Abstract

UDPipe is a trainable pipeline which performs sentence segmentation, tokenization, POS tagging, lemmatization and dependency parsing (Straka et al., 2016). We present a prototype for UDPipe 2.0 and evaluate it in the CoNLL 2018 UD Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, which employs three metrics for submission ranking. Out of 26 participants, the prototype placed first in the MLAS ranking, third in the LAS ranking and third in the BLEX ranking. In extrinsic parser evaluation EPE 2018, the system ranked first in the overall score.

The prototype utilizes an artificial neural network with a single joint model for POS tagging, lemmatization and dependency parsing, and is trained only using the CoNLL-U training data and pretrained word embeddings, contrary to both systems surpassing the prototype in the LAS and BLEX ranking in the shared task.


After the shared task, we slightly refined the model architecture, resulting in better performance both in the intrinsic evaluation (corresponding to first, second and second rank in MLAS, LAS and BLEX shared task metrics) and the extrinsic evaluation. The improved models will be available shortly in UDPipe at http://ufal.mff.cuni.cz/udpipe.

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 2.2 (Nivre et al., 2018) consists of 122 dependency treebanks in 71 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 2018 Shared Tasks: Multilingual Parsing from Raw Text to Universal Dependencies (CoNLL 2018 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. (2018). The current shared task is a reiteration of previous year’s CoNLL 2017 UD Shared Task (Zeman et al., 2017).

This paper describes our contribution to CoNLL 2018 UD Shared Task, a prototype of UDPipe 2.0. UDPipe (Straka et al., 2016)\(^1\) is an open-source tool which automatically generates sentence segmentation, tokenization, POS tagging, lemmatization and dependency trees, using UD treebanks as training data. The current version UDPipe 1.2 (Straka and Straková, 2017) is used as a baseline in CoNLL 2018 UD Shared Task. UDPipe 1.2 achieves low running times and moderately sized models, however, its performance is behind the current state-of-the-art, placing 13th, 17th and 18th in the three metrics (MLAS, LAS and BLEX, respectively). As a participation system in the shared task, we therefore propose a prototype for UDPipe 2.0, with the goal of reaching state-of-the-art performance.

\(^1\)http://ufal.mff.cuni.cz/udpipe
The contributions of this paper are:

- Description of UDPipe 2.0 prototype, which placed 1st in MLAS, 3rd in LAS and 3rd in BLEX, the three metrics of CoNLL 2018 UD Shared Task. In extrinsic parser evaluation EPE 2018, the prototype ranked first in overall score.

The prototype employs an artificial neural network with a single joint model for POS tagging, lemmatization, and parsing. It utilizes solely CoNLL-U training data and word embeddings, and does not require treebank-specific hyperparameter tuning.

- Runtime performance measurements of the prototype, using both CPU-only and GPU environments.

- Ablation experiments showing the effect of word embeddings, regularization techniques and various joint model architectures.

- Post-shared-task model refinement, improving both the intrinsic evaluation (corresponding to 1st, 2nd and 2nd rank in MLAS, LAS and BLEX shared task metrics) and the extrinsic evaluation. The improved models will be available soon in UDPipe.²

²http://ufal.mff.cuni.cz/udpipe

3 Model Overview

The objective of the shared task is to parse raw texts. In accordance with the CoNLL-U format, the participant systems are required to:

- tokenize the given text and segment it into sentences;
- split multi-word tokens into individual words (CoNLL-U format distinguishes between the surface tokens, e.g., won’t, and words, e.g., will and not);
- perform POS tagging, producing UPOS (universal POS) tags, XPOS (language-specific POS) tags and UFeats (universal morphological features);
- perform lemmatization;
- finally perform dependency parsing, including universal dependency relation labels.

We decided to reuse the tokenization, sentence segmentation and multi-word token splitting available in UDPipe 1.2, i.e., the baseline solution, and focus on POS tagging, lemmatization, and parsing, utilizing a deep neural network architecture.

For practical reasons, we decided to devise a joint model for POS tagging, lemmatization, and parsing, with the goal of sharing at least the trained word embeddings, which are usually the largest part of a trained neural network model.

For POS tagging, we applied a straightforward model in the lines of Ling et al. (2015) – first representing each word with its embedding, contextualizing them with bidirectional RNNs (Graves 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. The neural graph-based parser of Dozat et al. (2017) won the last year’s CoNLL 2017 UD Shared Task by a wide margin.
Regarding the dependency parsing, we reimple-
mented a biaffine attention parser of (Dozat et al.,
2017), which won the previous year’s shared task.
The parser also processes contextualized embed-
dings, followed by additional attention and classi-
fication layers. We considered two levels of shar-
ing:
- **loosely joint model**, where only the word em-
beddings are shared;
- **tightly joint model**, where the contextualized
embeddings are shared by the tagger and the
parser.

4 Model Implementation

We now describe each model component in a
greater detail.

4.1 Tokenization and Sentence Segmentation

We perform tokenization, sentence segmentation
and multi-word token splitting with the baseline
UDPipe 1.2 approach. In a nutshell, input charac-
ters are first embedded using trained embeddings,
then fixed size input segments (of 50 characters)
are processed by a bidirectional GRU (Cho et al.,
2014), and each character is classified into three
classes – a) there is a sentence break after this
character, b) there is a token break after this char-
acter, and c) there is no break after this character.
For detailed description, see Straka and Straková
(2017).

We only slightly modified the baseline models
in the following way: in addition to the segments
of size 50 we also consider longer segments of
200 characters during training (and choose the best
model for each language according to the develop-
ment set performance). Longer segments improve
sentence segmentation performance for treebanks
with nontrivial sentence breaks – such sentence
breaks are caused either by the fact that a treebank
does not contain punctuation, or that semantic sen-
tence breaks (e.g., end of heading and start of a
text paragraph) are not annotated in the treebank.
The evaluation of longer segments models is pre-
sented later in Section 6.1.

4.2 Embedding Input Words

We represent each input word using three kinds of
embeddings, as illustrated in Figure 1.
- **pretrained word embeddings**: pretrained
word embeddings are computed using large
plain texts and are constant throughout the
training. We utilize either word embed-
dings provided by the CoNLL 2017 UD
Shared Task organizers (of dimension 100),
Wikipedia fastText embeddings3 (of dimen-
sion 300), or no pretrained word embeddings,
choosing the alternative resulting in highest
development accuracy. To limit the size of
the pretrained embeddings, we keep at most
1M most frequent words of fastText embed-
dings, or at most 3M most frequent words of
the shared task embeddings.
- **trained word embeddings**: trained word em-
beddings are created for every training word,
initialized randomly, and trained with the rest
of the network.
- **character-level word embeddings**: character-
level word embeddings are computed simi-
larly as in Ling et al. (2015), utilizing a bidi-
rectional GRU.

4.3 POS Tagging

We process the embedded words through a
multi-layer bidirectional LSTM (Hochreiter and
Schmidhuber, 1997) to obtain contextualized em-
beddings. In case multiple RNN layers are em-
ployed, we utilize residual connections on all but
the first layer (Wu et al., 2016).

For each of the three kinds of tags (UPOS,
XPOS and UFeats), we construct a dictionary con-
taining all unique tags from the training data.
Then, we employ a softmax classifier for each tag
kind processing contextualized embeddings and
generating a class from the corresponding tag dic-
dionary.

However, a single-layer softmax classifier has
only a limited capacity. To allow more non-linear
processing for each tag kind, we prepend a dense
layer with tanh non-linearity and a residual con-
nexion before each softmax classifier.

Figure 1: Word embeddings used in the model.

3 http://github.com/facebookresearch/fastText/
blob/master/pretrained-vectors.md
4.4 Lemmatization

We lemmatize input words by classifying them into lemma generation rules. We consider these rules as a fourth tag kind (in addition to UPOS, XPOS and UFeats) and use analogous architecture.

To construct the lemma generation rule from a given form and lemma, we proceed as follows:

- We start by finding the longest continuous substring of the form and the lemma. If it is empty, we use the lemma itself as the class.
- If there is a common substring of the form and the lemma, we compute the shortest edit script converting the prefix of the form into the prefix of the lemma, and the shortest edit script converting the suffix of the form to the suffix of the lemma.

We consider two variants of the edit scripts. The first one permits only character operations delete_current_char and insert_char(c). The second variant additionally allows copy_current_char operation. For each treebank, we choose the variant producing less unique classes for all training data.

- All above operations are performed case insensitively. To indicate correct casing of the lemma, we consider the lemma to be a concatenation of segments, where each segment is composed of either a sequence of lowercase characters, or a sequence of uppercase characters. We represent the lemma casing by encoding the beginning of every such segment, where the offsets in the first half of the lemma are computed relatively to the start of the lemma, and the offsets in the second half of the lemma are computed relatively to the end of the lemma.

### Table 1: Statistics of number of lemma generation rules for 73 treebank training sets of the CoNLL 2018 UD Shared Task.

<table>
<thead>
<tr>
<th></th>
<th>Minimum</th>
<th>Q1</th>
<th>Median</th>
<th>Mean</th>
<th>Q3</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lemma generation rules</td>
<td>6</td>
<td>199</td>
<td>490</td>
<td>877</td>
<td>831</td>
<td>8475</td>
</tr>
</tbody>
</table>

4.5 Dependency Parsing

We base our parsing model on a graph-based biaffine attention parser architecture of the last year’s shared task winner (Dozat et al., 2017).

The model starts again with contextualized embeddings produced by bidirectional RNNs, with an artificial ROOT word prepended before the beginning of the sentence. The contextualized embeddings are non-linearly mapped into arc-head and arc-dep representation, which are combined using biaffine attention to produce for each word a distribution indicating the probability of all other words being its dependency head. Finally, we produce an arborescence (i.e., directed spanning tree) with maximum probability by utilizing the Chu-Liu/Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967).

To generate labels for dependency arcs, we proceed analogously – we non-linearly map the contextualized embeddings into rel-head and rel-dep and combine them using biaffine attention, producing for every possible dependency edge a probability distribution over dependency labels.

4.6 Joint Model Variants

We consider two variants of a joint tagging and parsing model, illustrated in Figure 3.

- The tightly joint model shares the contextualized embeddings between the tagger and the parser. Notably, the shared contextualized embeddings are computed using 2 layers of bidirectional LSTM. Then, both the tagger and the parser employ an additional layer of bidirectional LSTM, resulting in 4 bidirectional RNN layers.
Figure 3: The tightly joint model (on the left) and the loosely joint model (on the right).

- The **loosely joint model** shares only the word embeddings between the tagger and the parser, which both compute contextualized embeddings using 2 layers of bidirectional LSTM, resulting again in 4 RNN layers.

There is one additional difference between the tightly and loosely joint model. While in the tightly joint model the generated POS tags influence the parser model only independently through the shared contextualized embeddings (i.e., the POS tags can be considered regularization of the parser model), the loosely joint model extends the parser word embeddings by the embeddings of the predicted UPOS, XPOS and UFeats tags. Note that we utilize the predicted tags even during training (instead of the gold ones).

### 4.7 Model Hyperparameters

Considering the 73 treebank training sets of the CoNLL 2018 UD Shared Task, we do not employ any treebank-specific hyperparameter search. Most of the hyperparameters were set according to a single Czech-PDT treebank (the largest one), and no effort has been made to adjust them to the other treebanks.

To compute the character-level word embeddings, we utilize character embeddings and GRUs with dimension of 256. The trained word embeddings and the sentence-level LSTMs have a dimension of 512. The UPOS, XPOS and UFeats embeddings, if used, have a dimension of 128. The parser arc-head, arc-dep, rel-head and rel-dep representations have dimensions of 512, 512, 128 and 128, respectively.

### 4.8 Neural Network Training

For each of the 73 treebanks with a training set we train one model, utilizing only the training treebank and pretrained word embeddings. Each model was trained using the Adam algorithm (Kingma and Ba, 2014) on a GeForce GTX 1080 GPU with a batch size of 32 randomly chosen sentences (for batch size of 64 sentences training ended with out-of-memory error for some treebanks). The training consists of 60 epochs, with the learning rate being 0.001 for the first 40 epochs and 0.0001 for the last 20 epochs. To sufficiently train smaller treebanks, each epoch consists of one pass over the training data or 300 batches, whatever is larger.

Following Dozat and Manning (2016); Vaswani et al. (2017), we modify the default value of $\beta_2$ hyperparameter of Adam, but to a different value than both of the above papers – to 0.99, which resulted in best performance on the largest treebank. We also make sure Adam algorithm does not update first and second moment estimates for embeddings not present in a batch.

We regularize the training by several ways:
- We employ dropout with dropout probability 50% on all embeddings and hidden layers, with the exception of RNN states and residual connections.
- We utilize label smoothing of 0.03 in all softmax classifications.
- With a probability of 20%, we replace trained word embedding by an embedding of an unknown word.

With the described training method and regularization techniques, the model does not seem to overfit at all, or very little. Consequently, we do not perform early stopping and always utilize the model after full 60 epochs of training.

The training took 2–4 hours for most of the treebanks, with the two largest Russian-SynTagRus and Czech-PDT taking 15 and 20 hours, respectively.

### 5 CoNLL 2018 UD Shared Task

The official CoNLL 2018 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.

The shared task contains test sets of three kinds:
- For most treebanks large training and development sets were available, in which case we trained the model on the training set and choose among the pretrained word embeddings and tightly or loosely joint model ac-
For several treebanks very small training sets and no development sets were available. In these cases we manually split 10% of the training set to act as a development set and proceed as in the above case.

Nine test treebanks contained no training data at all. For these treebanks we adopted the baseline model strategy:

For Czech-PUD, English-PUD, Finnish-PUD, Japanese-Modern, and Swedish-PUD there were other treebank variants of the same language available in the training set. Consequently, we processed these treebanks using models trained for Czech PDT, English EWT, Finnish TDT, Japanese GSD, and Swedish Talbanken, respectively.

For Breton-KEB, Faroese-OFT, Naija-NSC, and Thai-PUD, we trained a universal mixed model, by using first 200 sentences of each training set (or less in case of very small treebanks) as training data and first 20 sentences of each development treebank as development data.

5.1 Shared Task Evaluation

The official CoNLL 2018 UD Shared Task results are presented in Table 4. In addition to F1 scores, we also include rank of our submission (out of the 26 participant systems).

In the three official metrics (LAS, MLAS and BLEX) our system reached third, first and third average performance. Additionally, our system achieved best average performance in XPOS and AllTags metrics. Furthermore, the lemmatization F1 score was the second best.

Interestingly, although our system achieves highest average score in MLAS (which is a combination of dependency parsing and morphological features), it reaches only third best average LAS and fourth best average UFeats. Furthermore, the TurkuNLP participation system surpasses our system in both LAS and UFeats. We hypothesise that the high performance of our system in MLAS metric is caused by the fact that the tagger and parser models are joined, thus producing consistent annotations.

Finally, we note that the segmentation improvements outlined in Section 4.1 resulted in third average F1 score of our system.

5.2 Extrinsic Parser Evaluation

Following the First Shared Task on Extrinsic Parser Evaluation (Oepen et al., 2017), the 2018 edition of Extrinsic Parser Evaluation Initiative (EPE 2018) ran in collaboration with the CoNLL 2018 UD Shared Task. The initiative allowed to evaluate the English systems submitted to the CoNLL shared task against three EPE downstream systems – biological event extraction, negation resolution, and fine-grained opinion analysis.

The results of our system are displayed in Table 2. Even though our system ranked only 3rd, 3rd, and 7th in the downstream task F1 scores, it was the best system in the overall score (and average of the three F1 scores).

5.3 Model Size

The statistics of the model sizes is listed in Table 3. Average model size is approximately 140MB, which is more than 10 times larger than baseline UDPipe 1.2 models. Note however that we do not perform any model quantization (which should result in almost four times smaller models, following for example approach of Wu et al. (2016)), and we did not consider the model size during hyperparameter selection. Taking into account that the largest part of the models are the trained word embeddings, the model size could be reduced substantially by reducing trained word embeddings dimension.

5.4 Runtime Performance

The runtime performance of our system is presented in Table 6. Compared to the baseline UDPipe 1.2, tagging and parsing on a single CPU
Table 4: Official CoNLL 2018 UD Shared Task results of the UDPipe 2.0 prototype. For each metric we present F1 score percentage and also rank (out of 20 participating systems).
## 6 Ablation Experiments

All ablation experiment results in this section are performed using test sets of 61 so called “big treebanks”, which are treebanks with provided development data, disregarding small treebanks and test treebanks without training data.

### 6.1 Baseline Sentence Segmentation

The performance of our system using baseline tokenization and segmentation models (cf. Section 4.1) is displayed in Table 5. The effect of achieving better sentence segmentation influences parsing more than tagging, which can handle wrong sentence segmentation more gracefully.

### 6.2 Pretrained Word Embeddings

Considering that pretrained word embeddings have demonstrated effective similarity extraction from large plain text (Mikolov et al., 2013), they have a potential of substantially increasing tagging and parsing performance. To quantify their effect, we have evaluated models trained without pretrained embeddings, presenting results in Table 5. Depending on the metric, the pretrained word embeddings improve performance by 0.3-1.7 F1 points.

### 6.3 Regularization Methods

The effect of early stopping, checkpoint averaging of last 5 epochs and label smoothing is shown also in Table 5. While early stopping and checkpoint averaging have little effect on performance, early stopping demonstrate slight improvement of 0.1-0.4 F1 points.

### 6.4 Tightly vs Loosely Joint Model

The last model variants presented in Table 5 show the effect of always using either the tightly or loosely joint model for all treebanks. In present implementation, loosely joint model accomplishes better tagging accuracy, while deteriorating parsing slightly. The tightly joint model performs slightly worse during tagging and most notably
domain during lemmatization, while improving dependency parsing.

Finally, Figure 4 shows result of logarithmic regression of using tightly or loosely joint model, depending on treebank size in words. Accordingly, the loosely joint model seems to be more suited for smaller treebanks, while the tightly joint model appear to be more suited for larger treebanks.

7 Post-competition Improved Models

Motivated by the decrease in lemmatization performance of the tightly joint model architecture, we refined the architecture of the models by adding a direct connection from character-level word embeddings to the lemma classifier. Our hope was to improve lemmatization performance in the tightly joint architecture by providing the lemma classifier a direct access to the embeddings of exact word composition.

As seen in Table 5, the improved models perform considerably better, and with only minor differences between the tight and loosely joint architectures. We therefore consider only the tightly joint improved models, removing a hyperparameter choice (which joint architecture to use).

The improved models show a considerable increase of 0.68 percentage points in lemmatization performance and minor increase in other tagging scores. The parsing performance also improves by 0.87, 0.19, and 0.16 points in BLEX, MLAS and LAS F1 scores in the ablation experiments.

Encouraged by the results, we performed an evaluation of the improved models in TIRA, achieving 73.28, 61.25, and 65.53 F1 scores in LAS, MLAS and BLEX metrics, which corresponds to increases of 0.17, 0.00 and 1.04 percentage points. Such scores would rank 2\textsuperscript{nd}, 1\textsuperscript{st}, and 2\textsuperscript{nd} in the shared task evaluation. We also submitted the improved models to extrinsic evaluation EPE 2018, improving the F1 scores of the three downstream tasks listed in Table 2 by 0.87, 0.00, and 0.27 percentage points, corresponding to 1\textsuperscript{st}, 3\textsuperscript{rd}, and 4\textsuperscript{th} rank. The overall score of the original models, already the best achieved in EPE 2018, further increased by 0.38 points with the improved models.

8 Conclusions and Future Work

We described a prototype for UDPipe 2.0 and its performance in the CoNLL 2018 UD Shared Task, where it achieved 1\textsuperscript{st}, 3\textsuperscript{rd} and 3\textsuperscript{rd} in the three official metrics, MLAS, LAS and BLEX, respectively. The source code of the prototype is available at http://github.com/CoNLL-UD-2018/UDPipe-Future.

We also described a minor modification of the prototype architecture, which improves both the intrinsic and the extrinsic evaluation. These improved models will be released shortly in UDPipe at http://ufal.mff.cuni.cz/udpipe, utilizing quantization to decrease model size.

Acknowledgments

The work described herein has been supported by the City of Prague under the “OP PPR” program, project No. CZ.07.1.02/0.0/0.0/16_023/0000108, grant GAUK 578218/2018 of Charles University, and it has been using language resources developed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2015071).

I would like to thank Jana Straková for considerably improving the quality of the paper.

References


Figure 4: Logarithmic regression of using tightly joint models (+1) and loosely joint models (-1) depending on treebank size in words.


