	88.28	82.13	89.54	61.81	**57.89**	57.40
Croatian	99.90	99.90	95.56	99.90	95.88	99.90	84.34	83.43	94.33	83.73	**77.73**	77.18
Czech	99.93	99.93	92.30	99.93	98.23	92.71	91.97	91.60	97.82	86.73	**83.19**	82.87
Czech-CAC	99.97	99.96	100.00	99.96	98.34	*91.92*	90.53	*90.36*	*97.31*	88.21	**84.40**	82.46
Czech-CLTT	99.34	99.34	94.19	99.34	95.49	*88.07*	86.14	*85.04*	*96.79*	80.52	**76.69**	71.64
Danish	99.60	99.60	78.97	99.60	95.28	99.60	94.37	93.25	94.51	78.91	**75.28**	73.38
Dutch	99.80	99.80	76.95	99.80	91.33	88.05	89.23	86.94	89.77	76.50	**70.52**	68.90
Dutch-LassySmall	99.99	99.99	81.83	99.99	97.43	99.99	97.17	96.39	97.99	82.76	**80.15**	78.15
English	99.03	99.03	75.33	99.03	93.50	92.88	94.44	91.48	96.10	80.34	**77.25**	75.84
English-LinES	99.92	99.92	87.40	99.92	94.87	*92.01*	99.39	*90.41*	*98.34*	79.06	**74.92**	72.94
English-ParTUT	99.57	99.55	98.40	99.55	93.41	*91.92*	91.45	*89.83*	*96.39*	81.13	**76.89**	73.64
Estonian	99.89	99.89	93.66	99.89	87.60	89.98	81.14	78.99	80.96	68.65	**60.01**	58.79
Finnish	99.69	99.69	86.75	99.69	94.49	95.68	91.42	90.35	86.49	80.74	**77.26**	73.75
Finnish-FTB	99.97	99.96	85.54	99.96	92.28	91.05	92.53	89.41	88.68	79.69	**75.31**	74.03
French	99.76	98.88	94.58	98.88	95.49	98.88	95.42	94.26	96.59	84.09	80.50	**80.75**
French-ParTUT	99.85	98.97	97.76	98.97	95.38	*85.35*	91.23	*82.06*	*94.87*	84.03	**80.17**	77.38
French-Sequoia	99.76	99.06	84.60	99.06	95.63	99.06	94.74	93.59	96.82	84.06	**81.35**	79.98
Galician	99.93	99.93	96.18	99.93	96.93	96.44	99.70	96.08	96.93	80.95	**77.73**	77.31
Galician-TreeGal	99.62	98.66	85.35	98.66	91.08	87.70	89.84	86.90	92.56	71.59	**66.31**	65.82
German	99.67	99.67	79.35	99.67	90.72	94.65	80.46	76.26	95.38	74.15	68.61	**69.11**
Gothic	100.00	100.00	24.12	100.00	94.32	94.87	87.06	85.03	92.45	69.26	**62.80**	59.81
Greek	99.87	99.87	90.00	99.87	95.35	95.35	89.89	88.62	94.44	84.31	**80.67**	79.26
Hebrew	99.98	85.16	100.00	85.16	80.87	80.87	77.57	76.78	79.58	62.06	**57.86**	57.23
Hindi	100.00	100.00	99.20	100.00	95.75	94.82	90.12	87.57	98.00	91.45	**87.28**	86.77
Hungarian	99.81	99.81	95.54	99.81	90.80	99.81	70.59	69.57	88.40	72.36	**66.54**	64.30
Indonesian	100.00	100.00	91.73	100.00	93.43	100.00	99.52	93.42	100.00	81.67	**75.47**	74.61
Irish	99.40	99.40	94.78	99.40	88.86	87.90	76.27	73.53	85.45	73.10	**62.87**	61.52
Italian	99.91	99.83	97.11	99.83	97.31	*97.06*	97.20	*96.26*	*97.34*	88.62	**86.11**	85.28
Japanese	90.97	90.97	95.01	90.97	88.19	90.97	90.95	88.19	90.19	75.81	**74.49**	72.21
Kazakh	96.07	95.63	81.23	95.63	50.69	50.56	46.06	39.57	59.46	41.77	**25.43**	24.51
Korean	99.69	99.69	92.41	99.69	94.22	89.13	99.34	89.13	99.32	66.64	**60.30**	59.09
Latin	99.99	99.99	98.56	99.99	83.66	68.03	72.75	68.02	51.85	57.57	**47.02**	43.77
Latin-ITTB	99.89	99.89	82.58	99.89	96.83	*91.58*	93.50	*89.71*	*97.61*	79.74	75.84	**76.98**
Latin-PROIEL	100.00	100.00	19.56	100.00	95.00	95.08	87.94	86.89	94.91	66.45	**61.55**	57.54
Latvian	98.94	98.94	98.32	98.94	88.40	75.00	82.02	74.45	86.76	68.38	**61.80**	59.95
Norwegian-Bokmaal	99.79	99.79	96.38	99.79	96.83	99.79	95.25	94.38	96.66	86.62	**83.89**	83.27
Norwegian-Nynorsk	99.93	99.93	92.08	99.93	96.54	99.93	95.02	94.15	96.48	85.86	**82.74**	81.56
Old Church Slavonic	99.99	99.99	40.94	99.99	93.55	93.60	86.72	85.43	90.69	72.60	**66.29**	62.76
Persian	100.00	99.65	97.76	99.65	96.02	95.94	96.09	95.36	93.58	84.18	**80.33**	79.24
Polish	99.98	99.87	99.18	99.87	95.43	83.36	83.46	81.35	93.34	86.31	**80.21**	78.78
Portuguese	99.66	99.54	89.24	99.54	96.30	72.63	93.36	71.59	96.70	86.30	**82.72**	82.11
Portuguese-BR	99.96	99.86	96.71	99.86	97.07	97.07	99.72	97.05	98.75	88.18	**85.97**	85.36
Romanian	99.67	99.67	93.72	99.67	96.62	95.87	96.05	95.71	96.54	85.74	**80.32**	79.88
Russian	99.90	99.90	96.59	99.90	94.69	94.38	84.17	82.61	74.91	80.94	**76.15**	74.03
Russian-SynTagRus	99.58	99.58	97.97	99.58	97.91	99.58	93.45	93.11	95.43	89.35	**86.80**	86.76
Slovak	100.00	100.00	84.26	100.00	92.85	77.32	79.61	76.93	86.17	80.78	**75.63**	72.75
Slovenian	99.96	99.96	98.86	99.96	96.11	88.01	88.33	87.50	95.27	85.37	**81.84**	81.15
Slovenian-SST	99.87	99.87	13.13	99.87	91.78	*86.40*	85.32	*82.33*	*93.79*	59.26	**53.94**	46.45
Spanish	99.91	99.74	95.26	99.74	95.54	99.74	96.10	93.70	95.89	85.32	**81.95**	81.47
Spanish-AnCora	99.97	99.95	98.26	99.95	98.14	98.14	97.57	96.89	98.09	87.91	**84.95**	83.78
Swedish	99.86	99.86	95.57	99.86	95.66	93.92	94.43	92.85	95.48	81.67	**77.58**	76.73
Swedish-LinES	99.97	99.97	86.43	99.97	94.26	*91.27*	99.60	*90.04*	*98.53*	80.14	**75.57**	74.29
Turkish	99.85	97.92	96.89	97.92	91.51	90.58	86.70	84.60	89.60	60.78	**53.78**	53.19
Ukrainian	99.66	99.66	94.84	99.66	87.33	70.77	71.00	69.74	86.64	69.28	**61.09**	60.76
Urdu	100.00	100.00	98.32	100.00	92.13	89.93	80.31	76.03	93.04	83.86	**77.09**	76.69
Uyghur	99.94	99.94	65.31	99.94	76.09	79.04	99.94	75.57	99.94	53.49	33.21	**34.18**
Vietnamese	84.26	84.26	92.87	84.26	75.29	73.30	83.93	73.26	83.54	44.99	**39.97**	37.47
Arabic-PUD	80.85	90.81	98.95	90.81	70.39	0.00	22.73	0.00	0.00	54.57	**44.34**	43.14
Czech-PUD	99.28	99.28	95.40	99.28	96.57	89.92	88.33	87.69	95.37	84.50	79.67	**79.80**
German-PUD	97.90	97.94	90.75	97.94	84.46	20.40	31.77	1.55	3.10	73.75	66.05	**66.53**
English-PUD	99.74	99.74	95.57	99.74	94.11	92.99	94.19	90.13	95.47	82.80	**79.21**	78.95
Spanish-PUD	99.48	99.43	94.14	99.43	88.17	1.76	54.21	0.00	3.43	84.96	**77.99**	77.65
Finnish-PUD	99.63	99.63	92.20	99.63	95.84	0.00	93.75	0.00	86.50	83.89	**80.86**	78.65
French-PUD	99.81	98.86	93.33	98.86	88.00	2.39	58.65	0.00	4.79	79.64	**74.19**	73.63
Hindi-PUD	98.78	98.78	93.26	98.78	84.69	33.09	18.11	4.80	0.00	65.56	**52.53**	50.85
Italian-PUD	99.64	99.22	94.11	99.22	93.10	*2.47*	57.26	*2.47*	*95.52*	87.39	**84.03**	83.70
Japanese-PUD	92.41	92.41	95.04	92.41	90.02	7.65	53.75	7.07	91.39	79.26	**78.36**	76.28
Portuguese-PUD	99.27	99.39	95.94	99.39	88.45	0.00	59.22	0.00	12.57	80.32	**74.43**	73.96
Russian-PUD	97.25	97.25	98.51	97.25	85.86	78.82	38.20	34.28	0.00	76.69	**69.37**	68.31
Swedish-PUD	98.35	98.35	94.44	98.35	91.16	88.07	74.58	73.09	84.55	75.43	**70.88**	70.62
Turkish-PUD	99.13	96.93	90.87	96.93	71.38	0.00	23.67	0.00	0.09	53.58	34.12	**34.53**
Buryat (surprise)	99.35	99.35	91.81	99.35	84.12	99.35	81.65	78.08	81.40	41.64	21.58	**31.50**
Kurmanji (surprise)	99.01	98.85	97.02	98.85	90.04	89.84	81.61	80.62	89.76	46.33	**32.89**	32.35
North Sámi (surprise)	99.88	99.88	98.79	99.88	86.81	88.98	81.93	77.76	81.86	45.53	**33.62**	30.60
Upper Sorbian (surprise)	99.84	99.84	90.69	99.84	90.30	99.84	74.20	72.43	87.70	63.34	**55.76**	53.83
Average Score	98.89	98.63	88.68	98.63	91.22	*79.48*	82.50	*73.47*	*82.64*	75.39	**69.52**	68.35

Table 2: Full results of *UDPipe 1.2 Participant System* and LAS F1-score of *UDPipe 1.1 Baseline System* for reference. The results in italic are not part of the official results and were generated using post-competition models due to time constraints.

Treebank	Enlarged training data using other treebanks							Original training data only						
	UPOS	XPOS	Feats	AllTags	Lemmas	UAS	LAS	UPOS	XPOS	Feats	AllTags	Lemmas	UAS	LAS
Ancient Greek	**85.55**	*43.69*	73.30	*43.67*	***82.89***	**65.37**	57.39	82.37	**72.33**	**85.82**	**72.32**	82.63	64.05	**57.44**
Ancient Greek-PROIEL	95.60	*93.34*	87.66	*84.85*	***92.73***	**71.72**	**66.51**	**95.74**	**95.94**	**88.49**	**87.04**	92.66	71.29	66.49
Czech-CAC	**98.34**	***91.92***	**90.53**	***90.36***	***97.31***	**88.21**	**84.40**	98.17	90.64	89.43	88.51	97.04	86.17	81.88
Czech-CLTT	95.49	***88.07***	86.14	*85.04*	***96.79***	**80.52**	**76.69**	**96.28**	86.86	**87.02**	**86.75**	95.56	78.66	74.67
English-LinES	94.87	*92.01*	99.39	*90.41*	*98.34*	79.06	74.92	**94.94**	**92.56**	**99.92**	**90.87**	**99.92**	**79.30**	**75.20**
English-ParTUT	**93.41**	*91.92*	91.45	*89.83*	*96.39*	**81.13**	**76.89**	93.08	**92.85**	**92.23**	**90.84**	**96.50**	79.86	75.31
French-ParTUT	**95.38**	*85.35*	91.23	*82.06*	***94.87***	**84.03**	**80.17**	94.48	**94.23**	**91.89**	**90.75**	94.29	83.24	79.07
Italian	**97.31**	***97.06***	**97.20**	***96.26***	***97.34***	**88.62**	**86.11**	97.22	97.04	97.00	96.14	97.28	88.53	85.72
Latin-ITTB	96.83	*91.58*	93.50	*89.71*	*97.61*	79.74	75.84	**97.15**	**92.64**	**93.51**	**91.24**	**97.73**	**80.03**	**76.26**
Slovenian-SST	**91.78**	***86.40***	**85.32**	***82.33***	***93.79***	**59.26**	**53.94**	88.90	81.59	81.77	79.12	91.39	53.60	47.50
Swedish-LinES	94.26	*91.27*	99.60	*90.04*	*98.53*	80.14	**75.57**	**94.33**	**91.76**	**99.97**	**90.56**	**99.97**	**80.25**	75.45
Italian-PUD	93.10	***2.47***	**57.26**	***2.47***	*95.52*	**87.39**	**84.03**	**93.18**	**2.47**	57.19	**2.47**	**95.57**	86.87	83.61

Table 3: The effect of additional training data from other treebanks of the same language in *UDPipe 1.2 Participant System*.

Treebank	GRU-based segmentation followed by parsing			Joint segmentation and parsing		
	Sents	UAS	LAS	Sents	UAS	LAS
Gothic	**32.46**	69.04	62.23	24.12	**69.26**	**62.80**
Latin-PROIEL	**30.37**	66.11	60.63	19.56	**66.45**	**61.55**
Slovenian-SST	**17.76**	57.93	51.95	13.13	**59.26**	**53.94**

Table 5: Joint segmentation and parsing in *UDPipe 1.2 Participant System*, optimized to maximize parsing likelihood, in comparison with sequential segmentation and parsing.

Beam size	UAS	LAS
1	74.36	68.46
5	75.33	69.45
10	75.39	69.51
15	75.41	69.53
20	**75.42**	**69.54**
Best on development data for each treebank	75.39	69.52

Table 7: *UDPipe 1.2 Participant System* parsing scores with various beam sizes.

In order to make the extensive results more visual, we show relative difference of baseline LAS score using the grey bars (on a scale that ignores 3 outliers). We use this visualization also in later tables, always showing relative difference to the first occurrence of the metric in question.

The effect of enlarging training data using other treebanks of the same language (Section 4.2) is evaluated in Table 3. We include only those treebanks in which the enlarged training data result in better LAS score and compare the performance to cases in which only the original training data is used.

The impact of tokenizer dimension 64 compared to dimension 24 can be found in Table 4. We also include the effect of not using the suffix rules for multi-word token splitting, and not using multi-word token splitting at all. As expected, for many languages the dimension 64 does not change the results, but yields superior performance for languages with either difficult tokenization or sentence segmentation.

The improvement resulting from joint sentence segmentation and parsing is evaluated in Table 5. While the LAS and UAS F1-scores of the joint approach improves, the sentence segmentation F1-score deteriorates significantly.

The overall effect of search-based oracle with various transition systems on parsing accuracy is summarized in Table 6. The search-based oracle improves results in all cases, but the increase is only slight if a dynamic oracle is also used. Note however that dynamic oracles for non-projective systems are usually either very inefficient (for link2, only $\mathcal{O}(n^8)$ dynamic oracle is proposed in Gómez-Rodríguez et al. (2014)) or not known (as is the case for the swap system).

Furthermore, if only a static oracle is used, partially or fully non-projective systems yield better overall performance than a projective one. Yet, a dynamic oracle improves performance of the projective system to the extent it yield better results (which is further improved by utilizing also a search-based oracle).

The influence of beam size on UAS and LAS scores is analyzed in Table 7. According to the results, tuning beam size for every treebank independently is worse than using large beam size all the time.

Finally, model size and runtime performance of individual UDPipe components are outlined in Table 8. The median of complete model size is circa 13MB and the speed of full processing (tokenization, tagging and parsing with beam size 5) is approximately 1700 words per second on a single core of an Intel Xeon E5-2630 2.4GHz processor.

Treebank	UDPipe 1.2 Participant System				Tokenizer dim 24			No suffix rules		No token splitting	
	Tokens	Words	Sents	LAS	Words	Sents	LAS	Words	LAS	Words	LAS
Ancient Greek	99.96	**99.96**	98.73	57.39	**99.96**	**98.85**	**57.42**	**99.96**	57.39	**99.96**	57.39
Ancient Greek-PROIEL	100.00	**100.00**	**47.09**	**66.51**	**100.00**	45.14	65.79	**100.00**	**66.51**	**100.00**	**66.51**
Arabic	99.98	**93.71**	**81.77**	66.06	**93.71**	80.89	**66.08**	92.89	65.13	78.39	45.84
Basque	99.96	**99.96**	**99.50**	**70.45**	**99.96**	99.08	70.39	**99.96**	**70.45**	**99.96**	**70.45**
Bulgarian	99.92	**99.92**	**92.85**	**84.92**	99.91	92.54	84.87	**99.92**	**84.92**	**99.92**	**84.92**
Catalan	99.97	**99.97**	**99.03**	**85.53**	99.96	**99.03**	85.52	99.77	85.17	99.67	84.93
Chinese	89.55	**89.55**	98.20	**57.89**	89.25	**98.50**	57.63	**89.55**	**57.89**	**89.55**	**57.89**
Croatian	99.90	99.90	95.56	77.73	**99.92**	**96.98**	**77.83**	99.90	77.73	99.90	77.73
Czech	99.93	**99.93**	**92.30**	**83.19**	99.92	91.82	83.16	**99.93**	**83.19**	99.80	82.96
Czech-CAC	99.97	**99.96**	**100.00**	**84.40**	**99.96**	99.76	**84.40**	**99.96**	**84.40**	99.72	84.03
Czech-CLTT	99.34	99.34	94.19	76.69	**99.52**	**96.49**	**77.30**	99.34	76.69	99.31	76.65
Danish	99.60	**99.60**	78.97	75.28	99.58	**80.07**	**75.43**	**99.60**	75.28	**99.60**	75.28
Dutch	99.80	99.80	76.95	**70.52**	**99.84**	**77.62**	70.09	99.80	**70.52**	99.80	**70.52**
Dutch-LassySmall	99.99	**99.99**	**81.83**	**80.15**	99.97	74.84	79.18	**99.99**	**80.15**	**99.99**	**80.15**
English	99.03	**99.03**	75.33	**77.25**	98.97	**75.67**	77.24	**99.03**	**77.25**	**99.03**	**77.25**
English-LinES	99.92	**99.92**	**87.40**	74.92	99.90	86.59	**74.96**	**99.92**	74.92	**99.92**	74.92
English-ParTUT	99.57	99.55	**98.40**	76.89	**99.61**	97.19	**77.04**	99.54	76.86	99.45	76.75
Estonian	99.89	**99.89**	93.66	**60.01**	99.88	**93.75**	59.99	**99.89**	**60.01**	**99.89**	**60.01**
Finnish	99.69	**99.69**	**86.75**	**77.26**	**99.69**	84.70	77.11	**99.69**	**77.26**	**99.69**	**77.26**
Finnish-FTB	99.97	**99.96**	**85.54**	**75.31**	99.94	84.72	75.03	99.95	75.28	99.74	75.08
French	99.76	98.88	**94.58**	**80.50**	**98.89**	94.09	80.41	98.88	**80.50**	95.54	74.98
French-ParTUT	99.85	**98.97**	**97.76**	**80.17**	98.88	97.38	80.06	**98.97**	**80.17**	95.00	74.43
French-Sequoia	99.76	**99.06**	**84.60**	**81.35**	99.04	84.00	81.34	**99.06**	**81.35**	95.07	74.74
Galician	99.93	99.93	**96.18**	77.73	**99.94**	95.98	**77.81**	99.93	77.73	99.93	77.73
Galician-TreeGal	99.62	98.66	85.35	66.31	**98.70**	**86.69**	**66.32**	98.09	65.48	87.58	48.87
German	99.67	99.67	**79.35**	**68.61**	**99.68**	79.34	68.41	99.67	**68.61**	97.17	64.81
Gothic	100.00	**100.00**	**24.12**	**62.80**	**100.00**	20.75	62.08	**100.00**	**62.80**	**100.00**	**62.80**
Greek	99.87	**99.87**	90.00	**80.67**	**99.87**	**90.44**	80.54	**99.87**	**80.67**	**99.87**	**80.67**
Hebrew	99.98	**85.16**	**100.00**	**57.86**	85.12	99.59	57.83	81.73	54.60	57.12	26.11
Hindi	100.00	**100.00**	**99.20**	**87.28**	**100.00**	**99.20**	**87.28**	**100.00**	**87.28**	**100.00**	**87.28**
Hungarian	99.81	**99.81**	95.54	66.54	**99.81**	**95.58**	**66.63**	**99.81**	66.54	**99.81**	66.54
Indonesian	100.00	**100.00**	**91.73**	75.47	**100.00**	90.71	**75.48**	**100.00**	75.47	**100.00**	75.47
Irish	99.40	99.40	**94.78**	62.87	**99.56**	94.14	**63.00**	99.40	62.87	99.40	62.87
Italian	99.91	**99.83**	**97.11**	**86.11**	99.78	96.91	85.97	99.58	85.53	88.92	68.98
Japanese	90.97	**90.97**	**95.01**	**74.49**	90.02	**95.01**	73.19	**90.97**	**74.49**	**90.97**	**74.49**
Kazakh	96.07	**95.63**	81.23	25.43	92.74	**81.56**	24.39	95.36	**25.95**	95.36	**25.95**
Korean	99.69	**99.69**	**92.41**	**60.30**	99.67	92.04	60.08	**99.69**	**60.30**	**99.69**	**60.30**
Latin	99.99	99.99	**98.56**	**47.02**	**100.00**	98.35	46.96	99.99	**47.02**	99.99	**47.02**
Latin-ITTB	99.89	99.89	**82.58**	75.84	**99.94**	82.49	**75.91**	99.89	75.84	99.89	75.84
Latin-PROIEL	100.00	**100.00**	**19.56**	**61.55**	**100.00**	18.43	**61.55**	**100.00**	**61.55**	**100.00**	**61.55**
Latvian	98.94	**98.94**	98.32	61.80	98.89	**98.37**	**61.81**	**98.94**	61.80	**98.94**	61.80
Norwegian-Bokmaal	99.79	**99.79**	**96.38**	**83.89**	99.78	95.79	83.86	**99.79**	**83.89**	**99.79**	**83.89**
Norwegian-Nynorsk	99.93	**99.93**	**92.08**	**82.74**	**99.93**	92.03	82.68	**99.93**	**82.74**	**99.93**	**82.74**
Old Church Slavonic	99.99	99.99	**40.94**	**66.29**	**100.00**	39.14	66.15	99.99	**66.29**	99.99	**66.29**
Persian	100.00	**99.65**	97.76	**80.33**	**99.65**	**98.74**	80.30	99.48	80.06	99.08	79.42
Polish	99.98	99.87	**99.18**	**80.21**	**99.88**	99.00	80.20	99.09	77.98	98.60	76.67
Portuguese	99.66	99.54	**89.24**	**82.72**	**99.55**	88.75	82.63	99.29	82.11	88.36	64.71
Portuguese-BR	99.96	**99.86**	96.71	85.97	99.85	**96.80**	**85.98**	**99.86**	85.97	89.41	67.97
Romanian	99.67	**99.67**	93.72	**80.32**	99.62	**93.85**	80.29	**99.67**	**80.32**	**99.67**	**80.32**
Russian	99.90	99.90	**96.59**	**76.15**	**99.91**	96.48	76.11	99.90	**76.15**	99.90	**76.15**
Russian-SynTagRus	99.58	**99.58**	**97.97**	**86.80**	99.50	97.72	86.70	**99.58**	**86.80**	**99.58**	**86.80**
Slovak	100.00	**100.00**	**84.26**	**75.63**	99.99	83.14	75.43	**100.00**	**75.63**	**100.00**	**75.63**
Slovenian	99.96	**99.96**	**98.86**	**81.84**	99.93	98.85	81.74	**99.96**	**81.84**	**99.96**	**81.84**
Slovenian-SST	99.87	99.87	13.13	**53.94**	**99.97**	**15.38**	53.86	99.87	**53.94**	99.87	**53.94**
Spanish	99.91	**99.74**	**95.26**	**81.95**	99.70	94.89	81.92	99.41	81.32	96.55	77.75
Spanish-AnCora	99.97	**99.95**	**98.26**	**84.95**	**99.95**	98.15	**84.95**	99.73	84.51	99.45	83.74
Swedish	99.86	**99.86**	**95.57**	**77.58**	99.78	93.17	77.30	**99.86**	**77.58**	**99.86**	**77.58**
Swedish-LinES	99.97	**99.97**	**86.43**	**75.57**	99.96	85.73	75.44	**99.97**	**75.57**	**99.97**	**75.57**
Turkish	99.85	**97.92**	96.89	**53.78**	**97.92**	**97.09**	53.73	97.28	52.58	96.04	50.88
Ukrainian	99.66	99.66	94.84	61.09	**99.77**	**94.89**	**61.21**	99.66	61.09	99.66	61.09
Urdu	100.00	**100.00**	98.32	77.09	**100.00**	**98.60**	**77.11**	**100.00**	77.09	**100.00**	77.09
Uyghur	99.94	**99.94**	65.31	**33.21**	99.85	**67.23**	33.18	**99.94**	**33.21**	**99.94**	**33.21**
Vietnamese	84.26	**84.26**	**92.87**	**39.97**	82.57	92.26	38.45	**84.26**	**39.97**	**84.26**	**39.97**
Arabic-PUD	80.85	90.81	98.95	44.34	**90.87**	**99.10**	**44.37**	89.91	43.92	80.85	36.61
Czech-PUD	99.28	99.28	95.40	79.67	**99.29**	**96.29**	**79.74**	99.28	79.67	99.13	79.46
German-PUD	97.90	**97.94**	**90.75**	**66.05**	97.83	86.58	65.43	**97.94**	**66.05**	95.58	62.62
English-PUD	99.74	**99.74**	95.57	79.21	99.66	**97.22**	**79.34**	**99.74**	79.21	**99.74**	79.21
Spanish-PUD	99.48	99.43	94.14	77.99	**99.50**	**94.36**	**78.03**	99.30	77.78	96.46	74.94
Finnish-PUD	99.63	**99.63**	**92.20**	80.86	99.60	91.87	**80.92**	**99.63**	80.86	**99.63**	80.86
French-PUD	99.81	98.86	93.33	74.19	**98.88**	**96.38**	**74.31**	98.86	74.19	96.15	69.79
Hindi-PUD	98.78	98.78	**93.26**	52.53	**98.84**	90.92	**52.54**	98.78	52.53	98.78	52.53
Italian-PUD	99.64	99.22	94.11	**84.03**	**99.25**	**94.40**	83.95	99.05	83.72	89.56	68.04
Japanese-PUD	92.41	**92.41**	**95.04**	**78.36**	90.95	**95.04**	76.00	**92.41**	**78.36**	**92.41**	**78.36**
Portuguese-PUD	99.27	99.39	**95.94**	**74.43**	**99.44**	95.21	74.37	99.24	74.18	89.49	60.18
Russian-PUD	97.25	97.25	98.51	69.37	**97.38**	**99.10**	**69.57**	97.25	69.37	97.25	69.37
Swedish-PUD	98.35	98.35	94.44	**70.88**	**98.41**	**94.47**	**70.88**	98.35	**70.88**	98.35	**70.88**
Turkish-PUD	99.13	**96.94**	90.87	**34.12**	96.31	**92.20**	34.02	96.02	32.56	95.99	32.15

Table 4: Impact of tokenizer dimension 64 versus 24, no suffix rules for multi-word token splitting, and no multi-word token splitting at all in the *UDPipe 1.2 Participant System*.

Transition system and oracle	No search-based oracle		Search-based oracle	
	UAS	LAS	UAS	LAS
Arc standard system with static oracle	74.29	68.27	74.80	68.87
Arc standard system with dynamic oracle	75.31	69.36	**75.40**	69.51
Swap system with static lazy oracle	74.73	68.76	75.16	69.27
Link2 system with static oracle	74.79	68.76	75.21	69.29
Any system, static oracle	74.72	68.71	75.21	69.31
Any system, any oracle	75.27	69.31	75.38	**69.52**

Table 6: The overall effect of search-based oracle on various transition systems.

Model configuration	Model size [MB]	Model speed [kwords/s]
Tokenizer dim 24	0.04 (0.03–0.15)	27.7 (20–37)
Tokenizer dim 64	0.20 (0.19–0.31)	6.0 (4.9–8.6)
Tagger&lemmatizer	9.4 (2.3–24.8)	6.5 (2.1–14)
Parser beam size 1	3.2 (1.9–6.9)	14.9 (12–19)
Parser beam size 5		2.7 (2.2–3.6)
Complete model	13.2 (4.4–31.9)	1.7 (1.2–2.3)

Table 8: *UDPipe 1.2 Participant System* model size and runtime performance, displayed as a median for all the treebanks, together with the 5^{th} and 95^{th} percentile. The complete model consists of a tokenizer with character embedding and GRU cell dimension 64, a tagger, a lemmatizer and a parser with beam size 5.

6 Conclusions and Future Work

We described our contributions to *CoNLL 2017 UD Shared Task*: *UDPipe 1.1 Baseline System* and *UDPipe 1.2 Participant System*. Both these systems and the pretrained models are available at `http://ufal.mff.cuni.cz/udpipe` under open-source Mozilla Public Licence (MPL). Binary tools as well as bindings for C++, Python, Perl, Java and C# are provided.

As our future work, we consider using deeper models in UDPipe for tokenizers, POS taggers and especially for the parser.

Acknowledgments

This work has been partially supported and has been using language resources and tools developed, stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2015071). This research was also partially supported by OP VVV projects CZ.02.1.01/0.0/0.0/16_013/0001781 and CZ.02.2.69/0.0/0.0/16_018/0002373, and by SVV project number 260 453.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globaly normalized transition-based neural networks. In *Association for Computational Linguistic*. http://arxiv.org/abs/1603.06042.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition.

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)*. Association for Computational Linguistics, Doha, Qatar, pages 740–750. http://www.aclweb.org/anthology/D14-1082.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR* abs/1409.1259. http://arxiv.org/abs/1409.1259.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research* 12:2493–2537.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M. Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*. pages 1370–1380. http://aclweb.org/anthology/P/P14/P14-1129.pdf.

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

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. http://www.aclweb.org/anthology/P15-1033.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. 2014. A tabular method for dynamic oracles in transition-based parsing. *TACL* 2:119–130. http://www.aclweb.org/anthology/Q14-1010.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, pages 917–927. http://www.aclweb.org/anthology/D14-1099.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2016. What do recurrent neural network grammars learn about syntax? *CoRR* abs/1611.05774. http://arxiv.org/abs/1611.05774.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W. Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *CoRR* abs/1508.02096. http://arxiv.org/abs/1508.02096.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*. pages 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.* 34(4):513–553. https://doi.org/10.1162/coli.07-056-R1-07-027.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '09, pages 351–359. http://dl.acm.org/citation.cfm?id=1687878.1687929.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017a. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague. http://hdl.handle.net/11234/1-1983.

Joakim Nivre, Željko Agić, Lars Ahrenberg, et al. 2017b. Universal dependencies 2.0 – CoNLL 2017 shared task development and test data. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-2184.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portorož, Slovenia, pages 1659–1666.

Martin Potthast, Tim Gollub, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, and Benno Stein. 2014. Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In Evangelos Kanoulas, Mihai Lupu, Paul Clough, Mark Sanderson, Mark Hall, Allan Hanbury, and Elaine Toms, editors, *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*. Springer, Berlin Heidelberg New York, pages 268–299. https://doi.org/10.1007/978-3-319-11382-1_22.

Milan Straka. 2017. CoNLL 2017 shared task - UDPipe baseline models and supplementary materials. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. http://hdl.handle.net/11234/1-1990.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*. European Language Resources Association, Portorož, Slovenia.

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)*.

Jana Straková, Milan Straka, and Jan Hajič. 2014. Open-source tools for morphology, lemmatization, pos tagging and named entity recognition. In *Proceedings of 52nd Annual Meeting of the*

Association for Computational Linguistics: System Demonstrations. Association for Computational Linguistics, Baltimore, Maryland, pages 13–18. http://www.aclweb.org/anthology/P14-5003.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR* abs/1706.03762. http://arxiv.org/abs/1706.03762.

Zhilin Yang, Ruslan Salakhutdinov, and William W. Cohen. 2016. Multi-task cross-lingual sequence tagging from scratch. *CoRR* abs/1603.06270. http://arxiv.org/abs/1603.06270.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gökırmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Hěctor Martínez Alonso, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadova, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Association for Computational Linguistics.