# IBM Research at the CoNLL 2018 Shared Task on Multilingual Parsing

**Hui Wan   Tahira Naseem   Young-Suk Lee   Vittorio Castelli   Miguel Ballesteros**
IBM Research AI
1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA
`hwan,tnaseem,ysuklee,vittorio@us.ibm.com`
`miguel.ballesteros@ibm.com`

## Abstract

This paper presents the IBM Research AI submission to the CoNLL 2018 Shared Task on Parsing Universal Dependencies. Our system implements a new joint transition-based parser, based on the Stack-LSTM framework and the Arc-Standard algorithm, that handles tokenization, part-of-speech tagging, morphological tagging and dependency parsing in one single model. By leveraging a combination of character-based modeling of words and recursive composition of partially built linguistic structures we qualified 13th overall and 7th in low resource. We also present a new sentence segmentation neural architecture based on Stack-LSTMs that was the 4th best overall.

## 1 Introduction

The CoNLL 2018 Shared Task on Parsing Universal dependencies consists of parsing raw text from different sources and domains into Universal Dependencies (Nivre et al., 2016, 2017a) for more than 60 languages and domains.[1] The task includes extremely low resource languages, like Kurmanji or Buriat, and high-resource languages like English or Spanish. The competition therefore invites to learn how to make parsers for low-resource language better by exploiting resources available for the high-resource languages. The task also includes languages from almost all language families, including Creole languages like Nigerian Pidgin[2] and completely different scripts (i.e. Chinese, Latin alphabet, Cyrillic alphabet, or Arabic). For further description of the task, data, framework and evaluation please refer to (Nivre et al., 2018, 2017b; Zeman et al., 2018; Potthast et al., 2014; Nivre and Fang, 2017).

In this paper we describe the IBM Research AI submission to the Shared Task on Parsing Universal Dependencies. Our starting point is the Stack-LSTM[3] parser (Dyer et al., 2015; Ballesteros et al., 2017) with character-based word representations (Ballesteros et al., 2015), which we extend to handle tokenization, POS tagging and morphological tagging. Additionally, we apply the ideas presented by Ammar et al. (2016) to all low resource languages since they benefited from high-resource languages in the same family. Finally, we also present two different ensemble algorithms that boosted our results (see Section 2.4).

Participants are requested to obtain parses from raw texts. This means that, sentence segmentation, tokenization, POS tagging and morphological tagging need to be done besides parsing. Participants can choose to use the baseline pipeline (UDPipe 1.2 (Straka et al., 2016)) for those steps besides parsing, or create their own implementation. We choose to use our own implementation for most of the languages. However, in a few treebanks with very hard tokenization, like Chinese and Japanese, we rely on UDPipe 1.2 and a run of our base parser (section 2.1), since this produces better results.

For the rest of languages, we produce parses from raw text that may be in documents (and thus we need to find the sentence markers within those documents); for some of the treebanks we adapted Ballesteros and Wanner (2016) punctuation prediction system (which is also based in the Stack-LSTM framework) to predict sentence markers. Given that the text to be segmented into sentences

[1]This is the second run of the task, please refer to (Zeman et al., 2017) for the 2017 Shared Task.

[2]`https://en.wikipedia.org/wiki/Nigerian_Pidgin`

[3]We use the dynamic neural network library Dynet - `http://dynet.io/` - (Neubig et al., 2017) to implement our parser.

*Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 92–102
Brussels, Belgium, October 31 – November 1, 2018. ©2018 Association for Computational Linguistics
https://doi.org/10.18653/v1/K18-2009

can be of a significant length, we implemented a sliding-window extension of the punctuation prediction system where the Stack-LSTM is reinitialized and primed when the window is advanced (see Section 3 for details).

Our system ranked 13th overall, 7th for low resource languages and 4th in sentence segmentation. It was also the best qualifying system in low resource language, Kurmanji, evidencing the effectiveness of our adaptation of Ammar et al. (2016) approach (see Section 2.3).

## 2 Our Parser

In this Section we present our base parser (see Section 2.1), our joint architecture (see Section 2.2) and our cross-lingual approach (see Section 2.3).

### 2.1 Stack-LSTM Parser

Our base model is the Stack-LSTM parser (Dyer et al., 2015; Ballesteros et al., 2017) with character-based word representations (Ballesteros et al., 2015). This parser implements the Arc-Standard with SWAP parsing algorithm (Nivre, 2004, 2009) and it uses Stack-LSTMs to model three data structures: a buffer B initialized with the sequence of words to be parsed, a stack S containing partially built parses, and a list A of actions previously taken by the parser. This parser expects tokenized input and a unique POS tag associated with every token.

We use Ballesteros et al. (2015) version of the parser which means that we compute character-based word vectors using bidirectional LSTMs (Graves and Schmidhuber, 2005); but, in addition, we also add pretrained word embeddings for all languages. The intention is to improve in morphologically rich languages and compensate for the rest of languages in which modeling characters is not that important.

### 2.2 Joint tokenization, tagging and dependency parsing

Inspired by joint models like the ones by Bohnet et al. (2013), Zhang and Clark (2008), Rasooli and Tetreault (2013); Alberti et al. (2015); Swayamdipta et al. (2016), among others, we extend the transition-based parser presented in 2.1 with extra actions that handle tokenization, UPOS tagging and morphological tagging.

**Actions:** Actions RIGHT-ARC$(r)$ , LEFT-ARC$(r)$ and SWAP remain unchanged, where $r$ represents the label assigned to the arc. The following actions are modified or added for the joint transition-based system.

1. SHIFT is extended to SHIFT$(p, f)$ in which $p$ is the UPOS tag assigned to the token being shifted, $f$ is the Morphological tag. This is the same as in (Bohnet et al., 2013).

2. A new action TOKENIZE$(i)$ is added to handle tokenization within the sentence. TOKENIZE$(i)$ tokenizes the string at the top of the buffer at offset $i$. The resulted two tokens are put at the top of the buffer. When a string needs to be tokenized into more than two tokens, a series of TOKENIZE and SHIFT actions will do the work.

3. A new action SPLIT is added to handle splitting of a string which is more complicated than inserting whitespace, for example, the word "des" in French is splitted into "de" and "les", as shown in Figure 2. SPLIT splits the top of the buffer token into a list of new tokens. The resulted tokens are then put at the top of the buffer.

4. A new action MERGE is added to handle the "compound" form of token that appears sometimes in training data. For example, in the French treebank, “200 000” (with a whitespace) is often treated as one token. In our parser, this is obtained by applying MERGE when “200” is at the top of stack, and “000” is at the top of buffer.

Figure 1 describes 1) parser transitions applied to the stack and buffer and 2) the resulting stack and buffer states. Figure 2 gives an example of transition sequence in our joint system.

**Modules:** Our joint system extends the transition-based parser in Section 2.1 with extra modules to handle tokenization, UPOS and morphological tagging. The final loss function is the sum of the loss functions from the parser itself and these extra modules. Due to time limitation we did not introduce weights in the sum.

1. Tokenization module. When a string appears at buffer top, for each offset inside the string, predict whether to tokenize here. If tokenization happens at some offset $i$, apply TOKENIZE$(i)$ and transit to next state accordingly. If no tokenization happens, predict an

| **Stack$_t$** | **Buffer$_t$** | **Action** | **Stack$_{t+1}$** | **Buffer$_{t+1}$** | **Dependency** |
|---|---|---|---|---|---|
| $S, (\mathbf{u}, u), (\mathbf{v}, v)$ | $B$ | RIGHT-ARC$(r)$ | $S, (g_r(\mathbf{u}, \mathbf{v}), u)$ | $B$ | $u \overset{r}{\rightarrow} v$ |
| $S, (\mathbf{u}, u), (\mathbf{v}, v)$ | $B$ | LEFT-ARC$(r)$ | $S, (g_r(\mathbf{v}, \mathbf{u}), v)$ | $B$ | $u \overset{r}{\leftarrow} v$ |
| $S, (\mathbf{u}, u), (\mathbf{v}, v)$ | $(\mathbf{w}, w), B$ | SWAP | $S, (\mathbf{v}, v)$ | $(\mathbf{u}, u), (\mathbf{w}, w), B$ | — |
| $S$ | $(\mathbf{u}, u), B$ | SHIFT$(p, f)$ | $S, (\mathbf{u}, u)$ | $B$ | — |
| $S$ | $(\mathbf{w}, w), B$ | TOKENIZE$(i)$ | $S$ | $(\mathbf{w_1}, w_1), (\mathbf{w_2}, w_2), B$ | — |
| $S$ | $(\mathbf{w}, w), B$ | SPLIT$(w_1, ..., w_n)$ | $S$ | $(\mathbf{w_1}, w_1), ..., (\mathbf{w_n}, w_n), B$ | — |
| $S, (\mathbf{u}, u)$ | $(\mathbf{v}, v), B$ | MERGE | $S$ | $(g(\mathbf{u}, \mathbf{v}), \text{“}u\ v\text{”}), B$ | — |

Figure 1: Parser transitions indicating the action applied to the stack and buffer and the resulting state. Bold symbols indicate (learned) embeddings of words and relations, script symbols indicate the corresponding words and relations. $g_r$ and $g$ are compositions of embeddings. $w_1$ and $w_2$ are obtained by tokenizing $w$ at offset $i$. “$u\ v$” is the “compound" form of word.

| **Transition** | **Stack** | **Buffer** | **Dependency** | **Tags** |
|---|---|---|---|---|
| | [] | [Le-1, canton-2, des-3, Ulis-4, compte-5, 25-6, 785-7, habitants.-8] | | |
| SHIFT(Le, DET) | [Le-1] | [ canton-2, des-3, Ulis-4, compte-5, 25-6, 785-7, habitants.-8] | | (Le, DET) |
| SHIFT(canton, NOUN) | [ Le-1, canton-2 ] | [ des-3, Ulis-4, compte-5, 25-6, 785-7, habitants.-8] | | (canton, NOUN) |
| SPLIT(des, de les) | [ Le-1, canton-2 ] | [ de-3, les-4, Ulis-5, compte-6, 25-7, 785-8, habitants.-9 ] | | |
| LEFTARC(det) | [ canton-2 ] | [ de-3, les-4, Ulis-5, compte-6, 25-7, 785-8, habitants.-9 ] | Le $\xleftarrow{det}$ canton | |
| SHIFT(de, ADP) | [ canton-2, de-3 ] | [ les-4, Ulis-5, compte-6, 25-7, 785-8, habitants.-9 ] | | (de, ADP) |
| SHIFT(les, DET) | [ canton-2, de-3, les-4 ] | [ Ulis-5, compte-6, 25-7, 785-8, habitants.-9 ] | | (les, DET) |
| SHIFT(Ulis, PROPN) | [ canton-2, de-3, les-4, Ulis-5 ] | [ compte-6, 25-7, 785-8, habitants.-9 ] | | (Ulis, PROPN) |
| LEFTARC(det) | [ canton-2, de-3, Ulis-5 ] | [ compte-6, 25-7, 785-8, habitants.-9 ] | les $\xleftarrow{det}$ Ulis | |
| LEFTARC(case) | [ canton-2, Ulis-5 ] | [ compte-6, 25-7, 785-8, habitants.-9 ] | de $\xleftarrow{case}$ Ulis | |
| RIGHTARC(nmod) | [ canton-2 ] | [ compte-6, 25-7, 785-8, habitants.-9 ] | canton $\xrightarrow{nmod}$ Ulis | |
| SHIFT(compte, VERB) | [ canton-2, compte-6 ] | [ 25-7, 785-8, habitants.-9 ] | | (compte, VERB) |
| LEFTARC(nsubj) | [ compte-6 ] | [ 25-7, 785-8, habitants.-9 ] | canton $\xleftarrow{nsubj}$ compte | |
| SHIFT(25, NUM) | [ compte-6, 25-7 ] | [ 785-8, habitants.-9 ] | | (25, NUM) |
| MERGE | [ compte-6 ] | [ 25 785-7, habitants.-8 ] | | |
| SHIFT(25 785, NUM) | [ compte-6, 25 785-7 ] | [ habitants.-8 ] | | (25 785, NUM) |
| TOKENIZE(offset=9) | [ compte-6, 25 785-7 ] | [ habitants-8, .-9 ] | | |
| SHIFT(habitants, NOUN) | [ compte-6, 25 785-7, habitants-8 ] | [ .-9 ] | | (habitants, NOUN) |
| LEFTARC(nummod) | [ compte-6, habitants-8 ] | [ .-9 ] | compte $\xleftarrow{nummod}$ 25 785 | |
| RIGHTARC(obj) | [ compte-6 ] | [ .-9 ] | compte $\xrightarrow{obj}$ habitants | |
| SHIFT(., PUNCT) | [ compte-6, .-9 ] | [ ] | | (., PUNCT) |
| RIGHTARC(punct) | [ compte-6 ] | [ ] | compte $\xrightarrow{punct}$ . | |

Figure 2: Transition sequence for “*Le canton des Ulis compte 25 785 habitants.*” with the joint model in Section 2.2. “habitants.-8” means that “habitants.” is the 8th token in the current token stream. Morphological tags are omitted in this figure.

action from the set of other (applicable) actions, and transit accordingly.

2. Tagging module. If SHIFT is predicted as the next action, a sub-routine will call classifiers to predict POS and morph-features. The joint system could also predict lemma, but experiment results lead to the decision of not predicting lemma.

3. Split module. If SPLIT is predicted as the next action, a sub-routine will call classifiers to predict the output of SPLIT.

**Word embeddings:** Parser state representation is composed by three Stack-LSTM's: stack, buffer, actions, as in (Ballesteros et al., 2017). To represent each word in the stack and the buffer, we use character-based word embeddings together with pretrained embeddings and word embeddings trained in the system. The character-based word embeddings are illustrated in Figure 3. For tokenization module, we deployed a character-based embeddings to represent not only the string to tokenize, but also the offset, as illustrated in Figure 4.

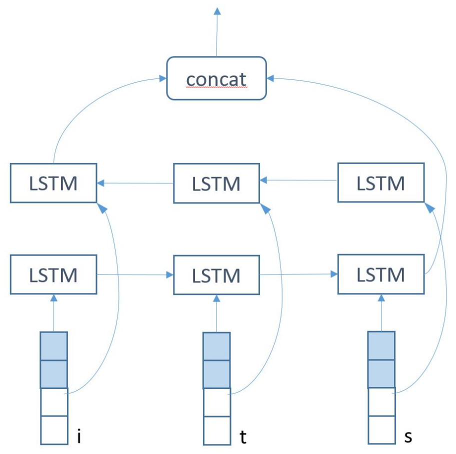

Figure 3: Character-based embeddings from bi-LSTMs to represent the token “its”.

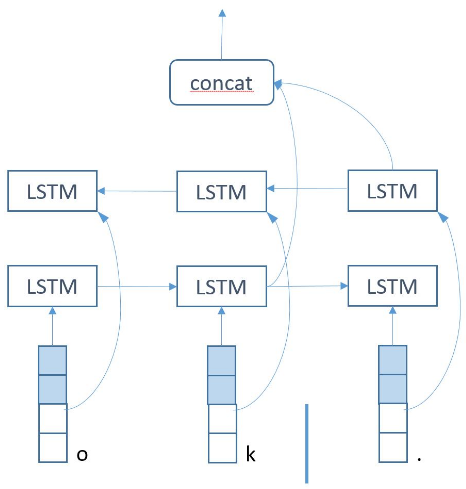

Figure 4: Character-based embeddings from bi-LSTMs to represent the string "ok." and the offset 2 in consideration.

## 2.3 Cross-Lingual Parser

We adapted cross-lingual architecture of Ammar et al. (2016) (also based in the Stack-LSTM parser) in our joint model presented in Section 2.2 to handle low-resource and zero-shot languages. This architecture enables effective training of the Stack-LSTM parser on multilingual training data. Words in each language are represented by multilingual word embeddings to allow cross-lingual sharing; whereas language specific characteristics are captured by means of language embeddings. Ammar et al. (2016) experiment with a) pre-specified language embeddings based on linguistic features and b) language embeddings learned jointly with the other parameters. The former requires external linguistic knowledge and the latter can be trained only when all languages in the set have enough annotated training data. We take a third approach – we pretrain language embeddings on raw text (explained in next section) and then keep them fixed during parser training. In our implementation, pretrained language embeddings are concatenated with word representation and with parser state.

We use cross-lingual version of our parser for all zero-shot languages (these are: Breton, Naija, Faroese and Thai), most low resource languages (these are: Buryat, Kurmanji, Kazakh, Sorbian Upper, Armenian, Irish, Vietnamese, Northern Sami and Uyghur ), and some other languages in which we observed strong improvements on development data when parsing with a cross-lingual model trained in the same language family (these are: Ancient Greek – grc_proiel and grc_perseus, Swedish – sv_pud, Norwegian Nynorsk – no_nynorsklia ).

| | Nearest Neighbors | | |
|---|---|---|---|
| | Character | Word | Sub-Word |
| German | Dutch | Swedish | Afrikaans |
| Portuguese | Galician | Galician | Galician |
| Ukrainian | Bulgarian | Bulgarian | Russian |
| Hindi | Hebrew | Urdu | Urdu |
| Kazakh | Ukrainian | Buryat | Buryat |
| Kurmanji | Turkish | Urdu | Naija |
| Persian | Arabic | Uyghur | Uyghur |

Table 1: Nearest neighbors of a variety of languages based on language vectors learned via models of varying granularities: Characters, Words and Sub-Word units (BPEs)

In zero-shot setup, we observed that language embeddings in fact hurt parser performance[4]. This is consistent with the findings of Ammar et al. (2016) for a similar setup as noted in footnote 30. In such cases, we trained multilingual parser without language embeddings, relying only on multilingual word embeddings.

**Language embeddings:** Ammar et al. (2016) architecture utilizes language embeddings that capture language nuances and allow generalization. We adapt the method of Östling and Tiedemann (2017) to pretrain language embeddings. This method is essentially a character-level language model, where a 2-layered LSTM predicts next character at each time step given previous character inputs. A language vector is concatenated to each input as well as the hidden layer before final softmax. The model is trained on a raw corpus containing texts from different languages. Language vectors are shared within the same language.

The model of Östling and Tiedemann (2017) operates at the level of characters; They restrict their experiments to the languages that are written in Latin, Cyrillic or Greek scripts. However, the shared task data spanned a variety of languages with scripts not included in this set. Moreover, there are languages in the shared task that are closely related yet written in different scripts – examples include Hindi-Urdu and Hebrew-Arabic pairs. In preliminary experiments, we found that the language vectors learned via the character-based model, fail to capture language similarities when the script is different. We therefore employ three variations of Östling and Tiedemann (2017) model that differ in granularity of input

[4]We performed this experiment in an off-line, artificiality created zero-shot setup.

units: we use 1) characters, 2) words and 3) sub-word units (Byte Pair Encodings, BPEs, Sennrich et al. (2015)) as inputs. Table 1 shows the nearest neighbors of a variety of languages based on the language vectors from each model. Notice that the nearest neighbor of Hindi is Urdu only when model operates on word and sub-word levels. The vectors learned from the three versions are concatenated to form final language embeddings.

This method requires a multilingual corpus for training. We take the first 20K tokens from each training corpus – for the corpora that had fewer tokens, additional raw text is taken from OPUS resources. For BPE inputs, we limit the size of BPE vocabulary to 100,000 symbols.

**Multilingual word embeddings:** The cross-lingual parser of Ammar et al. (2016) requires word vectors for each language to be in the same universal space. To this end, we use alignment matrices provided by Smith et al. (2017) for Bojanowski et al. (2017) word embeddings. However, for several low-resource languages, pre-computed alignment matrices were not available. These include Naija, Faroese, Kurmanji, Northern Sami, Uyghur, Buryat and Irish. For these languages, to map monolingual embeddings to multilingual space, we seed the mapping algorithm of Smith et al. (2017) with freely available dictionaries[5] combined with shared vocabulary with one of the already mapped languages.

## 2.4 Sentence-based Ensemble and MST Ensemble

### 2.4.1 Graph-based ensemble

We adapt Sagae and Lavie (2006) ensemble method to our Stack-LSTM only models (see Section 2.1) to obtain the final parses of Chinese, Japanese, Hebrew, Hungarian, Turkish and Czech. Kuncoro et al. (2016) already tried an ensemble of several Stack-LSTM parser models achieving state-of-the-art in English, German and Chinese, which motivated us to improve the results of our greedy decoding method.[6]

[5]Bilingual dictionaries used for multilingual mapping of word embeddings,
https://people.uta.fi/ km56049/same/svocab.html
https://github.com/apertium/apertium-kmr-eng
https://github.com/apertium/apertium-fao-nor

[6]Kuncoro et al. (2016) developed ensemble distillation into a single model which we did not attempt to try for the Shared Task but we leave for future developments.

### 2.4.2 Model Rescoring: Sentence Level Ensemble

For all of the languages and treebank combinations except for Chinese, Japanese, Hebrew, Hungarian, Turkish and Czech, we apply a sentence-level ensemble technique to obtain the final parses.

We train 10-20 parsing models per language-treebank (see Section 4.2). For an input sentence, with each model we generate a parsing output and a parsing score by adding up the scores of all the actions along the transition sequence (see Figure 1) . Then for each input sentence, we choose the parsing output with the highest model score. The joint model handles tokenization before considering other parsing actions, and makes tokenization decision on every offset; this means that we need to include the normalized score for each tokenization decision. The score assigned to tokenizing a string $S$ at offset $n$ is $(\sum_{i=1\ldots n-1} Score_{keep}(S,i) + Score_{tok}(S,n))/n$, and the score assigned to keeping $S$ as a whole is $(\sum_{i=1\ldots len(S)} Score_{keep}(S,i))/len(S)$.

This simple technique worked fairly well, leading to significant LAS F1 improvement compared with the single model output. From 77.53 LAS in average on single model output of 58 treebanks dev set, 10-model ensemble improves LAS to 78.75 and 20-model ensemble improves LAS to 79.79. Due to time limitation, we only ran a 20-model sentence-level ensemble on 15 treebanks (ar_padt, ca_ancora, cs_fictree, cu_proiel, da_ddt, el_gdt, en_ewt, fr_sequoia, fro_srcmf, gl_ctg, hi_hdtb, ko_gsd, ko_kaist, pl_sz, pt_bosque) while in the rest we ran a 10-model ensemble.

In multi-lingual setting, we ran 5-model ensemble in most cases except grc_proiel, grc_perseus and sv_lines where 10-models ensembles were used for decoding and no_nynorsklia where a single model was used for decoding.

## 3 Sentence Segmentation

For sentence segmentation we adapted the punctuation prediction system by Ballesteros and Wanner (2016). This model is derived from the Stack-LSTM parser introduced in Section 2.1 and it uses the same architecture (including a stack, a buffer and a stack containing the transitions already taken) but it is restricted to two distinct transitions, either SHIFT or BREAK (which adds a sentence marker between two tokens). The system is therefore context dependent and it makes decisions about sentence boundaries regardless of punctuation symbols or other typical indicative markers.[7]

We only applied our sentence segmentation system for the datasets in which we surpassed the development sets baseline numbers provided by the organizers of the Shared Task by a significant margin, these are: bg_btb, es_ancora, et_edt, fa_seraji, id_gsd, it_postwita, la_proiel, and ro_rrt.

**Handling document segmentation:** In 58 of the 73 datasets with training data, the train.txt file contains less than 10 paragraphs, and 53 of these contain no paragraph breaks. Thus, if we assumed (incorrectly) that paragraph breaks occur at sentence boundaries and naïvely used paragraphs as training units for the sentence break detector, we would face a huge computational hurdle: we would accumulate the loss over hundreds of thousands of words before computing backpropagation. We addressed this issue by adopting a sliding window approach. The data is segmented into windows containing $W$ words with an overlap of $O$ words. Each window is treated as a training unit, where the loss is computed, the optimizer is invoked and the stack LSTM state is reset. The main challenge of a sliding window approach is to compensate for edge effects: a trivial implementation would ignore the right and left context, which results in diminished ability of detecting sentence breaks near the beginning and the end of the window. Since we desire to keep $W$ to a manageable size, we cannot ignore edge effects. We use two different approaches to provide left and right context to the stack LSTM. The right context is provided by the last $O$ words of the window (with the obvious exception of the last window). Thus, the sentence segmentation algorithm predicts sentence breaks for the first $W - O$ words. To provide left context, we snapshot the stack and action buffer after the last prediction in the window, we slide the window to the right by $W - O$ words, we reset the LSTM state, and we prime the input buffer with the $L$ words to the left of the new window, the action buffer with the most recent $L$ actions, and the stack with the $L$ topmost entries from the snapshot. We explored using different parameter for the window overlap and the size of the left context, and concluded that asymmetric approaches did not provide an advantage over selecting $L = O$. The parameters for the system used for the evaluation are $W = 100$, $L = O = 30$.

[7]We again used character-based word representations (Ballesteros et al., 2015) and pretrained word embeddings in the same way as the system described in Section 2.1.

## 4 Models

### 4.1 Stack-LSTM

For 6 treebanks (cs_pdt, he_htb, ja_gsd, hu_szeged, tr_imst, zh_gsd), we trained 20 baseline Stack-LSTM models for parsing (utilizing UDPipe pre-processing for sentence segmentation, tokenization and UPOS tagging) per treebank. And the 20 parsing model outputs are rescored with graph-based ensemble (see Section 2.4.2). Independent LSTM models are trained on each treebank for labeling.

All models for the 6 treebanks are trained with dimension 200. Except for ja_gsd and zh_gsd, the models are trained with character embeddings.

We utilized diverse set of word embeddings for Stack-LSTM and graph-based models: cs_pdt, he_htb and tr_imst (CoNLL2017 embedding with dimension 100), ja_gsd (in-house cross-lingual embeddings with dimension 300), hu_szeged and zh_gsd (Facebook embeddings with dimension 300).

### 4.2 Joint Models

We set input and hidden-layer dimension to 100 and action vector dimension to 20. CoNLL 2017 pretrained embeddings (dimension 100) were used wherever available. We used Facebook embeddings (dimension 300) for af_afribooms, got_proiel and sr_set.

For en_pud and fi_pud where no training and dev set is available, the models trained from the biggest treebank in the same language (en_ewt and fi_tdt) are used to parse the testset. ru_syntagrus model is used to parse ru_taiga testset because of higher score. For gl_treegal and la_perseus where no development data is available, 1/10 of training data is set aside as development set.

We use sentence-based ensemble (see Section 2.4.2) for all models since the parser presented in Section 2.2 may produce a different number of tokens in the output due to tokenization.

### 4.3 Cross Lingual

Cross-lingual models are trained with input and hidden layers of dimension 100 each, and action vectors of dimension 20. Pretrained multilingual

| Target Treebank | Set of Training Treebanks |
|---|---|
| pcm_nsc | en_ewt, pt_bosque |
| fo_oft | no_bokmaal, no_nynorsk, sv_lines, sv_talbanken, da_ddt, sme_giella |
| br_keb | af_afribooms, en_ewt, nl_alpino, de_gsd, got_proiel, ga_idt |
| th_pud | id_gsd, vi_vtb |
| kmr_mg | hi_hdtb, ur_udtb, fa_seraji, kmr_mg |
| vi_vtb | id_gsd, vi_vtb, hu_szeged |
| bxr_bdt | ug_udt, bxr_bdt, fa_seraji, tr_imst |
| ug_udt | ug_udt, bxr_bdt, fa_seraji, tr_imst |
| kk_ktb | uk_iu, bg_bdt, sk_snk, sl_ssj, sl_sst, hr_set, cs_fictree, pl_lfg, lv_lvtb, cu_proiel, bxr_bdt, hsb_ufal, kk_ktb |
| hsb_ufal | uk_iu, bg_bdt, sk_snk, sl_ssj, sl_sst, hr_set, cs_fictree, pl_lfg, lv_lvtb, cu_proiel, bxr_bdt, hsb_ufal, kk_ktb |
| sme_giella | no_bokmaal, no_nynorsk, sv_lines, sv_talbanken, da_ddt, sme_giella |
| ga_idt | af_afribooms, en_ewt, nl_alpino, de_gsd, got_proiel, ga_idt |
| hy_armtdp | grc_perseus, grc_proiel, el_gdt, hy_armtdp |
| grc_perseus | grc_perseus, grc_proiel, el_gdt, hy_armtdp |
| grc_proiel | grc_perseus, grc_proiel, el_gdt, hy_armtdp |
| sv_pud | no_bokmaal, no_nynorsk, sv_lines, sv_talbanken, da_ddt, sme_giella |
| no_nyrosklia | no_bokmaal, no_nynorsk, sv_lines, sv_talbanken, da_ddt, sme_giella |
| sl_sst | uk_iu, bg_bdt, sk_snk, sl_ssj, sl_sst, hr_set, cs_fictree, pl_lfg, lv_lvtb, cu_proiel, bxr_bdt, hsb_ufal, kk_ktb |

Table 2: For each target treebank (left column), the parser was trained on a set of related languages' treebanks (right column). Top section contains zero-shot languages (no training data at all), middle section lists low resource languages (with small amounts of training data) and the bottom section lists the test treebanks for which little or no in-domain training data was provided but a different training treebank of the same language was available.

word embeddings are of dimension 300 and pre-trained language embeddings are of dimension 192 (concatenation of three 64 length vectors).

For each target language, cross-lingual parser is trained on a set of treebanks from related languages. Table 2 details the sets of source treebanks used to train the parser for each target treebank. In the case of low resource languages, training algorithm is modified to sample from each language equally often. This is to ensure that the parser is still getting most of its signal from the language of interest. In all cross-lingual experiments, sentence level ensemble (see Section 2.4.2) is used.

### 4.4 Segmentation

Sentence segmentation models have hidden layer dimension equal to 100. It relies on the fast-Text embeddings (Bojanowski et al., 2017), which have dimension equal to 300. The sliding window width is of 100 words, and the overlap between adjacent windows is 30 words.

## 5 Results

Table 4 presents the average F1 LAS results grouped by treebank size and type of our system compared to the baseline UDPipe 1.2 (Straka and Straková, 2017). Table 3 presents the F1 LAS results for all languages compared to the baseline UDPipe 1.2. Our system substantially surpassed the baseline but it is far from the best system of the task in most cases. Some exceptions are the low resource languages like kmr_mg in which our system is the best, bxr_bdt in which it is the second best and hsb_ufal in which it is the 3rd best; probably due to our cross-lingual approach (see Section 2.3). In ko_gsd and ko_kaist our scores are 17.61 and 13.56 higher than the baseline UDPipe 1.2 as a result of character-based embeddings (similar result as (Ballesteros et al., 2015)), but still far from the best system.

It is worth noting that, on most treebanks our system used joint model to do tokenization in one pass together with parsing, and we trained with no more than UD-2.2 training data. Our overall tokenization score is 97.30, very close (-0.09) to the baseline UDPipe 1.2, our tokenization score on big treebanks is 99.24, the same as the baseline.

For sentence segmentation, as explained in Section 3, we only used our system for the treebanks in which it performed better than the baseline in the development set. We ranked 4th, 0.5 above the baseline and 0.36 below the top-ranking system. Table 5 shows the results of our system for the 8 treebanks for which we submitted a run with our own sentence segmenter. For the other treebanks we used the baseline UDPipe 1.2. We remark that for la_projel, where no punctuation marks are available, our system outperformed UDPipe Future by 3.79 and UDPipe 1.2 by 3.99. Finally, for it_postwita, a dataset where the punctuation is as indicative of sentence breaks and other character patterns, our system outperformed UDPipe future

| Treebank | Baseline | Ours | Treebank | Baseline | Ours |
|---|---|---|---|---|---|
| af_afribooms | 77.88% | 80.53% (13) | ar_padt | 66.41% | 69.13% (15) |
| bg_btb | 84.91% | 86.83% (15) | br_keb | 10.25% | 9.45% (18) |
| bxr_bdt | 12.61% | 19.22% (2) | ca_ancora | 85.61% | 87.60% (16) |
| cs_cac | 83.72% | 87.08% (15) | cs_fictree | 82.49% | 85.83% (15) |
| cs_pdt | 83.94% | 87.08% (14) | cs_pud | 80.08% | 83.00% (12) |
| cu_proiel | 65.46% | 69.65% (10) | da_ddt | 75.43% | 77.87% (15) |
| de_gsd | 70.85% | 73.91% (16) | el_gdt | 82.11% | 84.37% (13) |
| en_ewt | 77.56% | 78.37% (16) | en_gum | 74.20% | 76.11% (17) |
| en_lines | 73.10% | 74.31% (16) | en_pud | 79.56% | 81.13% (13) |
| es_ancora | 84.43% | 86.90% (16) | et_edt | 75.02% | 79.89% (15) |
| eu_bdt | 70.13% | 75.09% (15) | fa_seraji | 79.10% | 82.71% (15) |
| fi_ftb | 75.64% | 81.33% (13) | fi_pud | 80.15% | 83.49% (12) |
| fi_tdt | 76.45% | 79.79% (15) | fo_oft | 25.19% | 38.84% (7) |
| fr_gsd | 81.05% | 84.32% (13) | fro_srcmf | 79.27% | 83.49% (11) |
| fr_sequoia | 81.12% | 82.79% (15) | fr_spoken | 65.56% | 65.34% (17) |
| ga_idt | 62.93% | 61.45% (19) | gl_ctg | 76.10% | 76.81% (17) |
| gl_treegal | 66.16% | 62.44% (12) | got_proiel | 62.16% | 63.52% (11) |
| grc_perseus | 57.75% | 66.23% (11) | grc_proiel | 67.57% | 71.77% (13) |
| he_htb | 57.86% | 60.17% (15) | hi_hdtb | 87.15% | 89.72% (14) |
| hr_set | 78.61% | 82.64% (15) | hsb_ufal | 23.64% | 42.33% (3) |
| hu_szeged | 66.76% | 67.25% (16) | hy_armtdp | 21.79% | 19.25% (20) |
| id_gsd | 74.37% | 77.03% (15) | it_isdt | 86.26% | 87.14% (18) |
| it_postwita | 66.81% | 73.85% (5) | ja_gsd | 72.32% | 73.29% (15) |
| ja_modern | 22.71% | 22.74% (7) | kk_ktb | 24.21% | 23.72% (12) |
| kmr_mg | 23.92% | 30.41% (1) | ko_gsd | 61.40% | 79.01% (13) |
| ko_kaist | 70.25% | 83.81% (15) | la_ittb | 75.95% | 82.43% (15) |
| la_perseus | 47.61% | 43.40% (19) | la_proiel | 59.66% | 64.69% (14) |
| lv_lvtb | 69.43% | 73.17% (14) | nl_alpino | 77.60% | 81.43% (15) |
| nl_lassysmall | 74.56% | 76.49% (17) | no_bokmaal | 83.47% | 86.64% (16) |
| no_nynorsk | 82.13% | 85.31% (16) | no_nynorsklia | 48.95% | 58.28% (8) |
| pcm_nsc | 12.18% | 13.03% (12) | pl_lfg | 87.53% | 90.42% (15) |
| pl_sz | 81.90% | 82.81% (17) | pt_bosque | 82.07% | 84.12% (16) |
| ro_rrt | 80.27% | 82.87% (14) | ru_syntagrus | 84.59% | 87.79% (15) |
| ru_taiga | 55.51% | 63.00% (9) | sk_snk | 75.41% | 77.91% (16) |
| sl_ssj | 77.33% | 82.22% (13) | sl_sst | 46.95% | 48.36% (11) |
| sme_giella | 56.98% | 55.97% (18) | sr_set | 82.07% | 83.84% (16) |
| sv_lines | 74.06% | 75.90% (16) | sv_pud | 70.63% | 76.65% (9) |
| sv_talbanken | 77.91% | 80.63% (16) | th_pud | 0.70% | 0.67% (12) |
| tr_imst | 54.04% | 57.19% (16) | ug_udt | 56.26% | 60.25% (14) |
| uk_iu | 74.91% | 77.74% (14) | ur_udtb | 77.29% | 79.80% (15) |
| vi_vtb | 39.63% | 43.48% (7) | zh_gsd | 57.91% | 60.63% (16) |

Table 3: Final F1 LAS results of our system compared with the baseline. We show our ranking for the particular treebank between parenthesis next to our score.

by 29.84 and UDPipe 1.2 by 36.99.

The 2018 edition of the Extrinsic Parser Evaluation Initiative (EPE 2018) (Fares et al., 2018) runs in collaboration with the 2018 Shared Task on Multilingual Parsing. Parsers are evaluated against the three EPE downstream systems: biological event extraction, fine-grained opinion analysis, and negation resolution. This provides opportunities for correlating intrinsic metrics with downstream effects on the three relevant applications. Our system qualified 12th overall, being 10th in event extraction, 13th in negation resolution and 14th in opinion analysis.

| Treebank | Baseline | Ours |
|---|---|---|
| All | 65.80% | 69.11% (13) |
| Big treebanks | 74.14% | 77.55% (15) |
| PUD treebanks | 66.63% | 69.40% (10) |
| Small treebanks | 55.01% | 56.13% (18) |
| Low resource | 17.17% | 21.88% (7) |

Table 4: Average F1 LAS results (grouped by treebank size and type) of our system compared with the baseline. We show our ranking in the same category between parenthesis next to our score.

| Treebank | Baseline | Ours (Rank) |
|---|---|---|
| bg_btb | 92.85 | 92.24 (24) |
| es_ancora | 98.26 | 97.89 (24) |
| et_edt | 90.02 | 91.29 (6) |
| fa_seraji | 98.74 | 98.09(24) |
| id_gsd | 92.00 | 92.03 (6) |
| it_postwita | 21.80 | 58.79 (2) |
| la_proiel | 35.16 | 39.15 (2) |
| ro_rrt | 93.72 | 94.40 (5) |

Table 5: Sentence segmentation F1 and rank, compared to baseline. We only include the results in which our system surpassed the baseline in the development set.

## 6 Conclusion

We presented the IBM Research submission to the CoNLL 2018 Shared Task on Universal Dependency Parsing. We presented a new transition-based algorithm for joint (1) tokenization, (2) tagging and (3) parsing that extends the arc-standard algorithm with new transitions. In addition, we also used the same Stack-LSTM framework for sentence segmentation achieving good results.

## Acknowledgments

We thank Radu Florian, Todd Ward and Salim Roukos for useful discussions.

## References

Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1354–1359.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Miguel Ballesteros, Chris Dyer, Yoav Goldberg, and Noah A. Smith. 2017. Greedy transition-based dependency parsing with stack lstms. *Computational Linguistics*, 43(2):311–347.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 349–359.

Miguel Ballesteros and Leo Wanner. 2016. A neural network architecture for multilingual punctuation generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1048–1053. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

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.

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 (ACL)*, pages 334–343.

Murhaf Fares, Stephan Oepen, Lilja Øvrelid, Jari Björne, and Richard Johansson. 2018. The 2018 Shared Task on Extrinsic Parser Evaluation. On the downstream utility of English Universal Dependency parsers. In *Proceedings of the 22nd Conference on Natural Language Learning*, Brussels, Belgia.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one mst parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753. Association for Computational Linguistics.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Joakim Nivre. 2004. Inductive dependency parsing. Technical Report 04070, Växjö University, School of Mathematics and Systems Engineering.

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 (ACL-IJCNLP)*, pages 351–359.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Silvie Cinková, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Radu Ion, Elena Irimia, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shinsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Robert Östling, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jonathan North Washington, Mats Wirén, Tak-sum Wong, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. 2017a. Universal dependencies 2.1. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre and Chiao-Ting Fang. 2017. Universal dependency evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 86–95.

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

Joakim Nivre 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, Prague, http://hdl.handle.net/11234/1-2184.

Joakim Nivre et al. 2018. Universal Dependencies 2.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/.

Robert Östling and Jörg Tiedemann. 2017. Continuous multilinguality with language vectors. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 644–649. Association for Computational Linguistics.

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 *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*, pages 268–299, Berlin Heidelberg New York. Springer.

Mohammad Sadegh Rasooli and Joel Tetreault. 2013. Joint parsing and disfluency detection in linear time. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 124–129, Seattle, Washington, USA. Association for Computational Linguistics.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Samuel L. Smith, David H. P. Turban, Steven Hamblin, and Nils Y. Hammerla. 2017. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *CoRR*, abs/1702.03859.

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)*, Portorož, Slovenia. European Language Resources Association.

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. Association for Computational Linguistics.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, joint syntactic-semantic parsing with stack lstms. *CoRR*, abs/1606.08954.

Daniel Zeman, Filip Ginter, Jan Hajič, Joakim Nivre, Martin Popel, Milan Straka, and et al. 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*, pages 1–20. 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 1–20, Brussels, Belgium. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. Joint word segmentation and POS tagging using a single perceptron. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 888–896.