Deep Contextualized Word Embeddings for Universal Dependency Parsing

YIJIA LIU, WANXIANG CHE, YUXUAN WANG, BO ZHENG, BING QIN, and TING LIU,
Harbin Institute of Technology, China

Deep contextualized word embeddings (Embeddings from Language Model, short for ELMo), as an emerging and effective replacement for the static word embeddings, have achieved success on a bunch of syntactic and semantic NLP problems. However, little is known about what is responsible for the improvements. In this article, we focus on the effect of ELMo for a typical syntax problem—universal POS tagging and dependency parsing. We incorporate ELMo as additional word embeddings into the state-of-the-art POS tagger and dependency parser, and it leads to consistent performance improvements. Experimental results show the model using ELMo outperforms the state-of-the-art baseline by an average of 0.91 for POS tagging and 1.11 for dependency parsing. Further analysis reveals that the improvements mainly result from the ELMo's better abstraction ability on the out-of-vocabulary (OOV) words, and the character-level word representation in ELMo contributes a lot to the abstraction. Based on ELMo's advantage on OOV, experiments that simulate low-resource settings are conducted and the results show that deep contextualized word embeddings are effective for data-insufficient tasks where the OOV problem is severe.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; **Natural language processing**; **Phonology / morphology**;

Additional Key Words and Phrases: Natural language processing, deep contextualized word embeddings, universal dependency parsing, POS tagging, out-of-vocabulary words, visualization

ACM Reference format:
Yijia Liu, Wanxiang Che, Yuxuan Wang, Bo Zheng, Bing Qin, and Ting Liu. 2019. Deep Contextualized Word Embeddings for Universal Dependency Parsing. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* 19, 1, Article 9 (July 2019), 17 pages.
https://doi.org/10.1145/3326497

1 INTRODUCTION

Effectively learning from raw text has long been the goal of natural language processing (NLP). Deep contextualized word embeddings [35, ELMo], which use a bidirectional long short term memory [16, LSTM] language model to learn contextual information from the raw text, have been shown as a simple yet effective method to improve several NLP tasks, like question answering,

Fund Project: This work was supported by the Natural Science Foundation of China (NSFC) via grants no. 61632011, no. 61772156, and no. 61772153.
Authors' address: Y. Liu, W. Che (corresponding author), Y. Wang, B. Zheng, B. Qin, and T. Liu, Harbin Institute of Technology, No. 2 YiKuang Street, Technique and Innovation Building, HIT Science Park, Harbin, HeiLongJiang 150001, China; emails: {yjliu, car, yxwang, bzheng, qinb, tliu}@ir.hit.edu.cn.

2375-4699/2019/07-ART9 $15.00
https://doi.org/10.1145/3326497

textual entailment, semantic role labeling [35], coreference resolution [22], and constituency parsing [20], thus draws a lot of research attention in the NLP community [36].

Although achieving good performance, little is known about what is responsible for the improvements, especially for syntactic problems. To track the source of improvements, an ideal test-bed should consist of various datasets for a certain problem. By comparing ELMo's effects on different datasets, the relation between the improvements and the intrinsic characteristics of a dataset can be revealed, which in turn helps us better understand ELMo. In this article, we choose universal dependencies [33, short for *UD*] as the test-bed, since it provides a cross-linguistically consistent grammatical annotation guideline and more than 100 dependency treebanks for over 60 languages in the world. Answering *whether deep contextualized word embeddings improve the universal dependency parsing performance* and exploring *what is responsible for such improvements* on these treebanks can teach us with the insight for using ELMo on the syntactic tasks.

In this article, we study the effect of ELMo on UD part-of-speech (POS) tagging and parsing. We base our POS tagger and dependency parser on the state-of-the-art algorithm of Dozat et al. [12] and use ELMo as an additional word embedding to improve the algorithm. Experimental results on the 57 treebanks from the CoNLL 2018 shared task—Universal Dependencies parsing [41] show that using ELMo leads to consistent improvements and outperforms the state-of-the-art baseline without ELMo. The averaged improvement for POS tagging is 0.91 accuracy and the improvement for dependency parsing 1.11 LAS.

After obtaining the improved UD parsing results, we further analyze the reasons. Five attributes of the treebank along with their correlations to the performance gains are examined and this analysis reveals that the ELMo improves parsing by mainly improving the out-of-vocabulary (OOV) word performance. Further ablation on the ELMo model shows that ELMo can abstract an unseen OOV word as well as the one recalled during ELMo learning, and ELMo's character-level word representation contributes a lot to this abstraction. A visualization further confirms this by clustering OOV words of the same POS together in the embedding space.

Based on such analysis, we seek ELMo's ability to solve the syntactic problems in low-resource settings by simulating experiments. These results show that the model using ELMo requires fewer labeled data to achieve high-quality parsing, and this indicates that ELMo is a promising technique for NLP tasks whose data are insufficient.

Major contributions of this work include the following:

- —We achieve improved UD parsing performance on an extensive set of data with ELMo and these results outperform the state-of-the-art baseline, which testifies to the effect of ELMo on UD parsing.
- —We conduct a thorough analysis of the results and show the major source of improvements as better OOV performance. The improved OOV is mainly led by ELMo's word abstraction ability to which the character-level word representation in ELMo contributes a lot.
- —Based on the analysis, we conduct low-resource simulation experiments and the results show that ELMo helps to achieve high-quality parsing with fewer labeled data.

2 RELATED WORK

Dependency Parsing. Dependency parsing is a fundamental NLP problem, which analyzes the grammatical structure of a sentence and establishes relationships between "head" words and their modifying words. The formalism and dataset for dependency parsing have evolved from the language and genre dependent dataset of Penn Treebank [27] into a universal syntax formalism covering hundreds of treebanks [33]. The treebanks of the UD project are annotated using a set of universal

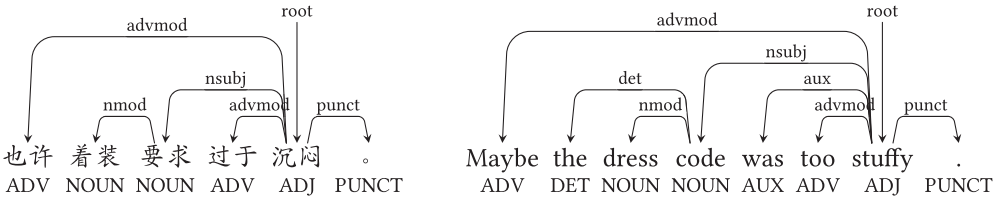

Fig. 1. Examples of UD trees. The sentences in left and right figures are a translation to each other and they share a similar syntactic structure.

POS tags and dependency relations but differ in languages and genres (see Figure 1 for an example). This provides many structural and linguistic challenges for the parser.

In past decades, several data-driven dependency parsing algorithms have been proposed, including (1) the graph-based algorithms [30] which find the maximum spanning tree based on the scoring functions defined on the sub-tree structures, and (2) the transition-based algorithms [14, 32, 43] which model the generation of dependency tree using a probabilistic finite-state machine. With the success of applying neural networks to NLP problems, the current state-of-the-art parser [11, 12] simplified the decoding algorithm into a head word classification. The quality of context representation plays an important role in this algorithm and Dozat and Manning [11] use a stacked long-short-term memory (LSTM) to model the context.

Semi-Supervised Learning. Effectively using the unlabeled data has long been a promising research direction for *semi-supervised* dependency parser. Chen et al. [6] derived features from automatically parsed unlabeled data to improve graph-based parser's performance before the resurge of deep neural models, and their method was further developed by Chen et al. [7]. Liang et al. [24] also studied the problem of using a complex structure model's output to train a simple classifier on the unlabeled data. The application of word embedding in NLP problems can also be considered as a success in making use of the unlabeled data since the embeddings are pretrained on the large-scaled unlabeled data [31, 34]. The deep contextualized word embeddings [1, 10, 28, 35] stepped further in this research direction by deriving context-aware embeddings and improving multiple NLP tasks.

Our work follows previous semi-supervised work and for the first time applies ELMo to cross-lingual syntactic analysis on a wide range of languages. Consistent improvements in the experimental results confirm that ELMo is an effective way of using unlabeled data.

Disseting Contextualized Word Embeddings. With the success of deep contextualized word embeddings, the research community becomes more and more interested in understanding its effect. Peters et al. [36] studied the alternatives of LSTM for modeling the context in bidirectional language modeling and they found that the alternative architectures like gated CNN [9] and Transformer [40] are also effective to learn high-quality contextual vectors. Bowman et al. [5] studied the effect of contextualized word embeddings in the perspective of the multi-task learning problem and made a thorough study of the tasks beyond language modeling. They found that among the studied tasks, language modeling is still a good choice. Zhang and Bowman [42] further confirmed this by showing the superiority of language modeling over translation in syntactic problems. Similar conclusions on the syntactic representations of language modeling were drawn by Tenney et al. [39].

Compared with the work which focuses more on sentence modeling, our work pays more attention to words and fills the landscape of understanding contextual vectors by studying their effect on universal dependency parsing. Our work suggests that the ability to abstract OOV word from ELMo is the key factor for ELMo's success on syntactic problems.

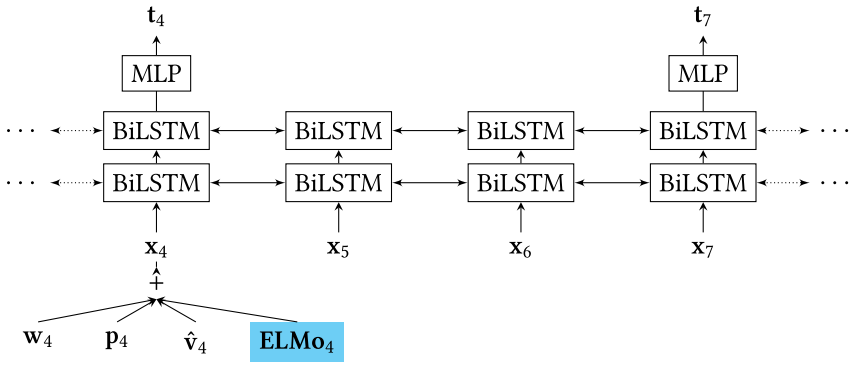

Fig. 2. The framework for deep biaffine tagger. In this article, we use ELMo as an additional word input, as highlighted in the cyan block.

3 DEEP BIAFFINE TAGGER AND PARSER

We based our system on the tagger and parser of Dozat and Manning [11] and Dozat et al. [12]. The core idea of the tagger and parser is using an LSTM network to produce the vector representation for each word and then predict POS tags and dependency relations using the representation.

Deep Biaffine Tagger. For the tagger whose input is the word alone, this representation is calculated as

$$\mathbf{h}_i = \text{BiLSTM}(\mathbf{v}_1^{(word)}, \ldots, \mathbf{v}_n^{(word)})_i,$$

where $\mathbf{v}_i^{(word)}$ is the word representation. After getting $\mathbf{h}_i$, the scores of tags are calculated as

$$\begin{aligned}
\mathbf{h}_i^{(pos)} &= \text{MLP}^{(pos)}(\mathbf{h}_i), \\
\mathbf{s}_i^{(pos)} &= W \cdot \mathbf{h}_i^{(pos)} + \mathbf{b}^{(pos)}, \\
y_i^{(pos)} &= \operatorname*{argmax}_j s_{i,j}^{(pos)},
\end{aligned}$$

where each element in $\mathbf{s}_i^{(pos)}$ represents the possibility that the ith word is assigned with the corresponding tag (see Figure 2).

Deep Biaffine Parser. For the parser whose inputs are the word and POS tag, the vector representation of each word $\mathbf{h}_i$ is calculated as

$$\mathbf{x}_i = \mathbf{v}_i^{(word)} \oplus \mathbf{v}_i^{(tag)}, \tag{1}$$

$$\mathbf{h}_i = \text{BiLSTM}(\mathbf{x}_1, \ldots, \mathbf{x}_n)_i, \tag{2}$$

where $\oplus$ denotes concatenation. A pair of representations is fed into a biaffine classifier to predict the possibility that there is a dependency arc between these two words. The scores over head words are calculated as

$$\begin{aligned}
\mathbf{s}_i^{(arc)} &= H^{(arc\text{-}head)} W^{(arc)} \mathbf{h}_i^{(arc\text{-}dep)} + H^{(arc\text{-}head)} \mathbf{b}^{(arc)}, \\
y^{(arc)} &= \operatorname*{argmax}_j s_{i,j}^{(arc)},
\end{aligned}$$

where $\mathbf{h}_i^{(arc\text{-}dep)}$ is computed by feeding $\mathbf{h}_i$ into a multi-layer perceptron network (MLP) and $H^{(arc\text{-}head)}$ is the stack of $\mathbf{h}_i^{(arc\text{-}head)}$ which is calculated in the same way as $\mathbf{h}_i^{(arc\text{-}dep)}$ but using another MLP. After getting the head $y^{(arc)}$ word, its relation with the ith word is decided by

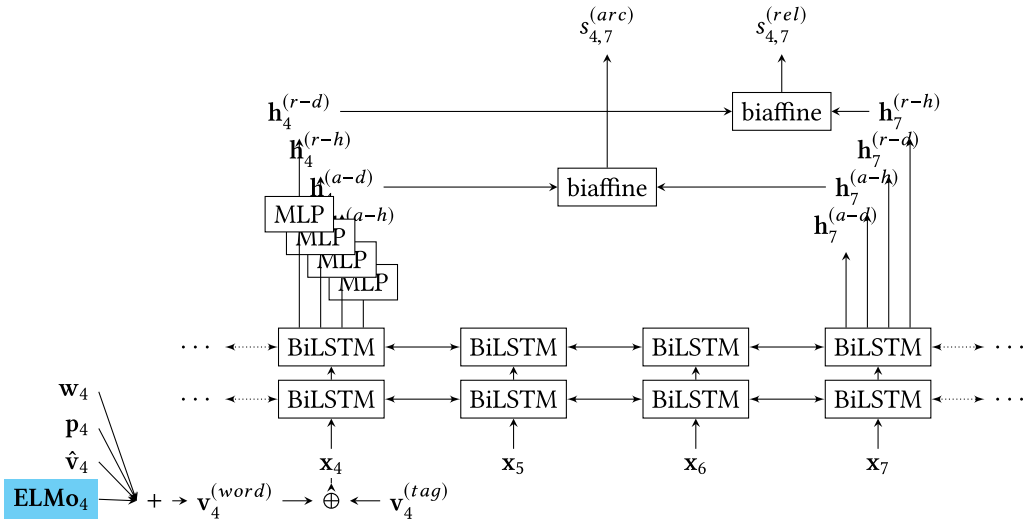

Fig. 3. The framework for deep biaffine parser. We demonstrate how to calculate the arc score ($s^{(arc)}_{4,7}$) and relation score ($s^{(rel)}_{4,7}$) between the fourth and seventh words. Like in Figure 2, we highlight the additional ELMo embeddings in the cyan block.

calculating

$$\mathbf{s}^{(rel)}_i = \left(\mathbf{h}^{(rel-head)}_{y^{(arc)}}\right)^T \mathbf{U}^{(rel)} \mathbf{h}^{(rel-dep)}_i + W^{(rel)} \left(\mathbf{h}^{(rel-dep)}_i \oplus \mathbf{h}^{(rel-head)}_{y^{(arc)}}\right) + \mathbf{b}^{(rel)},$$
$$y^{(rel)} = \operatorname*{argmax}_j s^{(rel)}_{i,j},$$

where $\mathbf{h}^{(rel-head)}$ and $\mathbf{h}^{(rel-dep)}$ are calculated in the same way as $\mathbf{h}^{(arc-dep)}_i$ and $\mathbf{h}^{(arc-head)}_i$. The overall framework for the deep biaffine parser is shown in Figure 3.

This decoding process can lead to cycles in the result. Dozat et al. [12] employed an iterative fixing method on the cycles. We encourage the reader of this article to refer to their paper for more details on training and decoding.

Word Representation. For both the biaffine tagger and parser, Dozat et al. [12] obtained the word representation $\mathbf{v}^{(word)}_i$ by summing a tunable embedding $\mathbf{w}_i$, a fixed word2vec embedding $\mathbf{p}_i$, and an LSTM-encoded character representation $\hat{\mathbf{v}}_i$ as

$$\mathbf{v}^{(word)}_i = \mathbf{w}_i + \mathbf{p}_i + \hat{\mathbf{v}}_i. \tag{3}$$

According to Dozat et al. [12], the tunable embedding $\mathbf{w}_i$ captures the representation of holistic words, the fixed embedding $\mathbf{p}_i$ transfers the knowledge derived from unlabeled data, and the LSTM-encoded character representation $\hat{\mathbf{v}}_i$ further helps to represent the words of languages with rich morphology. The practice of summing multiple features can be treated as a variant of Residual Network [15].

In addition to summation, another widely adopted method for combining multiple features is concatenation. The DenseNet [18] developed the feature concatenation idea in the area of image classification. In previous dependency parsing work [11, 25], both concatenation and summation achieve the state-of-the-art performance.

4 DEEP CONTEXTUALIZED WORD EMBEDDINGS

Deep contextualized word embeddings [10, 35] have shown to be very effective on a range of syntactic and semantic tasks. Peters et al. [35] proposed to derive the contextual representations by first training a *bidirectional language model* on the large-scale corpus; then, using the internal layer activations as the representation for the downstream task. Given a sentence $(w_1, \ldots, w_n)$ and its context-independent word representations $(\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_n)$, the forward language model probability is formalized as

$$p(w_i \mid w_1, \ldots, w_{i-1}) = \frac{1}{Z} \exp\left(W_{w_i} \overrightarrow{\mathbf{h}}_{i-1} + b_{w_i}\right),$$

where $\overrightarrow{\mathbf{h}}_i$ is the last output of the forward context representation function $\overrightarrow{\text{LSTM}}(\tilde{\mathbf{v}}_1, \ldots, \tilde{\mathbf{v}}_i)$ and $Z = \sum_w \exp(W_w \overrightarrow{\mathbf{h}}_{i-1} + b_w)$ is the normalization term. The same definition is used for backward language modeling.

Peters et al. [35] used a character-level CNN for the context-independent representation where $\tilde{\mathbf{v}}_i = \text{CNN}(w_i)$. By using a character-level CNN to represent words, ELMo can output reasonable context-independent word embeddings for arbitrary words, thus has the ability to overcome the OOV word problem. This ability significantly helps UD parsing as we can see in the following section. Peters et al. [35] also used multi-layer LSTMs with skip connections [15] to parameterize $\overrightarrow{\text{LSTM}}$ and $\overleftarrow{\text{LSTM}}$. The multi-layer mechanism can be iteratively described as

$$\overrightarrow{\mathbf{h}}_i^{(k)} = \overrightarrow{\text{LSTM}}^{(k)}\left(\overrightarrow{\mathbf{h}}_1^{(k-1)}, \ldots, \overrightarrow{\mathbf{h}}_i^{(k-1)}\right).$$

Here, $\overrightarrow{\mathbf{h}}_i^{(0)} = \tilde{\mathbf{v}}_i$ and $\overleftarrow{\mathbf{h}}_i^{(0)} = \tilde{\mathbf{v}}_i$. The final contextualized embeddings are computed by weighted pooling of the activations of $L+1$ different layers as $\mathbf{ELMo}_i = \gamma \sum_{k=0}^{L} s_k \cdot (\overrightarrow{\mathbf{h}}_i^{(k)} \oplus \overleftarrow{\mathbf{h}}_i^{(k)})$. In Peters et al. [35], L equals two.

To learn the parameters, Peters et al. [35] proposed to train the model to predict the next words in the large unlabeled corpus. The learning objective is a summation of each word's log probabilities of two directions as

$$\sum_{i=1}^{n} \left(\log p(w_i \mid w_1, \ldots, w_{i-1}) + \log p(w_i \mid w_{i+1}, \ldots, w_n)\right).$$

Using ELMo in Deep Biaffine Parser. The goal of this article is to study ELMo's effect on UD parsing and we use it as an additional word representation in the deep biaffine parser.

Peters et al. [37] studied the problem of whether tuning the pretrained contextualized embeddings in various tasks and suggested freezing ELMo when using it in the model with rich task-specific parameters. Since our tagging and parsing models are rich in the task-specific parameters (including the BiLSTM, MLP, and the biaffine classifier in Figure 3), we follow their suggestion and compute $\mathbf{ELMo}_i$ without tuning its parameters during tagger and parser training.

After getting $\mathbf{ELMo}_i$, we add it as an additional word embedding. As discussed in Section 3, Dozat et al. [12] summed up different features to form the word representation (Equation (3)). We follow their work and treat ELMo as another word representation, thus project it to the same dimension as $\mathbf{v}_i^{(word)}$ and add it into the three existing word features. The calculation of $\mathbf{v}_i^{(word)}$ becomes

$$\mathbf{v}_i^{(word)} = \mathbf{w}_i + \mathbf{p}_i + \hat{\mathbf{v}}_i + W^{(ELMo)} \cdot \mathbf{ELMo}_i \tag{4}$$

for both the tagger and parser (see the cyan part in Figure 2 and Figure 3 for an illustration). We need to note that training the tagger and parser includes tuning $W^{(ELMo)}$ in this setting. To avoid overfitting, we impose a dropout function on the projected vector $W^{(ELMo)} \cdot \mathbf{ELMo}_i$ during training.

Peters et al. [35] proposed to either concatenate ELMo with the task-specific word representation ($\mathbf{x}_i$ in Equation (1) in our case) or with the task-specific context-dependent representation ($\mathbf{h}_i$ in Equation (2) in our case). A slight improvement was observed with more sophisticated usage of ELMo according to their experimental results. However, with the goal of studying ELMo's effect on UD parsing rather than exhausted architecture search for further mediocre improvements, we just use ELMo as an additional word embedding in Dozat et al. [12]'s parser for simplification.

5 EXPERIMENTS

5.1 Settings

We conduct experiments on 57 treebanks across 42 languages released in the CoNLL 2018 universal dependency parsing shared task [41]. We choose the treebanks of the corresponding languages because their unlabeled data were released in the shared task. For all the treebanks, we adopt the standard training/dev/test splits. For the unlabeled data, we randomly sample a set of 20 million words from the raw text and use the sampled data to train our ELMo.[1]

Our experiments include both the POS tagging and UD parsing. For the POS tagging experiments, we use the gold tokenized words as input. In the shared-task data, both universal POS tags (UPOS) and treebank-specific POS tags (XPOS) are provided. We follow Dozat et al. [12] and train the POS tagger to predict UPOS and XPOS via a multi-task learning objective. We only report the UPOS results because the XPOS of some dataset is meaningless.[2] In addition, our experimental results showed similar trends for the UPOS and XPOS.

For the UD parsing experiments, we use both the gold tokenized words and the automatically assigned POS tags as input. The automatic POS tags are obtained by five-way jackknifing using our ELMo enhanced POS tagger. Labeled attachment score (LAS) is used to evaluate the UD parsing performance.

For training the ELMo, we use the *sample softmax* technique to make training on large vocabulary feasible [19]. The major difference between our training method and the standard *sample softmax* is that we use a window of 8,192 words surrounding the target word as negative samples and it shows better performance in our preliminary experiments. The training of ELMo on one language takes roughly 3 days on an NVIDIA P100 GPU. For training the tagger and parser, we use the same hyper-parameter settings as Dozat et al. [12].

In the following comparison, we study the model using ELMo as input (w/ELMo) and the model without ELMo (w/o ELMo).

5.2 Results

Table 1 shows the experimental results for POS tagging and universal dependency parsing. From this table, we can see that ELMo consistently improves the performance on the 57 tested treebanks. More specifically, ELMo improves the UPOS accuracy for 51 out of 57 treebanks with a macro accuracy gain of 0.91 and the macro error reduction of 24% for the POS tagging. For the universal dependency parsing, ELMo improves the LAS for 54 out of 57 treebanks. The macro LAS improvement is 1.11 and the macro error reduction is 7%. These results clearly show that the ELMo can help the POS tagging and universal dependency parsing.

In this table, we compare our ELMo-enhanced tagger and parser with the other state-of-the-art UD parsing systems including (1) the joint POS tagging, lemmatization, and parsing model from UDpipe 2.0 [38] and (2) the BERT initialized multi-lingual multi-task model from UDify [21]. The

[1]We release our pre-trained ELMo at https://github.com/HIT-SCIR/ELMoForManyLangs.
[2]For example, *Japanese-GSD* does not provide XPOS tags and the corresponding fields are filled with underscores (_).

Table 1. The Experimental Results of UD Parsing Comparing Models With and Without ELMo

	w/o ELMo		w/ELMo			w/o ELMo		w/ELMo	
Treebank	UPOS	LAS	UPOS	LAS	Treebank	UPOS	LAS	UPOS	LAS
Arabic-PADT	96.73	82.88	**96.93**	**83.64**	Italian-PoSTWITA	95.93	80.54	**96.82**	**81.77**
Bulgarian-BTB	98.40	90.80	**99.13**	**91.78**	Japanese-GSD	97.76	92.51	**97.96**	**92.70**
Catalan-AnCora	97.37	91.05	**98.89**	**91.40**	Korean-GSD	96.25	83.71	**96.60**	**84.94**
Czech-CAC	99.01	91.22	**99.25**	**91.40**	Korean-Kaist	95.32	**86.62**	**95.68**	86.40
Czech-FicTree	98.55	91.08	**98.57**	**91.57**	Latin-ITTB	**98.55**	**89.04**	98.53	88.90
Czech-PDT	98.99	92.36	**99.13**	**92.39**	Latin-PROIEL	96.23	79.06	**97.03**	**80.49**
Old_Church_Slavonic-PROIEL	**96.61**	81.01	96.47	**82.26**	Latvian-LVTB	93.96	82.43	**96.07**	**83.87**
Danish-DDT	96.97	85.19	**97.95**	**86.34**	Dutch-Alpino	96.09	87.43	**96.35**	**90.21**
German-GSD	94.40	81.26	**94.89**	**81.79**	Dutch-LassySmall	95.78	85.96	**96.61**	**87.51**
Greek-GDT	96.57	88.18	**98.02**	**89.31**	Norwegian-Bokmaal	97.46	90.78	**98.36**	**91.67**
English-EWT	95.96	87.62	**96.74**	**88.27**	Norwegian-Nynorsk	97.19	90.42	**98.20**	**91.33**
English-GUM	93.57	84.08	**96.42**	**86.00**	Polish-LFG	97.90	**95.03**	**98.83**	94.73
English-LinES	96.52	79.68	**97.14**	**80.46**	Polish-SZ	97.24	91.74	**98.32**	**92.18**
Spanish-AnCora	97.78	90.44	**98.84**	**90.86**	Portuguese-Bosque	96.20	89.12	**97.21**	**89.85**
Estonian-EDT	96.26	85.10	**97.49**	**85.48**	Romanian-RRT	97.53	86.90	**97.97**	**86.95**
Basque-BDT	95.22	81.85	**96.48**	**83.56**	Russian-SynTagRus	97.81	92.96	**99.02**	**93.09**
Persian-Seraji	97.22	87.20	**97.93**	**88.15**	Slovak-SNK	93.96	88.11	**97.08**	**90.13**
Finnish-FTB	95.89	89.11	**96.87**	**89.94**	Slovenian-SSJ	97.74	92.01	**98.61**	**92.04**
Finnish-TDT	96.24	87.99	**97.48**	**89.65**	Swedish-LinES	96.39	81.72	**97.38**	**83.21**
French-GSD	97.13	87.93	**97.59**	**88.62**	Swedish-Talbanken	97.60	86.61	**98.09**	**88.26**
French-Sequoia	98.27	89.12	**98.91**	**91.18**	Turkish-IMST	95.66	64.97	**96.65**	**68.03**
French-Spoken	94.24	74.95	**96.31**	**77.79**	Uyghur-UDT	**89.84**	65.32	89.68	**68.13**
Galician-CTG	**97.71**	83.31	**97.71**	**83.45**	Ukrainian-IU	96.20	85.50	**97.74**	**86.40**
Ancient_Greek-Perseus	92.05	75.12	**94.39**	**78.73**	Urdu-UDTB	**94.39**	81.66	94.21	**82.26**
Ancient_Greek-PROIEL	96.86	81.76	**97.85**	**83.75**	Vietnamese-VTB	88.99	60.27	**90.54**	**62.12**
Hebrew-HTB	96.43	85.21	**97.34**	**87.21**	Chinese-GSD	**94.60**	79.70	94.30	**79.85**
Hindi-HDTB	97.24	92.00	**97.60**	**92.19**	**macro average score**			UPOS	LAS
Croatian-SET	97.35	86.51	**98.14**	**87.24**	w/o ELMo			96.15	84.94
Hungarian-Szeged	92.96	76.69	**96.59**	**81.35**	w/ ELMo			**97.07**	**86.04**
Indonesian-GSD	91.82	79.21	**93.76**	**79.03**	UDpipe 2.0 [38]			96.94	85.05
Italian-ISDT	97.89	91.52	**98.41**	**92.48**	UDify [21]			95.88	85.50

The bold number shows the winning model. Our ELMo-enhanced model outperforms that without ELMo on a large proportion of the evaluated treebanks.

results are shown in the bottom right block.[3] From this comparison, ELMo helps the biaffine parser to achieve state-of-the-art performance on the tested treebanks.

We also study the correlation between UPOS improvements and LAS improvements. Their Pearson correlation is 0.5196 with a *p-value* of less than 0.001. This shows the improvements resulted from using ELMo are consistent across our two syntactic tasks.

[3]The evaluation in the original UDpipe 2.0 paper involved sentence segmentation and tokenization. This made the original results not directly comparable. The results we report are extracted from Table 6 of Kondratyuk [21].

6 ANALYSIS

6.1 OOV Rate is More Related to the Performance Gains

Section 5.2 shows that POS tagging and UD parsing can be improved with ELMo. However, the effects of ELMo vary across different treebanks, from a negative effect on *Galician-CTG* to the largest positive effect on *Hungarian-Szeged*.[4] This observation indicates the effects of ELMo can be related to the attributes of treebanks. To gain insight into these attributes, we conduct analysis experiments across the treebanks of different languages to examine how their attributes relate with the performance gains. In this article, we focus on five attributes of each treebank which include the following:

— *Training size*: We use the log number of tokens in the training data to denote the training size.[5]
— *Morphological complexity*: We follow Bentz et al. [3] and use the *word entropy* as a corpus-based evaluation for the *morphological complexity*. According to Bentz et al. [3], word entropy shows good correlation with the morphological complexity defined by linguistics [13, World Atlas of Language Structures project].
— *Nonprojectivity*: We use the percentage of nonprojective arcs in the training data as the evaluation of the *nonprojectivity*.
— *Polysemy rate*: we define a word as a polysemous word if it has multiple UPOS presented in the dataset. We use the proportion of such words as the polysemy rate. Since the contextualized embeddings are designed to cope with the polysemy in words, we expect that the parser using ELMo to be better at the treebank, which is rich in polysemous words.
— *OOV rate*: We define a word as an OOV word as it occurs in the test data but not in the training data. The percentage of such words in the test data is defined as the OOV rate.

Three of these attributes (*training size, morphological complexity, and nonprojectivity*) are inspired by the analysis work of Dozat et al. [12].

We first plot the UPOS/LAS improvements and error reductions against these attributes for each treebank in Figure 4. From this figure, we observe a generally monotonic relation between these attributes and ELMo's performance gains, except for the *nonprojectivity* which seems unrelated. Based on this plot, we calculate two monotonic correlation coefficients (Pearson correlation and Spearman correlation) of these attributes with the performance gains and the results are shown in Table 2. Since the values of these correlations are not fully interpretable, we only use them as an indication of which attribute is more related to the performance gain. From this table, we can see that among all these factors, the *OOV rate* is most correlated with the improvements. We need to note that the size of the training data also presents some correlation with the dependency parsing improvements. However, there is usually more OOV words in a smaller treebank, which means the *training size* and *OOV rate* are correlated.[6] We consider *OOV rate* as a more informative attribute and its correlations are consistent in POS tagging and dependency parsing.

Beyond these attributes, one may argue the potential existence of non-treebank related attributes that correlate with ELMo's performance gains. We follow Ma et al. [25] and McDonald and Nivre [29] to examine three structural attributes including *dependency length*, *distance to root*, and *number of modifier siblings*. The underlying assumption is that the parser with ELMo might favor a certain type of arcs (say better long-distance dependencies because of better context encoding). We plot their relations in Figure 5 along with the relative error reduction. We can see

[4]ELMo leads to a drop of 0.30 on UPOS for *Galician-CTG* and a gain of 4.56 on *Hungarian-Szeged*.
[5]With a base of 10.
[6]In our case, the Pearson correlation between *training size* and *OOV rate* is −0.5477.

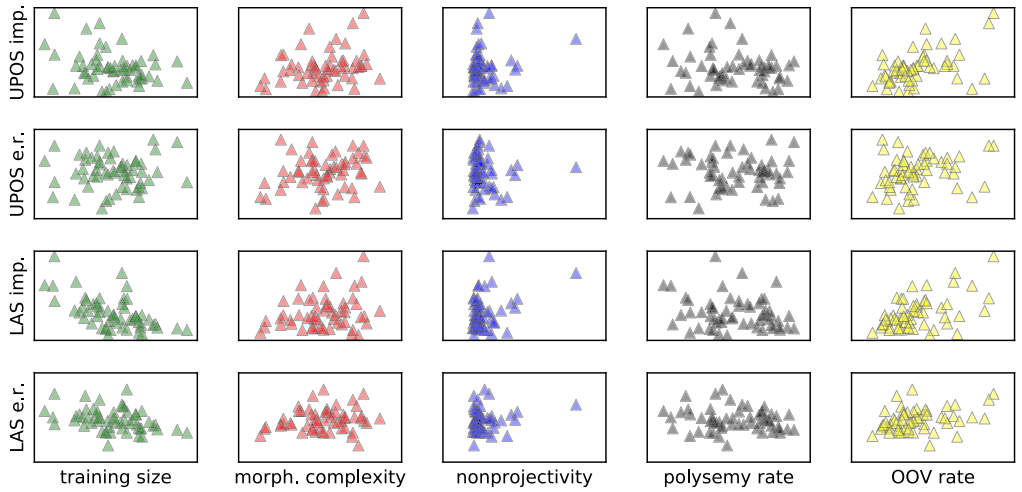

Fig. 4. The relation between five attributes and the ELMo's improvements (*imp.*) and error reduction (*e.r.*) for POS tagging (*UPOS*) and UD parsing (*LAS*). Each point in this figure denotes a treebank.

Table 2. The Pearson/Spearman Correlation between Five Attributes and the Improvements and Error Reduction for Each Treebank

	UPOS		LAS	
metric	*improvement*	*error reduction*	*improvement*	*error reduction*
training size	−0.30 / −0.23	0.03 / −0.05	**−0.58 / −0.56**	**−0.36 / −0.39**
morphological complexity	0.26 / 0.25	0.20 / 0.20	0.24 / 0.19	0.17 / 0.17
nonprojectivity	0.14 / 0.05	−0.07 / −0.09	0.33 / 0.11	0.14 / 0.04
polysemy rate	−0.20 / −0.15	−0.10 / −0.15	−0.17 / −0.14	−0.13 / −0.12
OOV rate	**0.51 / 0.44**	**0.22 / 0.24**	0.52 / 0.43	0.33 / 0.33

Bold number show the highest (absolute) correlation value.

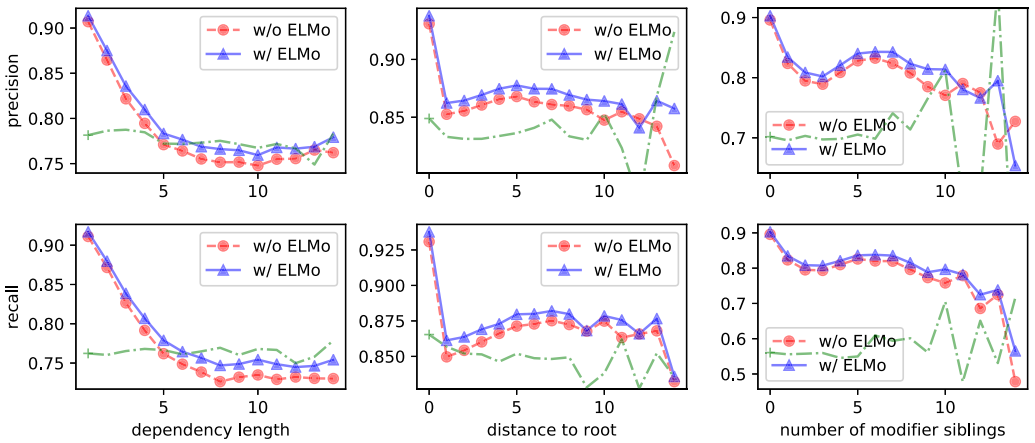

Fig. 5. Dependency arc precision/recall relative to three attributes. The dashed green line shows the error reduction. The limits of y-axis for error reduction are fixed to (0.0, 0.3).

Table 3. A Comparison on the IV and OOV Performance of Two Models

	IV		OOV	
models	UPOS	LAS	UPOS	LAS
w/o ELMo	98.00	87.97	88.50	79.52
w/ ELMo	**98.27**	**88.58**	**93.36**	**81.32**
	13.50%	5.07%	42.26%	8.79%

The last row shows the error reductions led by using ELMo against the baseline without ELMo.

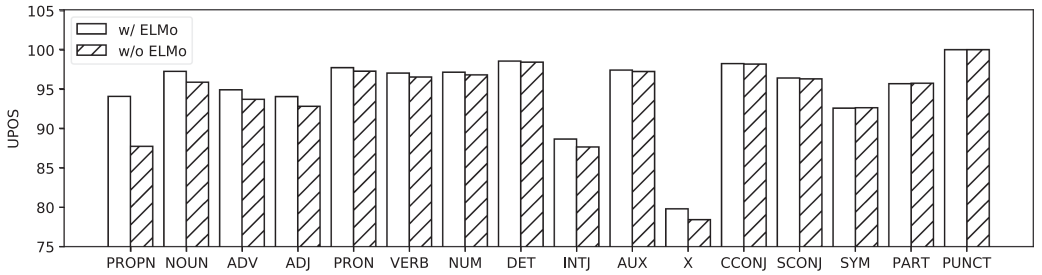

Fig. 6. The POS tag error distributions of our models with and without ELMo.

that the error reductions do not show consistent trends with these attributes, which indicates the weak correlation between performance gains and structural attributes. Thus, we stick to analyzing ELMo's effect of OOV in the following section.

6.2 Detailed Out-of-Vocabulary Words Performance

To further confirm that the improvements that are brought by ELMo mainly happen in OOV words, we compare the model's in-vocabulary-word (IV) and OOV performances on a concatenation of the test data.[7] The results are shown in Table 3. From this table, ELMo brings more improvements to OOV words than IV words on both the POS tagging and dependency parsing. For POS tagging, the absolute improvement is 4.86 and error reduction is 42.26%. For dependency parsing, the absolute improvement is 1.80 and error reduction is 8.79%.

In addition to the detailed OOV performance, we plot the POS error distributions of our models with and without ELMo in Figure 6. In this figure, these two models tie on the tags for function words like *auxiliary word*s (AUX), *coordinating conjunction*s (CCONJ), and *subordinating conjunction*s (SCONJ). But, the model with ELMo clearly wins in the *proper noun*s (PROPN). This result testifies to ELMo's effect on OOV in another direction because OOV word, as an emerging new entity name, happens more with the PROPN tag.

6.3 Memorize or Guess: ELMo Can Guess OOV's Syntactic Functions

OOV words have long been the problem that hinders the performance of NLP systems. The general solution for the OOV words problem is representing a word with high-level abstraction. Such abstraction can be achieved by either (1) memorizing the association between the OOV word and their abstractive symbols (POS, cluster id, and so on) [23] or (2) abstracting the OOV word's context and meta-features [2, 8]. In terms of learning better word abstraction, ELMo is a potential solution for two reasons: (1) The training process of ELMo encounters numerous words, which means that

[7] The proportions of IV and OOV words within the test data are 87.38% and 12.62%.

Table 4. A Comparison on the OOV Word Performance

	recalled (79%)		un-recalled (21%)	
models	UPOS	LAS	UPOS	LAS
w/o ELMo	88.85	79.52	88.29	79.52
w/ ELMo	**93.90**	**81.42**	**92.05**	**80.99**
	45.29%	9.27%	32.11%	7.18%

We categorize OOV words into two types, indicating whether a word is recalled during the learning of ELMo. The number in the brackets represents the proportion of recalled and un-recalled OOV. The last row shows the error reductions as in Table 3.

ELMo "memorizes" words. (2) ELMo uses a CNN to model a word and uses bidirectional LSTM to model context, which means that ELMo "guesses" the word's syntactic functions by contextual and morphological features.

It is expected that ELMo improves the "memorized" OOV words. However, we are interested in the performance of OOV words that are never encountered during ELMo and parser learning, thus calling for properly "guessing." To accomplish this, we categorize an OOV word into two types, indicating whether this word is encountered (recalled) during the learning of ELMo. The recalled OOV performance can be treated as an evaluation of the "memorizing" effects, and the un-recalled OOV performance reflects the "guessing" effects. We show the OOV performance on POS tagging and dependency parsing in Table 4. From this table, we can see that the performances of both recalled and un-recalled OOV words are both higher than those of the model without ELMo, and their improvements are comparable where the error reduction on "memorized" OOV is slightly higher. This result shows that ELMo can guess an OOV's syntactic function as well as memorize it. This should be done through proper abstraction and this ability contributes to the improvements.

6.4 Context or Morphology: ELMo Guesses OOV's Syntactic Function According to Its Morphology

In the previous section, ELMo can "guess" a word's syntactic functions through abstraction. We would further ask how ELMo achieves this, according to OOV's context which is encoded by the bidirectional LSTM or OOV's morphological features which are encoded by character-level word representation.

To track which of these two factors counts more, we compare the ELMo with character CNN (marked as *char ELMo*) with several baselines including the following:

- *FastText*: We replace the Word2vec embeddings with FastText [4] in the *w/o ELMo* baseline since it adopts the character-level encoding scheme and can naturally deal with OOVs. This comparison tests if we can achieve similar performance with static embeddings learned with morphology in mind.
- *character CNN*: We only use the context-independent layer of the char ELMo as the embeddings. This baseline eliminates the contextual effects in ELMo and directly tests if similar performance gain can be achieved without context encoding.
- *word ELMo*: We replace the character-level CNN with vanilla word embeddings and train new contextualized embeddings using the same settings in Section 5.1. This baseline tests if similar performance gain can be achieved by mainly encoding the context.

Table 5. The Comparison with Baselines Using Different Inputs

	w/o context						w/context			
	Word2vec		FastText		char CNN		char ELMo		word ELMo	
Treebank	UPOS	LAS	UPOS	LAS	UPOS	LAS	UPOS	LAS	UPOS	LAS
Dutch-Alpino	93.32	86.73	94.69	86.13	95.65	**88.10**	**96.13**	87.86	94.82	86.73
Dutch-LassySmall	93.11	85.06	95.02	84.95	**96.54**	86.56	96.52	**87.05**	95.33	86.22
English-EWT	94.24	86.71	94.42	86.20	96.14	87.35	**96.21**	**87.53**	95.47	87.10
English-GUM	92.59	83.00	93.16	81.90	95.79	85.84	**95.97**	**86.07**	95.55	85.00
English-LinES	94.73	78.24	95.16	77.78	96.69	79.53	**96.93**	**79.59**	96.27	79.37
French-GSD	95.67	87.07	96.61	86.82	97.36	**87.52**	**97.47**	87.40	96.86	**87.40**
French-Sequoia	97.10	87.83	97.41	87.50	98.27	89.82	**98.62**	**90.12**	98.16	89.64
French-Spoken	92.67	73.90	94.67	74.14	96.27	**75.88**	**96.59**	75.64	95.86	75.64
Hungarian-Szeged	87.09	70.85	92.90	74.67	95.95	79.89	**96.46**	**81.02**	92.90	76.35
Italian-ISDT	96.82	90.18	97.68	89.80	98.05	90.11	**98.24**	**90.85**	97.93	90.46
Italian-PoSTWITA	92.84	78.79	94.25	79.67	95.79	**80.15**	**96.06**	80.01	94.94	79.10
Slovak-SNK	86.27	84.42	92.98	84.35	**96.65**	**86.23**	96.61	86.11	93.63	85.73
Macro average	93.04	82.73	94.91	82.83	96.60	84.75	**96.82**	**84.94**	95.64	84.06
Micro IV	96.80	84.94	96.81	84.73	97.58	86.19	**97.68**	**86.32**	97.42	85.96
error reduction			0.39%	−1.41%	24.41%	8.30%	24.61%	9.17%	19.57%	6.78%
Micro OOV	72.59	72.15	83.69	73.34	90.74	77.38	**91.50**	**77.82**	85.52	74.68
error reduction			40.51%	4.27%	66.20%	18.76%	69.00%	20.35%	47.18%	9.06%
Micro unrecalled OOV	62.83	62.83	75.05	68.23	85.56	72.79	**86.74**	**73.60**	68.48	64.07
error reduction			32.86%	14.52%	61.16%	26.86%	64.32%	28.96%	15.19%	3.32%

Six languages[8] are studied, either because of a large OOV proportion or multiple-treebanks of different sizes presented for the certain language. To eliminate the OOV abstraction from the character model ($\hat{\mathbf{v}}_i$ in Equation (4)) in the biaffine parser, we only use the combination of tuned word vector ($\mathbf{w}$ in Equation (4)) and the fixed embeddings (either static or contextualized) as input. The comparison is shown in Table 5. From this table, we can see that the ELMo with character CNN (*char CNN* and *char ELMo*) consistently outperforms that with vanilla word embeddings (*Word2vec* and word ELMo) and using character CNN leads to significant OOV error reduction. This indicates that the character CNN plays an important role in better abstracting; OOV thus achieves better performance.

By comparing *Word2vec* and *FastText* in Table 5, we observe improvements with properly modeling morphology features in the static embeddings. However, the model with *FastText* lags that of either using character CNN from ELMo or using contextual embeddings. Since FastText is learned on a magnitude larger unlabeled corpus than our ELMo, we attribute the better morphology representation of ELMo's character CNN to the scheme of language modeling.

Another notable finding is that the model using the context-independent layer of ELMo achieves similar performance with the model that uses full ELMo and it outperforms the word ELMo. We further investigate the part that requires contextual encoding. Polysemy is a typical problem that requires modeling context and we evaluate the polysemous/monosemous words performance of the ELMo model. We use whether a word has multiple POS in the training data as a proxy for judging its polysemy and the parsing performance is shown in Table 6.[9] From this table, we observe

[8]Including *Dutch*, *English*, *French*, *Hungarian*, *Italian*, and *Slovak*.

[9]Since we use POS tags as an indicator for polysemy, we do not include UPOS results concerning the bias.

Table 6. A Comparison of the Polysemous/ Monosemous Word Performance

model	Polysemous	Monosemous
w/o ELMo	88.90	87.84
w/ELMo	**89.47**	**88.12**
	5.13%	5.13%

We determine a word as polysemy when it has multiple POS in the training data. The last row shows the error reductions.

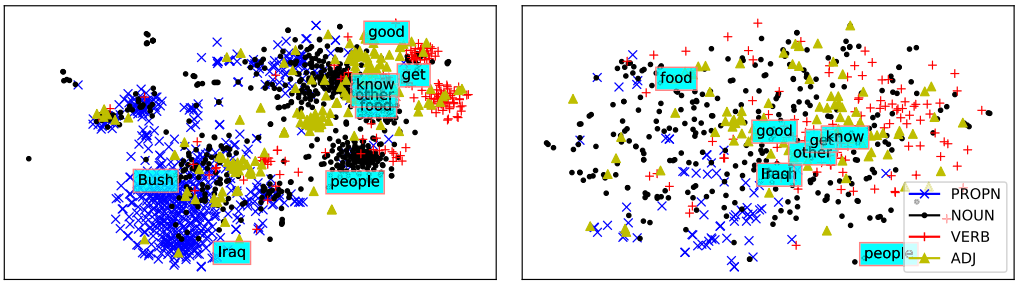

Fig. 7. Visualization of the OOV word embeddings from ELMo (left) and Word2vec (right) along with the prototype word for 4 POS tags for content words. The left figure presents more words than the right one because ELMo can produce embeddings for any word, while Word2vec only yields embeddings for encountered words, and some OOV words are not recalled.

the identical error reduction on polysemous and monosemic words. This indicates that although designed to represent polysemy, the by-product of better context-independent word representation from ELMo also helps to achieve better performance.

6.5 Visualization

As discussed in the previous section, character CNN plays an important role. To get a view of the character-level word representation, we visualize its output on OOV words using T-SNE [26]. We also plot some prototype words of certain POS tags.[10] The results are shown in Figure 7. The colors of the points represent the POS tags for corresponding words.[11] From this figure, we can see that the embeddings of different OOV words are clustered by their POS tags and this shows the embeddings from ELMo have the ability to capture syntactic properties of words, especially the rare words. Additional comparison with the embeddings from Word2vec shows that the Word2vec embeddings for the same POS tag scatter in different places in the embedding space, which do not capture syntactic similarity between words.

6.6 Low-Resource Parsing Simulation

All our analyses point to that the ELMo improves the performance of syntactic NLP tasks by improving the OOV words. A typical scenario that presents severe OOV is the data-insufficient NLP problems in low-resource, one-shot, or even zero-shot settings. It's promising to use ELMo for these tasks to achieve better performance. To test this hypothesis, we simulate the low-resource

[10]The OOV words are extracted from *UD_English-EWT* data. We use *Bush* and *Iraq* as the prototype words for PROPN, *people* and *food* for NOUN, *good* and *other* for ADJ, and *know* and *get* for VERB.
[11]About 97% of the OOV words have mono POS tag.

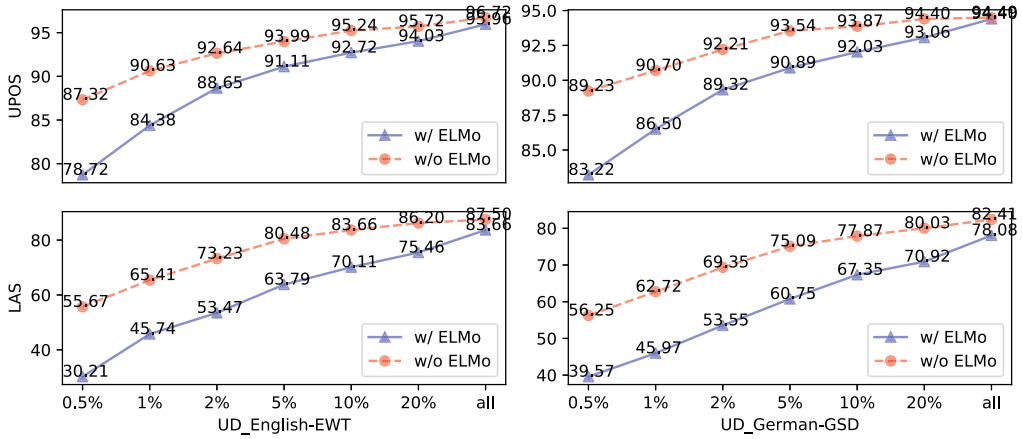

Fig. 8. The simulation results on low-resource settings.

settings by only using 0.5%, 1%, 2%, 5%, 10%, and 20% of the UD_English-EWT and UD_German-GSD training data and compare the model with and without ELMo's performance. The plot is shown in Figure 8.[12] From this figure, we can see that the system benefits more from ELMo when there is only a small proportion of training data. This observation confirms our hypothesis that contextual vectors help in low-resource settings. According to Howard and Ruder [17], the pre-trained language model helps to train a model with only a small number of labeled data because of its transfer ability. In this article, we further attribute this transfer ability to the word abstraction from ELMo.

7 CONCLUSIONS

In this article, we use ELMo as an additional word embedding which leads to consistent improvements on the syntactic problems—universal POS tagging and dependency parsing. Further analysis indicates the improvements mainly result from better OOV word abstraction to which the character-level word representation in the ELMo contributes a lot. The analysis results hint that ELMo can be a promising technique for NLP tasks whose data are insufficient and a simulation of the low-resource situation confirms that.

REFERENCES

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proc. of Coling*. http://www.aclweb.org/anthology/C18-1139.

[2] Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with LSTMs. In *Proc. of EMNLP*.

[3] Christian Bentz, Tatyana Ruzsics, Alexander Koplenig, and Tanja Samardzic. 2016. A comparison between morphological complexity measures: Typological data vs. language corpora. In *Proc. of the Workshop on Computational Linguistics for Linguistic Complexity (CL4LC'16)*.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5, 1 (2017), 135–146. https://www.aclweb.org/anthology/Q17-1010.

[5] Samuel R. Bowman, Ellie Pavlick, Edouard Grave, Benjamin Van Durme, Alex Wang, Jan Hula, Patrick Xia, Raghavendra Pappagari, R. Thomas McCoy, Roma Patel, Najoung Kim, Ian Tenney, Yinghui Huang, Katherin Yu, Shuning Jin,

[12] To simulate the real world situation, we assume that we cannot access a high-quality POS tagger for low-resource parsing, thus we only use words as input for the parsing experiments. This made the result in Figure 8 with full training data not identical to that in Table 1.

and Berlin Chen. 2018. Looking for ELMo's friends: Sentence-level pretraining beyond language modeling. *CoRR* abs/1812.10860 (2018). arxiv:1812.10860 http://arxiv.org/abs/1812.10860.

[6] Wenliang Chen, Jun'ichi Kazama, Kiyotaka Uchimoto, and Kentaro Torisawa. 2009. Improving dependency parsing with subtrees from auto-parsed data. In *Proc. of EMNLP*.

[7] W. Chen, M. Zhang, and Y. Zhang. 2015. Distributed feature representations for dependency parsing. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23, 3 (March 2015), 451–460. DOI: https://doi.org/10.1109/TASLP.2014.2365359

[8] Michael Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*. DOI: https://doi.org/10.3115/1118693.1118694

[9] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proc. of ICML*, Vol. 70.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *CoRR* abs/1810.04805 (2018). arxiv:1810.04805

[11] Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR* abs/1611.01734 (2016). arxiv:1611.01734

[12] Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proc. of CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

[13] Matthew S. Dryer and Martin Haspelmath (Eds.). 2013. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

[14] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. of ACL*.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR* abs/1512.03385 (2015). arxiv:1512.03385 http://arxiv.org/abs/1512.03385.

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (Nov. 1997). DOI: https://doi.org/10.1162/neco.1997.9.8.1735

[17] Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proc. of ACL*.

[18] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. 2016. Densely connected convolutional networks. *CoRR* abs/1608.06993 (2016). arxiv:1608.06993 http://arxiv.org/abs/1608.06993.

[19] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proc. of ACL*.

[20] Vidur Joshi, Matthew Peters, and Mark Hopkins. 2018. Extending a parser to distant domains using a few dozen partially annotated examples. In *Proc. of ACL*.

[21] Daniel Kondratyuk. 2019. 75 languages, 1 model: Parsing universal dependencies universally. *CoRR* abs/1904.02099 (2019). arxiv:1904.02099 http://arxiv.org/abs/1904.02099.

[22] Kenton Lee, Luheng He, and Luke Zettlemoyer. 2018. Higher-order coreference resolution with coarse-to-fine inference. In *Proc. of NAACL*.

[23] Percy Liang. 2005. *Semi-supervised Learning for Natural Language*. Master's Thesis. MIT.

[24] P. Liang, H. Daumé, and D. Klein. 2008. Structure compilation: Trading structure for features. In *Proc. of the 25th International Conference on Machine Learning (ICML'08)*, 592–599.

[25] Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proc. of ACL*. https://www.aclweb.org/anthology/P18-1130.

[26] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9 (2008).

[27] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn treebank. *Computational Linguistic* 19, 2 (1993), 313–330.

[28] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NIPS 30*. 6294–6305.

[29] Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP*.

[30] Ryan T. McDonald and Fernando C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*.

[31] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *CoRR* abs/1310.4546 (2013).

[32] Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics* 34, 4 (2008).

[33] 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 *Proc. of LREC-2016*.

[34] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proc. of EMNLP*.

[35] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.

[36] Matthew Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018. Dissecting contextual word embeddings: Architecture and representation. In *Proc. of EMNLP*.

[37] Matthew Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? Adapting pretrained representations to diverse tasks. *CoRR* abs/1903.05987 (2019). arxiv:1903.05987 http://arxiv.org/abs/1903.05987.

[38] Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proc. of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. https://www.aclweb.org/anthology/K18-2020.

[39] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R. Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019. What do you learn from context? Probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJzSgnRcKX.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS 30*. 5998–6008.

[41] Daniel Zeman, Filip Ginter, Jan Hajič, Joakim Nivre, Martin Popel, and Milan Straka. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proc. of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*.

[42] Kelly Zhang and Samuel Bowman. 2018. Language modeling teaches you more than translation does: Lessons learned through auxiliary syntactic task analysis. In *Proc. of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. http://www.aclweb.org/anthology/W18-5448.

[43] Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proc. of ACL*.

Received March 2019; revised April 2019; accepted April 2019