Stack-Pointer Networks for Dependency Parsing

Xuezhe Ma
Carnegie Mellon University
xuezhem@cs.cmu.edu

Zecong Hu*
Tsinghua University
huzecong@gmail.com

Jingzhou Liu
Carnegie Mellon University
liujingzhou@cs.cmu.edu

Nanyun Peng
University of Southern California
npeng@isi.edu

Graham Neubig and **Eduard Hovy**
Carnegie Mellon University
{gneubig, ehovy}@cs.cmu.edu

Abstract

We introduce a novel architecture for dependency parsing: *stack-pointer networks* (**StackPtr**). Combining pointer networks (Vinyals et al., 2015) with an internal stack, the proposed model first reads and encodes the whole sentence, then builds the dependency tree top-down (from root-to-leaf) in a depth-first fashion. The stack tracks the status of the depth-first search and the pointer networks select one child for the word at the top of the stack at each step. The StackPtr parser benefits from the information of the whole sentence and all previously derived subtree structures, and removes the left-to-right restriction in classical transition-based parsers. Yet, the number of steps for building any (including non-projective) parse tree is linear in the length of the sentence just as other transition-based parsers, yielding an efficient decoding algorithm with $O(n^2)$ time complexity. We evaluate our model on 29 treebanks spanning 20 languages and different dependency annotation schemas, and achieve state-of-the-art performance on 21 of them.

1 Introduction

Dependency parsing, which predicts the existence and type of linguistic dependency relations between words, is a first step towards deep language understanding. Its importance is widely recognized in the natural language processing (NLP) community, with it benefiting a wide range of NLP applications, such as coreference resolution (Ng, 2010; Durrett and Klein, 2013; Ma et al., 2016), sentiment analysis (Tai et al., 2015), machine translation (Bastings et al., 2017), information extraction (Nguyen et al., 2009; Angeli et al., 2015; Peng et al., 2017), word sense disambiguation (Fauceglia et al., 2015), and low-resource languages processing (McDonald et al., 2013; Ma and Xia, 2014). There are two dominant approaches to dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007): local and greedy *transition-based* algorithms (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; Zhang and Nivre, 2011; Chen and Manning, 2014), and the globally optimized *graph-based* algorithms (Eisner, 1996; McDonald et al., 2005a,b; Koo and Collins, 2010).

Transition-based dependency parsers read words sequentially (commonly from left-to-right) and build dependency trees incrementally by making series of multiple choice decisions. The advantage of this formalism is that the number of operations required to build any projective parse tree is linear with respect to the length of the sentence. The challenge, however, is that the decision made at each step is based on local information, leading to error propagation and worse performance compared to graph-based parsers on root and long dependencies (McDonald and Nivre, 2011). Previous studies have explored solutions to address this challenge. Stack LSTMs (Dyer et al., 2015; Ballesteros et al., 2015, 2016) are capable of learning representations of the parser state that are sensitive to the complete contents of the parser's state. Andor et al. (2016) proposed a globally normalized transition model to replace the locally normalized classifier. However, the parsing accuracy is still behind state-of-the-art graph-based parsers (Dozat and Manning, 2017).

Graph-based dependency parsers, on the other hand, learn scoring functions for parse trees and perform exhaustive search over all possible trees for a sentence to find the globally highest scoring

*Work done while at Carnegie Mellon University.

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers), pages 1403–1414
Melbourne, Australia, July 15 - 20, 2018.

tree. Incorporating this global search algorithm with distributed representations learned from neural networks, neural graph-based parsers (Kiperwasser and Goldberg, 2016; Wang and Chang, 2016; Kuncoro et al., 2016; Dozat and Manning, 2017) have achieved the state-of-the-art accuracies on a number of treebanks in different languages. Nevertheless, these models, while accurate, are usually slow (e.g. decoding is $O(n^3)$ time complexity for first-order models (McDonald et al., 2005a,b) and higher polynomials for higher-order models (McDonald and Pereira, 2006; Koo and Collins, 2010; Ma and Zhao, 2012b,a)).

In this paper, we propose a novel neural network architecture for dependency parsing, *stack-pointer networks* (**StackPtr**). StackPtr is a transition-based architecture, with the corresponding asymptotic efficiency, but still maintains a global view of the sentence that proves essential for achieving competitive accuracy. Our StackPtr parser has a pointer network (Vinyals et al., 2015) as its backbone, and is equipped with an internal stack to maintain the order of head words in tree structures. The StackPtr parser performs parsing in an incremental, top-down, depth-first fashion; at each step, it generates an arc by assigning a child for the head word at the top of the internal stack. This architecture makes it possible to capture information from the whole sentence and all the previously derived subtrees, while maintaining a number of parsing steps linear in the sentence length.

We evaluate our parser on 29 treebanks across 20 languages and different dependency annotation schemas, and achieve state-of-the-art performance on 21 of them. The contributions of this work are summarized as follows:

(i) We propose a neural network architecture for dependency parsing that is simple, effective, and efficient.

(ii) Empirical evaluations on benchmark datasets over 20 languages show that our method achieves state-of-the-art performance on 21 different treebanks[1].

(iii) Comprehensive error analysis is conducted to compare the proposed method to a strong graph-based baseline using biaffine attention (Dozat and Manning, 2017).

[1]Source code is publicly available at https://github.com/XuezheMax/NeuroNLP2

2 Background

We first briefly describe the task of dependency parsing, setup the notation, and review Pointer Networks (Vinyals et al., 2015).

2.1 Dependency Parsing and Notations

Dependency trees represent syntactic relationships between words in the sentences through labeled directed edges between head words and their dependents. Figure 1 (a) shows a dependency tree for the sentence, "But there were no buyers".

In this paper, we will use the following notation:

Input: $\mathbf{x} = \{w_1, \ldots, w_n\}$ represents a generic sentence, where w_i is the ith word.

Output: $\mathbf{y} = \{p_1, p_2, \cdots, p_k\}$ represents a generic (possibly non-projective) dependency tree, where each path $p_i = \$, w_{i,1}, w_{i,2}, \cdots, w_{i,l_i}$ is a sequence of words from the root to a leaf. "$" is an universal virtual root that is added to each tree.

Stack: σ denotes a stack configuration, which is a sequence of words. We use $\sigma|w$ to represent a stack configuration that pushes word w into the stack σ.

Children: $\mathrm{ch}(w_i)$ denotes the list of all the children (modifiers) of word w_i.

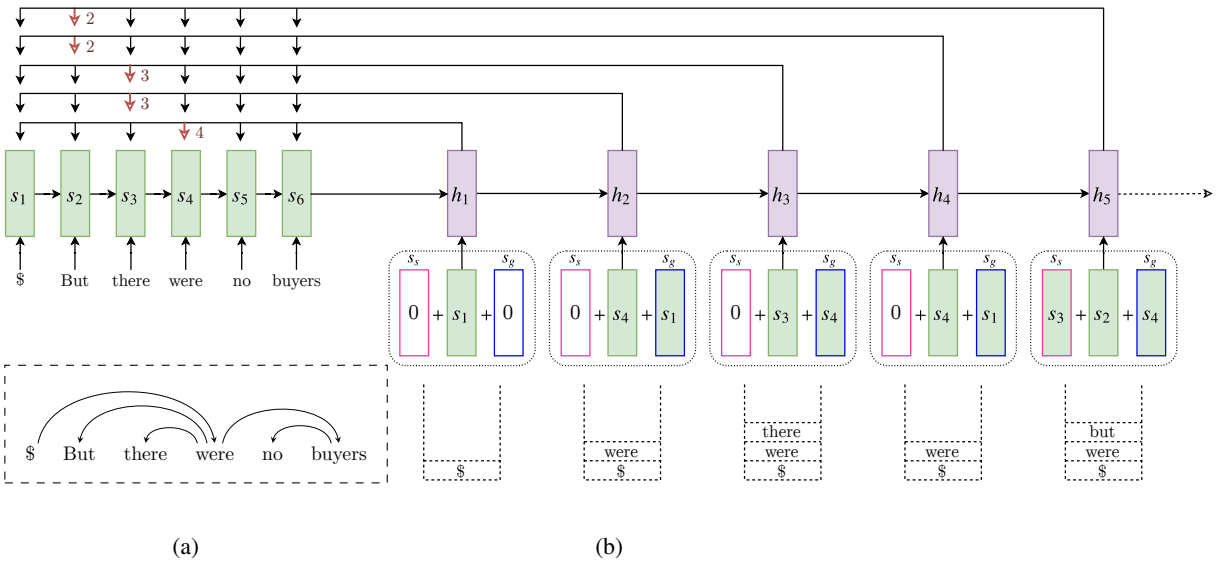
2.2 Pointer Networks

Pointer Networks (Ptr-Net) (Vinyals et al., 2015) are a variety of neural network capable of learning the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. This model cannot be trivially expressed by standard sequence-to-sequence networks (Sutskever et al., 2014) due to the variable number of input positions in each sentence. Ptr-Net solves the problem by using attention (Bahdanau et al., 2015; Luong et al., 2015) as a pointer to select a member of the input sequence as the output.

Formally, the words of the sentence $\mathbf{x}$ are fed one-by-one into the encoder (a multiple-layer bi-directional RNN), producing a sequence of *encoder hidden states* s_i. At each time step t, the decoder (a uni-directional RNN) receives the input from last step and outputs *decoder hidden state* h_t. The *attention vector* a^t is calculated as follows:

$$\begin{aligned} e_i^t &= score(h_t, s_i) \\ a^t &= softmax(e^t) \end{aligned} \quad (1)$$

where $score(\cdot, \cdot)$ is the *attention scoring function*, which has several variations such as dot-product,



(a) (b)

Figure 1: Neural architecture for the STACKPTR network, together with the decoding procedure of an example sentence. The BiRNN of the encoder is elided for brevity. For the inputs of decoder at each time step, vectors in red and blue boxes indicate the sibling and grandparent.

concatenation, and biaffine (Luong et al., 2015). PTR-NET regards the attention vector a^t as a probability distribution over the source words, i.e. it uses a_i^t as pointers to select the input elements.

3 Stack-Pointer Networks

3.1 Overview

Similarly to PTR-NET, STACKPTR first reads the whole sentence and encodes each word into the encoder hidden state s_i. The internal stack σ is always initialized with the root symbol \$. At each time step t, the decoder receives the input vector corresponding to the top element of the stack σ (the head word w_p where p is the word index), generates the hidden state h_t, and computes the attention vector a^t using Eq. (1). The parser chooses a specific position c according to the attention scores in a^t to generate a new dependency arc (w_h, w_c) by selecting w_c as a child of w_h. Then the parser pushes w_c onto the stack, i.e. $\sigma \rightarrow \sigma|w_c$, and goes to the next step. At one step if the parser points w_h to itself, i.e. $c = h$, it indicates that all children of the head word w_h have already been selected. Then the parser goes to the next step by popping w_h out of σ.

At test time, in order to guarantee a valid dependency tree containing all the words in the input sentences exactly once, the decoder maintains a list of "available" words. At each decoding step, the parser selects a child for the current head word, and removes the child from the list of available words to make sure that it cannot be selected as a child of other head words.

For head words with multiple children, it is possible that there is more than one valid selection for each time step. In order to define a deterministic decoding process to make sure that there is only one ground-truth choice at each step (which is necessary for simple maximum likelihood estimation), a predefined order for each $\mathrm{ch}(w_i)$ needs to be introduced. The predefined order of children can have different alternatives, such as left-to-right or inside-out[2]. In this paper, we adopt the inside-out order[3] since it enables us to utilize second-order *sibling* information, which has been proven beneficial for parsing performance (McDonald and Pereira, 2006; Koo and Collins, 2010) (see § 3.4 for details). Figure 1 (b) depicts the architecture of STACKPTR and the decoding procedure for the example sentence in Figure 1 (a).

3.2 Encoder

The encoder of our parsing model is based on the bi-directional LSTM-CNN architecture (BLSTM-CNNs) (Chiu and Nichols, 2016; Ma and Hovy, 2016) where CNNs encode character-level information of a word into its character-level repre-

[2]Order the children by the distances to the head word on the left side, then the right side.

[3]We also tried left-to-right order which obtained worse parsing accuracy than inside-out.

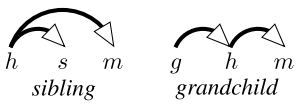
sentation and BLSTM models context information of each word. Formally, for each word, the CNN, with character embeddings as inputs, encodes the character-level representation. Then the character-level representation vector is concatenated with the word embedding vector to feed into the BLSTM network. To enrich word-level information, we also use POS embeddings. Finally, the encoder outputs a sequence of hidden states s_i.

3.3 Decoder

The decoder for our parser is a uni-directional LSTM. Different from previous work (Bahdanau et al., 2015; Vinyals et al., 2015) which uses word embeddings of the previous word as the input to the decoder, our decoder receives the encoder hidden state vector (s_i) of the top element in the stack σ (see Figure 1 (b)). Compared to word embeddings, the encoder hidden states contain more contextual information, benefiting both the training and decoding procedures. The decoder produces a sequence of decoder hidden states h_i, one for each decoding step.

3.4 Higher-order Information

As mentioned before, our parser is capable of utilizing higher-order information. In this paper, we incorporate two kinds of higher-order structures — *grandparent* and *sibling*. A sibling structure is a head word with two successive modifiers, and a grandparent structure is a pair of dependencies connected head-to-tail:



To utilize higher-order information, the decoder's input at each step is the sum of the encoder hidden states of three words:

$$\beta_t = s_h + s_g + s_s$$

where β_t is the input vector of decoder at time t and h, g, s are the indices of the head word and its grandparent and sibling, respectively. Figure 1 (b) illustrates the details. Here we use the element-wise sum operation instead of concatenation because it does not increase the dimension of the input vector β_t, thus introducing no additional model parameters.

3.5 Biaffine Attention Mechanism

For attention score function (Eq. (1)), we adopt the biaffine attention mechanism (Luong et al., 2015; Dozat and Manning, 2017):

$$e_i^t = h_t^T \mathbf{W} s_i + \mathbf{U}^T h_t + \mathbf{V}^T s_i + \mathbf{b}$$

where $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}$ are parameters, denoting the weight matrix of the bi-linear term, the two weight vectors of the linear terms, and the bias vector.

As discussed in Dozat and Manning (2017), applying a multilayer perceptron (MLP) to the output vectors of the BLSTM before the score function can both reduce the dimensionality and overfitting of the model. We follow this work by using a one-layer perceptron to s_i and h_i with elu (Clevert et al., 2015) as its activation function.

Similarly, the dependency label classifier also uses a biaffine function to score each label, given the head word vector h_t and child vector s_i as inputs. Again, we use MLPs to transform h_t and s_i before feeding them into the classifier.

3.6 Training Objectives

The StackPtr parser is trained to optimize the probability of the dependency trees given sentences: $P_\theta(\mathbf{y}|\mathbf{x})$, which can be factorized as:

$$\begin{aligned} P_\theta(\mathbf{y}|\mathbf{x}) &= \prod_{i=1}^{k} P_\theta(p_i|p_{<i}, \mathbf{x}) \\ &= \prod_{i=1}^{k} \prod_{j=1}^{l_i} P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, \mathbf{x}), \end{aligned} \tag{2}$$

where θ represents model parameters. $p_{<i}$ denotes the preceding paths that have already been generated. $c_{i,j}$ represents the jth word in p_i and $c_{i,<j}$ denotes all the proceeding words on the path p_i. Thus, the StackPtr parser is an autoregressive model, like sequence-to-sequence models, but it factors the distribution according to a top-down tree structure as opposed to a left-to-right chain. We define $P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, \mathbf{x}) = a^t$, where attention vector a^t (of dimension n) is used as the distribution over the indices of words in a sentence.

Arc Prediction Our parser is trained by optimizing the conditional likelihood in Eq (2), which is implemented as the cross-entropy loss.

Label Prediction We train a separated multi-class classifier in parallel to predict the dependency labels. Following Dozat and Manning (2017), the classifier takes the information of the

head word and its child as features. The label classifier is trained simultaneously with the parser by optimizing the sum of their objectives.

3.7 Discussion

Time Complexity. The number of decoding steps to build a parse tree for a sentence of length n is $2n-1$, linear in n. Together with the attention mechanism (at each step, we need to compute the attention vector a^t, whose runtime is $O(n)$), the time complexity of decoding algorithm is $O(n^2)$, which is more efficient than graph-based parsers that have $O(n^3)$ or worse complexity when using dynamic programming or maximum spanning tree (MST) decoding algorithms.

Top-down Parsing. When humans comprehend a natural language sentence, they arguably do it in an incremental, left-to-right manner. However, when humans consciously annotate a sentence with syntactic structure, they rarely ever process in fixed left-to-right order. Rather, they start by reading the whole sentence, then seeking the main predicates, jumping back-and-forth over the sentence and recursively proceeding to the subtree structures governed by certain head words. Our parser follows a similar kind of annotation process: starting from reading the whole sentence, and processing in a top-down manner by finding the main predicates first and only then search for sub-trees governed by them. When making latter decisions, the parser has access to the entire structure built in earlier steps.

3.8 Implementation Details

Pre-trained Word Embeddings. For all the parsing models in different languages, we initialize word vectors with pretrained word embeddings. For Chinese, Dutch, English, German and Spanish, we use the structured-skipgram (Ling et al., 2015) embeddings. For other languages we use Polyglot embeddings (Al-Rfou et al., 2013).

Optimization. Parameter optimization is performed with the Adam optimizer (Kingma and Ba, 2014) with $\beta_1 = \beta_2 = 0.9$. We choose an initial learning rate of $\eta_0 = 0.001$. The learning rate η is annealed by multiplying a fixed decay rate $\rho = 0.75$ when parsing performance stops increasing on validation sets. To reduce the effects of "gradient exploding", we use gradient clipping of 5.0 (Pascanu et al., 2013).

Dropout Training. To mitigate overfitting, we apply dropout (Srivastava et al., 2014; Ma et al., 2017). For BLSTM, we use recurrent dropout (Gal and Ghahramani, 2016) with a drop rate of 0.33 between hidden states and 0.33 between layers. Following Dozat and Manning (2017), we also use embedding dropout with a rate of 0.33 on all word, character, and POS embeddings.

Hyper-Parameters. Some parameters are chosen from those reported in Dozat and Manning (2017). We use the same hyper-parameters across the models on different treebanks and languages, due to time constraints. The details of the chosen hyper-parameters for all experiments are summarized in Appendix A.

4 Experiments

4.1 Setup

We evaluate our StackPtr parser mainly on three treebanks: the English Penn Treebank (PTB version 3.0) (Marcus et al., 1993), the Penn Chinese Treebank (CTB version 5.1) (Xue et al., 2002), and the German CoNLL 2009 corpus (Hajič et al., 2009). We use the same experimental settings as Kuncoro et al. (2016).

To make a thorough empirical comparison with previous studies, we also evaluate our system on treebanks from CoNLL shared task and the Universal Dependency (UD) Treebanks[4]. For the CoNLL Treebanks, we use the English treebank from CoNLL-2008 shared task (Surdeanu et al., 2008) and all 13 treebanks from CoNLL-2006 shared task (Buchholz and Marsi, 2006). The experimental settings are the same as Ma and Hovy (2015). For UD Treebanks, we select 12 languages. The details of the treebanks and experimental settings are in § 4.5 and Appendix B.

Evaluation Metrics Parsing performance is measured with five metrics: unlabeled attachment score (UAS), labeled attachment score (LAS), unlabeled complete match (UCM), labeled complete match (LCM), and root accuracy (RA). Following previous work (Kuncoro et al., 2016; Dozat and Manning, 2017), we report results excluding punctuations for Chinese and English. For each experiment, we report the mean values with corresponding standard deviations over 5 repetitions.

[4] http://universaldependencies.org/

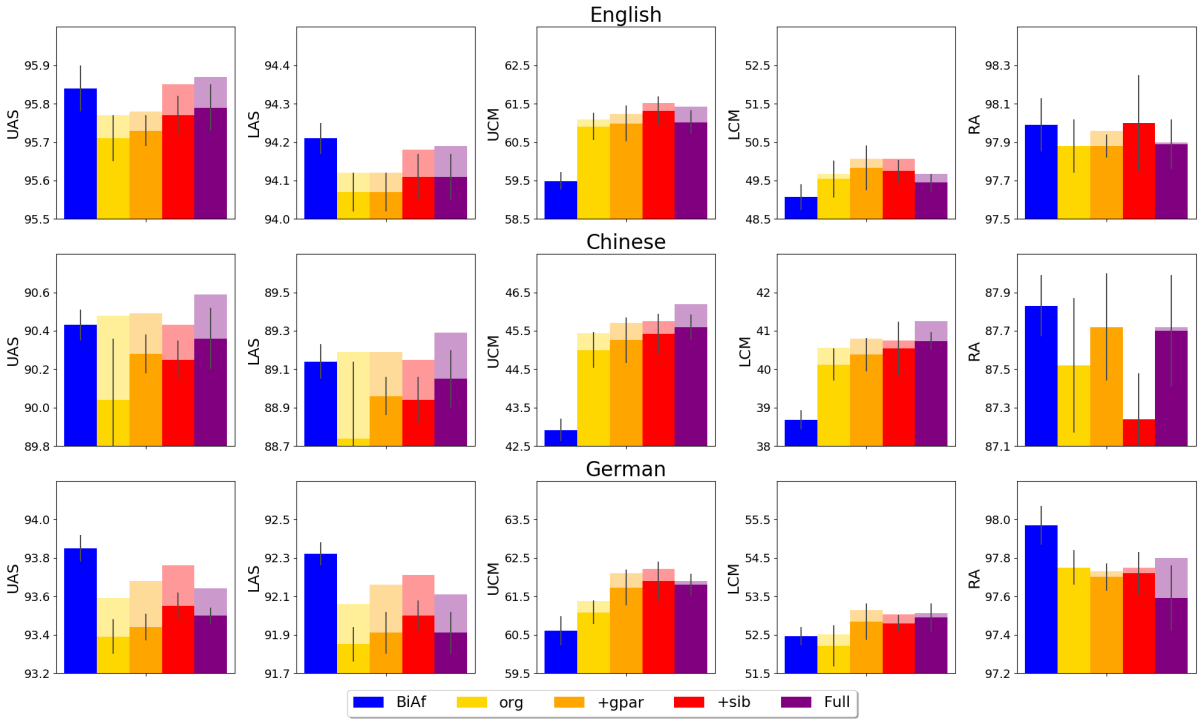


Figure 2: Parsing performance of different variations of our model on the test sets for three languages, together with baseline BIAF. For each of our STACKPTR models, we perform decoding with beam size equal to 1 and 10. The improvements of decoding with beam size 10 over 1 are presented by stacked bars with light colors.

Baseline For fair comparison of the parsing performance, we re-implemented the graph-based Deep Biaffine (BIAF) parser (Dozat and Manning, 2017), which achieved state-of-the-art results on a wide range of languages. Our re-implementation adds character-level information using the same LSTM-CNN encoder as our model (§ 3.2) to the original BIAF model, which boosts its performance on all languages.

4.2 Main Results

We first conduct experiments to demonstrate the effectiveness of our neural architecture by comparing with the strong baseline BIAF. We compare the performance of four variations of our model with different decoder inputs — Org, +gpar, +sib and Full — where the Org model utilizes only the encoder hidden states of head words, while the +gpar and +sib models augments the original one with grandparent and sibling information, respectively. The Full model includes all the three information as inputs.

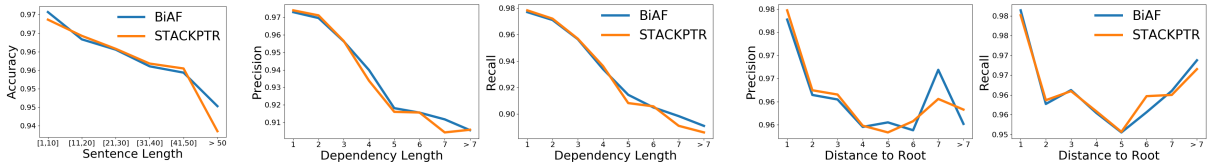
Figure 2 illustrates the performance (five metrics) of different variations of our STACKPTR parser together with the results of baseline BIAF re-implemented by us, on the test sets of the three languages. On UAS and LAS, the Full variation of STACKPTR with decoding beam size 10 outperforms BIAF on Chinese, and obtains competitive performance on English and German. An interesting observation is that the Full model achieves the best accuracy on English and Chinese, while performs slightly worse than +sib on German. This shows that the importance of higher-order information varies in languages. On LCM and UCM, STACKPTR significantly outperforms BIAF on all languages, showing the superiority of our parser on complete sentence parsing. The results of our parser on RA are slightly worse than BIAF. More details of results are provided in Appendix C.

4.3 Comparison with Previous Work

Table 1 illustrates the UAS and LAS of the four versions of our model (with decoding beam size 10) on the three treebanks, together with previous top-performing systems for comparison. Note that the results of STACKPTR and our re-implementation of BIAF are the average of 5 repetitions instead of a single run. Our Full model significantly outperforms all the transition-based parsers on all three languages, and achieves better results than most graph-based parsers. Our

System		English		Chinese		German	
		UAS	LAS	UAS	LAS	UAS	LAS
Chen and Manning (2014)	T	91.8	89.6	83.9	82.4	–	–
Ballesteros et al. (2015)	T	91.63	89.44	85.30	83.72	88.83	86.10
Dyer et al. (2015)	T	93.1	90.9	87.2	85.7	–	–
Bohnet and Nivre (2012)	T	93.33	91.22	87.3	85.9	91.4	89.4
Ballesteros et al. (2016)	T	93.56	91.42	87.65	86.21	–	–
Kiperwasser and Goldberg (2016)	T	93.9	91.9	87.6	86.1	–	–
Weiss et al. (2015)	T	94.26	92.41	–	–	–	–
Andor et al. (2016)	T	94.61	92.79	–	–	90.91	89.15
Kiperwasser and Goldberg (2016)	G	93.1	91.0	86.6	85.1	–	–
Wang and Chang (2016)	G	94.08	91.82	87.55	86.23	–	–
Cheng et al. (2016)	G	94.10	91.49	88.1	85.7	–	–
Kuncoro et al. (2016)	G	94.26	92.06	88.87	87.30	91.60	89.24
Ma and Hovy (2017)	G	94.88	92.98	89.05	87.74	92.58	90.54
BIAF: Dozat and Manning (2017)	G	95.74	94.08	89.30	88.23	93.46	91.44
BIAF: re-impl	G	95.84	**94.21**	90.43	89.14	**93.85**	**92.32**
STACKPTR: Org	T	95.77	94.12	90.48	89.19	93.59	92.06
STACKPTR: +gpar	T	95.78	94.12	90.49	89.19	93.65	92.12
STACKPTR: +sib	T	95.85	94.18	90.43	89.15	93.76	92.21
STACKPTR: Full	T	**95.87**	94.19	**90.59**	**89.29**	93.65	92.11

Table 1: UAS and LAS of four versions of our model on test sets for three languages, together with top-performing parsing systems. "T" and "G" indicate transition- and graph-based models, respectively. For BIAF, we provide the original results reported in Dozat and Manning (2017) and our re-implementation. For STACKPTR and our re-implementation of BiAF, we report the average over 5 runs.



(a) (b) (c)

Figure 3: Parsing performance of BIAF and STACKPTR parsers relative to length and graph factors.

POS	UAS	LAS	UCM	LCM
Gold	96.12±0.03	95.06±0.05	62.22±0.33	55.74±0.44
Pred	95.87±0.04	94.19±0.04	61.43±0.49	49.68±0.47
None	95.90±0.05	94.21±0.04	61.58±0.39	49.87±0.46

Table 2: Parsing performance on the test data of PTB with different versions of POS tags.

re-implementation of BIAF obtains better performance than the original one in Dozat and Manning (2017), demonstrating the effectiveness of the character-level information. Our model achieves state-of-the-art performance on both UAS and LAS on Chinese, and best UAS on English. On German, the performance is competitive with BIAF, and significantly better than other models.

4.4 Error Analysis

In this section, we characterize the errors made by BIAF and STACKPTR by presenting a number of experiments that relate parsing errors to a set of linguistic and structural properties. For simplicity, we follow McDonald and Nivre (2011) and report labeled parsing metrics (either accuracy, precision, or recall) for all experiments.

4.4.1 Length and Graph Factors

Following McDonald and Nivre (2011), we analyze parsing errors related to structural factors.

Sentence Length. Figure 3 (a) shows the accuracy of both parsing models relative to sentence lengths. Consistent with the analysis in McDonald and Nivre (2011), STACKPTR tends to perform better on shorter sentences, which make fewer parsing decisions, significantly reducing the chance of error propagation.

Dependency Length. Figure 3 (b) measures the precision and recall relative to dependency lengths. While the graph-based BIAF parser still performs better for longer dependency arcs and transition-based STACKPTR parser does better for shorter ones, the gap between the two systems is marginal, much smaller than that shown

	Bi-Att	NeuroMST	BIAF	STACKPTR	**Best Published**	
	UAS [LAS]	UAS [LAS]	UAS [LAS]	UAS [LAS]	UAS	LAS
ar	80.34 [68.58]	80.80 [69.40]	82.15±0.34 [71.32±0.36]	**83.04±0.29 [72.94±0.31]**	81.12	–
bg	93.96 [89.55]	94.28 [90.60]	94.62±0.14 [**91.56±0.24**]	**94.66±0.10** [91.40±0.08]	94.02	–
zh	–	93.40 [90.10]	**94.05±0.27 [90.89±0.22]**	93.88±0.24 [90.81±0.55]	93.04	–
cs	91.16 [85.14]	91.18 [85.92]	92.24±0.22 [87.85±0.21]	**92.83±0.13 [88.75±0.16]**	91.16	85.14
da	91.56 [85.53]	91.86 [87.07]	**92.80±0.26 [88.36±0.18]**	92.08±0.15 [87.29±0.21]	92.00	–
nl	87.15 [82.41]	87.85 [84.82]	90.07±0.18 [**87.24±0.17**]	**90.10±0.27** [87.05±0.26]	87.39	–
en	–	94.66 [92.52]	95.19±0.05 [93.14±0.05]	**93.25±0.05 [93.17±0.05]**	93.25	–
de	92.71 [89.80]	93.62 [91.90]	94.52±0.11 [93.06±0.11]	**94.77±0.05 [93.21±0.10]**	92.71	89.80
ja	93.44 [90.67]	**94.02 [92.60]**	93.95±0.06 [92.46±0.07]	93.38±0.08 [91.92±0.16]	93.80	–
pt	92.77 [88.44]	92.71 [88.92]	93.41±0.08 [89.96±0.24]	**93.57±0.12 [90.07±0.20]**	93.03	–
sl	86.01 [75.90]	86.73 [77.56]	87.55±0.17 [78.52±0.35]	**87.59±0.36 [78.85±0.53]**	87.06	–
es	88.74 [84.03]	89.20 [85.77]	90.43±0.13 [87.08±0.14]	**90.87±0.26 [87.80±0.31]**	88.75	84.03
sv	90.50 [84.05]	91.22 [86.92]	92.22±0.15 [88.44±0.17]	**92.49±0.21 [89.01±0.22]**	91.85	85.26
tr	78.43 [66.16]	77.71 [65.81]	**79.84±0.23 [68.63±0.29]**	79.56±0.22 [68.03±0.15]	78.43	66.16

Table 3: UAS and LAS on 14 treebanks from CoNLL shared tasks, together with several state-of-the-art parsers. Bi-Att is the bi-directional attention based parser (Cheng et al., 2016), and NeuroMST is the neural MST parser (Ma and Hovy, 2017). "Best Published" includes the most accurate parsers in term of UAS among Koo et al. (2010), Martins et al. (2011), Martins et al. (2013), Lei et al. (2014), Zhang et al. (2014), Zhang and McDonald (2014), Pitler and McDonald (2015), and Cheng et al. (2016).

in McDonald and Nivre (2011). One possible reason is that, unlike traditional transition-based parsers that scan the sentence from left to right, STACKPTR processes in a top-down manner, thus sometimes unnecessarily creating shorter dependency arcs first.

Root Distance. Figure 3 (c) plots the precision and recall of each system for arcs of varying distance to the root. Different from the observation in McDonald and Nivre (2011), STACKPTR does not show an obvious advantage on the precision for arcs further away from the root. Furthermore, the STACKPTR parser does not have the tendency to over-predict root modifiers reported in McDonald and Nivre (2011). This behavior can be explained using the same reasoning as above: the fact that arcs further away from the root are usually constructed early in the parsing algorithm of traditional transition-based parsers is not true for the STACKPTR parser.

4.4.2 Effect of POS Embedding

The only prerequisite information that our parsing model relies on is POS tags. With the goal of achieving an end-to-end parser, we explore the effect of POS tags on parsing performance. We run experiments on PTB using our STACKPTR parser with gold-standard and predicted POS tags, and without tags, respectively. STACKPTR in these experiments is the Full model with beam=10.

Table 2 gives results of the parsers with different versions of POS tags on the test data of PTB. The parser with gold-standard POS tags significantly outperforms the other two parsers, showing that dependency parsers can still benefit from accurate POS information. The parser with predicted (imperfect) POS tags, however, performs even slightly worse than the parser without using POS tags. It illustrates that an end-to-end parser that doesn't rely on POS information can obtain competitive (or even better) performance than parsers using imperfect predicted POS tags, even if the POS tagger is relative high accuracy (accuracy $> 97\%$ in this experiment on PTB).

4.5 Experiments on Other Treebanks

4.5.1 CoNLL Treebanks

Table 3 summarizes the parsing results of our model on the test sets of 14 treebanks from the CoNLL shared task, along with the state-of-the-art baselines. Along with BIAF, we also list the performance of the bi-directional attention based Parser (Bi-Att) (Cheng et al., 2016) and the neural MST parser (NeuroMST) (Ma and Hovy, 2017) for comparison. Our parser achieves state-of-the-art performance on both UAS and LAS on eight languages — Arabic, Czech, English, German, Portuguese, Slovene, Spanish, and Swedish. On Bulgarian and Dutch, our parser obtains the best UAS. On other languages, the performance of our parser is competitive with BIAF, and significantly better than others. The only exception is Japanese, on which NeuroMST obtains the best scores.

	Dev				Test			
	BIAF		STACKPTR		BIAF		STACKPTR	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
bg	93.92±0.13	89.05±0.11	**94.09±0.16**	**89.17±0.14**	94.30±0.16	**90.04±0.16**	**94.31±0.06**	89.96±0.07
ca	94.21±0.05	91.97±0.06	**94.47±0.02**	**92.51±0.05**	94.36±0.06	92.05±0.07	**94.47±0.02**	**92.39±0.02**
cs	94.14±0.03	90.89±0.04	**94.33±0.04**	**91.24±0.05**	94.06±0.04	90.60±0.05	**94.21±0.06**	**90.94±0.07**
de	91.89±0.11	88.39±0.17	**92.26±0.11**	**88.79±0.15**	90.26±0.19	86.11±0.25	**90.26±0.07**	**86.16±0.01**
en	**92.51±0.08**	**90.50±0.07**	92.47±0.03	90.46±0.02	91.91±0.17	89.82±0.16	**91.93±0.07**	**89.83±0.06**
es	93.46±0.05	91.13±0.07	**93.54±0.06**	**91.34±0.05**	93.72±0.07	91.33±0.08	**93.77±0.07**	**91.52±0.07**
fr	**95.05±0.04**	**92.76±0.07**	94.97±0.04	92.57±0.06	92.62±0.15	89.51±0.14	**92.90±0.20**	**89.88±0.23**
it	94.89±0.12	92.58±0.12	**94.93±0.09**	**92.90±0.10**	**94.75±0.12**	**92.72±0.12**	94.70±0.07	92.55±0.09
nl	93.39±0.08	90.90±0.07	**93.94±0.11**	**91.67±0.08**	93.44±0.09	91.04±0.06	**93.98±0.05**	**91.73±0.07**
no	95.44±0.05	93.73±0.05	**95.52±0.08**	**93.80±0.08**	95.28±0.05	93.58±0.05	**95.33±0.03**	**93.62±0.03**
ro	91.97±0.13	85.38±0.03	**92.06±0.08**	**85.58±0.12**	**91.94±0.07**	**85.61±0.13**	91.80±0.11	85.34±0.21
ru	93.81±0.05	91.85±0.06	**94.11±0.07**	**92.29±0.10**	94.40±0.03	92.68±0.04	**94.69±0.04**	**93.07±0.03**

Table 4: UAS and LAS on both the development and test datasets of 12 treebanks from UD Treebanks, together with BIAF for comparison.

4.5.2 UD Treebanks

For UD Treebanks, we select 12 languages — Bulgarian, Catalan, Czech, Dutch, English, French, German, Italian, Norwegian, Romanian, Russian and Spanish. For all the languages, we adopt the standard training/dev/test splits, and use the universal POS tags (Petrov et al., 2012) provided in each treebank. The statistics of these corpora are provided in Appendix B.

Table 4 summarizes the results of the STACKPTR parser, along with BIAF for comparison, on both the development and test datasets for each language. First, both BIAF and STACKPTR parsers achieve relatively high parsing accuracies on all the 12 languages — all with UAS are higher than 90%. On nine languages — Catalan, Czech, Dutch, English, French, German, Norwegian, Russian and Spanish — STACKPTR outperforms BIAF for both UAS and LAS. On Bulgarian, STACKPTR achieves slightly better UAS while LAS is slightly worse than BIAF. On Italian and Romanian, BIAF obtains marginally better parsing performance than STACKPTR.

5 Conclusion

In this paper, we proposed STACKPTR, a transition-based neural network architecture, for dependency parsing. Combining pointer networks with an internal stack to track the status of the top-down, depth-first search in the decoding procedure, the STACKPTR parser is able to capture information from the whole sentence and all the previously derived subtrees, removing the left-to-right restriction in classical transition-based parsers, while maintaining linear parsing steps, w.r.t the length of the sentences. Experimental results on 29 treebanks show the effectiveness of our parser across 20 languages, by achieving state-of-the-art performance on 21 corpora.

There are several potential directions for future work. First, we intend to consider how to conduct experiments to improve the analysis of parsing errors qualitatively and quantitatively. Another interesting direction is to further improve our model by exploring reinforcement learning approaches to learn an optimal order for the children of head words, instead of using a predefined fixed order.

Acknowledgements

The authors thank Chunting Zhou, Di Wang and Zhengzhong Liu for their helpful discussions. This research was supported in part by DARPA grant FA8750-18-2-0018 funded under the AIDA program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual nlp. In *Proceedings of CoNLL-2013*. Sofia, Bulgaria, pages 183–192.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of ACL-2016 (Volume 1: Long Papers)*. Berlin, Germany, pages 2442–2452.

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction.

In *Proceedings of ACL-2015 (Volume 1: Long Papers)*. Beijing, China, pages 344–354.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR-2015*.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proceedings of EMNLP-2015*. Lisbon, Portugal, pages 349–359.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack lstm parser. In *Proceedings of EMNLP-2016*. Austin, Texas, pages 2005–2010.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Simaan. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of EMNLP-2017*. Copenhagen, Denmark, pages 1957–1967.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of EMNLP-2012*. Jeju Island, Korea, pages 1455–1465.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceeding of CoNLL-2006*. New York, NY, pages 149–164.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP-2014*. Doha, Qatar, pages 740–750.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *Proceedings of EMNLP-2016*. Austin, Texas, pages 2204–2214.

Jason Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics* 4:357–370.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* .

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *Proceedings of ICLR-2017 (Volume 1: Long Papers)*. Toulon, France.

Greg Durrett and Dan Klein. 2013. Easy victories and uphill battles in coreference resolution. In *Proceedings of EMNLP-2013*. Seattle, Washington, USA, pages 1971–1982.

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 ACL-2015 (Volume 1: Long Papers)*. Beijing, China, pages 334–343.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-1996 (Volume 1)*. Association for Computational Linguistics, pages 340–345.

Nicolas R Fauceglia, Yiu-Chang Lin, Xuezhe Ma, and Eduard Hovy. 2015. Word sense disambiguation via propstore and ontonotes for event mention detection. In *Proceedings of the The 3rd Workshop on EVENTS: Definition, Detection, Coreference, and Representation*. Denver, Colorado, pages 11–15.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, et al. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL-2009: Shared Task*. pages 1–18.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics* 4:313–327.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of ACL-2010*. Uppsala, Sweden, pages 1–11.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP-2010*. Cambridge, MA, pages 1288–1298.

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 EMNLP-2016*. Austin, Texas, pages 1744–1753.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of ACL-2014 (Volume 1: Long Papers)*. Baltimore, Maryland, pages 1381–1391.

Wang Ling, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Two/too simple adaptations of word2vec for syntax problems. In *Proceedings of NAACL-2015*. Denver, Colorado, pages 1299–1304.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of EMNLP-2015*. Lisbon, Portugal, pages 1412–1421.

Xuezhe Ma, Yingkai Gao, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. 2017. Dropout with expectation-linear regularization. In *Proceedings of the 5th International Conference on Learning Representations (ICLR-2017)*. Toulon, France.

Xuezhe Ma and Eduard Hovy. 2015. Efficient inner-to-outer greedy algorithm for higher-order labeled dependency parsing. In *Proceedings of EMNLP-2015*. Lisbon, Portugal, pages 1322–1328.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of ACL-2016 (Volume 1: Long Papers)*. Berlin, Germany, pages 1064–1074.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective mst parsing. In *Proceedings of IJCNLP-2017 (Volume 1: Long Papers)*. Taipei, Taiwan, pages 59–69.

Xuezhe Ma, Zhengzhong Liu, and Eduard Hovy. 2016. Unsupervised ranking model for entity coreference resolution. In *Proceedings of NAACL-2016*. San Diego, California, USA.

Xuezhe Ma and Fei Xia. 2014. Unsupervised dependency parsing with transferring distribution via parallel guidance and entropy regularization. In *Proceedings of ACL-2014*. Baltimore, Maryland, pages 1337–1348.

Xuezhe Ma and Hai Zhao. 2012a. Fourth-order dependency parsing. In *Proceedings of COLING 2012: Posters*. Mumbai, India, pages 785–796.

Xuezhe Ma and Hai Zhao. 2012b. Probabilistic models for high-order projective dependency parsing. *Technical Report, arXiv:1502.04174* .

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics* 19(2):313–330.

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of ACL-2013 (Volume 2: Short Papers)*. Sofia, Bulgaria, pages 617–622.

Andre Martins, Noah Smith, Mario Figueiredo, and Pedro Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of EMNLP-2011*. Edinburgh, Scotland, UK., pages 238–249.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of ACL-2005*. Ann Arbor, Michigan, USA, pages 91–98.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics* 37(1):197–230.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal dependency annotation for multilingual parsing. In *Proceedings of ACL-2013*. Sofia, Bulgaria, pages 92–97.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proceeding of EACL-2006*.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT/EMNLP-2005*. Vancouver, Canada, pages 523–530.

Vincent Ng. 2010. Supervised noun phrase coreference research: The first fifteen years. In *Proceedings of ACL-2010*. Association for Computational Linguistics, Uppsala, Sweden, pages 1396–1411.

Truc-Vien T. Nguyen, Alessandro Moschitti, and Giuseppe Riccardi. 2009. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of EMNLP-2009*. Singapore, pages 1378–1387.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*. Prague, Czech Republic, pages 915–932.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of COLING-2004*. Geneva, Switzerland, pages 64–70.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of ICML-2013*. pages 1310–1318.

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics* 5:101–115.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of LREC-2012*. Istanbul, Turkey, pages 2089–2096.

Emily Pitler and Ryan McDonald. 2015. A linear-time transition system for crossing interval trees. In *Proceedings of NAACL-2015*. Denver, Colorado, pages 662–671.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The conll-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of CoNLL-2008*. pages 159–177.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings ACL-2015 (Volume 1: Long Papers)*. Beijing, China, pages 1556–1566.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. pages 2692–2700.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proceedings of ACL-2016 (Volume 1: Long Papers)*. Berlin, Germany, pages 2306–2315.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of ACL-2015 (Volume 1: Long Papers)*. Beijing, China, pages 323–333.

Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated chinese corpus. In *Proceedings of COLING-2002*. pages 1–8.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*. Nancy, France, volume 3, pages 195–206.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of ACL-2014 (Volume 2: Short Papers)*. Baltimore, Maryland, pages 656–661.

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2014. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of EMNLP-2014*. Doha, Qatar, pages 1013–1024.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA, pages 188–193.

Appendix: Stack-Pointer Network for Dependency Parsing

Appendix A: Hyper-Parameters

Table 5 summarizes the chosen hyper-parameters used for all the experiments in this paper. Some parameters are chosen directly or similarly from those reported in Dozat and Manning (2017). We use the same hyper-parameters across the models on different treebanks and languages, due to time constraints.

Layer	Hyper-parameter	Value
CNN	window size	3
	number of filters	50
LSTM	encoder layers	3
	encoder size	512
	decoder layers	1
	decoder size	512
MLP	arc MLP size	512
	label MLP size	128
Dropout	embeddings	0.33
	LSTM hidden states	0.33
	LSTM layers	0.33
Learning	optimizer	Adam
	initial learning rate	0.001
	(β_1, β_2)	(0.9, 0.9)
	decay rate	0.75
	gradient clipping	5.0

Table 5: Hyper-parameters for all experiments.

Appendix B: UD Treebanks

Table 6 shows the corpora statistics of the treebanks for 12 languages. For evaluation, we report results excluding punctuation, which is any tokens with POS tags "PUNCT" or "SYM".

	Corpora		*#Sent*	*#Token (w.o punct)*
Bulgarian	BTB	Training	8,907	124,336 (106,813)
		Dev	1,115	16,089 (13,822)
		Test	1,116	15,724 (13,456)
Catalan	AnCora	Training	13,123	417,587 (371,981)
		Dev	1,709	56,482 (50,452)
		Test	1,846	57,738 (51,324)
Czech	PDT, CAC	Training	102,993	1,806,230 (1,542,805)
	CLTT	Dev	11,311	191,679 (163,387)
	FicTree	Test	12,203	205,597 (174,771)
Dutch	Alpino	Training	18,310	267,289 (234,104)
	LassySmall	Dev	1,518	22,091 (19,042)
		Test	1,396	21,126 (18,310)
English	EWT	Training	12,543	204,585 (180,308)
		Dev	2,002	25,148 (21,998)
		Test	2,077	25,096 (21,898)
French	GSD	Training	14,554	356,638 (316,780)
		Dev	1,478	35,768 (31,896)
		Test	416	10,020 (8,795)
German	GSD	Training	13,841	263,536 (229,204)
		Dev	799	12,348 (10,727)
		Test	977	16,268 (13,929)
Italian	ISDT	Training	12,838	270,703 (239,836)
		Dev	564	11,908 (10,490)
		Test	482	10,417 (9,237)
Norwegian	Bokmaal	Training	29,870	48,9217 (43,2597)
	Nynorsk	Dev	4,300	67,619 (59,784)
		Test	3,450	54,739 (48,588)
Romanian	RRT	Training	8,043	185,113 (161,429)
		Dev	752	17,074 (14,851)
		Test	729	16,324 (14,241)
Russian	SynTagRus	Training	48,814	870,034 (711,184)
		Dev	6,584	118,426 (95,676)
		Test	6,491	117,276 (95,745)
Spanish	GSD	Training	28,492	827,053 (730,062)
	AnCora	Dev	4,300	89,487 (78,951)
		Test	2,174	64,617 (56,973)

Table 6: Corpora statistics of UD Treebanks for 12 languages. *#Sent* and *#Token* refer to the number of sentences and the number of words (w./w.o punctuations) in each data set, respectively.

Appendix C: Main Results

Table 7 illustrates the details of the experimental results. For each STACKPRT parsing model, we ran experiments with decoding beam size equals to 1, 5, and 10. For each experiment, we report the mean values with corresponding standard deviations over 5 runs.

		English							
		Dev				**Test**			
Model	beam	UAS	LAS	UCM	LCM	UAS	LAS	UCM	LCM
BiAF	–	95.73±0.04	**93.97±0.06**	60.58±0.77	47.47±0.63	95.84±0.06	**94.21±0.04**	59.49±0.23	49.07±0.34
Basic	1	95.71±0.02	93.88±0.03	62.33±0.33	47.75±0.32	95.71±0.06	94.07±0.06	60.91±0.35	49.54±0.48
	5	95.71±0.04	93.88±0.05	62.40±0.45	47.80±0.44	95.76±0.11	94.12±0.11	61.09±0.43	49.67±0.41
	10	95.72±0.03	93.89±0.04	**62.40±0.45**	**47.80±0.44**	95.77±0.11	94.12±0.11	61.09±0.43	49.67±0.41
+gpar	1	95.68±0.04	93.82±0.02	61.82±0.36	47.32±0.14	95.73±0.04	94.07±0.05	60.99±0.46	49.83±0.59
	5	95.67±0.01	93.83±0.02	61.93±0.32	47.44±0.20	95.76±0.06	94.11±0.06	61.23±0.47	50.07±0.59
	10	95.69±0.02	93.83±0.02	61.95±0.32	47.44±0.20	95.78±0.05	94.12±0.06	61.24±0.46	**50.07±0.59**
+sib	1	95.75±0.03	93.93±0.04	61.93±0.49	47.66±0.48	95.77±0.15	94.11±0.06	61.32±0.37	49.75±0.29
	5	95.74±0.02	93.93±0.05	62.16±0.22	47.68±0.54	95.84±0.09	94.17±0.09	61.52±0.57	49.91±0.76
	10	95.75±0.02	93.94±0.06	62.17±0.20	47.68±0.54	95.85±0.10	94.18±0.09	**61.52±0.57**	49.91±0.76
Full	1	95.63±0.08	93.78±0.08	61.56±0.63	47.12±0.36	95.79±0.06	94.11±0.06	61.02±0.31	49.45±0.23
	5	95.75±0.06	93.90±0.08	62.06±0.42	47.43±0.36	95.87±0.04	94.20±0.03	61.43±0.49	49.68±0.47
	10	**95.75±0.06**	93.90±0.08	62.08±0.39	47.43±0.36	**95.87±0.04**	94.19±0.04	61.43±0.49	49.68±0.47
		Chinese							
		Dev				**Test**			
Model	beam	UAS	LAS	UCM	LCM	UAS	LAS	UCM	LCM
BiAF	–	90.20±0.17	88.94±0.13	43.41±0.83	38.42±0.79	90.43±0.08	89.14±0.09	42.92±0.29	38.68±0.25
Basic	1	89.76±0.32	88.44±0.28	45.18±0.80	40.13±0.63	90.04±0.32	88.74±0.40	45.00±0.47	40.12±0.42
	5	89.97±0.13	88.67±0.14	45.33±0.58	40.25±0.65	90.46±0.15	89.17±0.18	45.41±0.48	40.53±0.48
	10	89.97±0.14	88.68±0.14	45.33±0.58	40.25±0.65	90.48±0.11	89.19±0.15	45.44±0.44	40.56±0.43
+gpar	1	90.05±0.14	88.71±0.16	45.63±0.52	40.45±0.61	90.28±0.10	88.96±0.10	45.26±0.59	40.38±0.43
	5	90.17±0.14	88.85±0.13	46.03±0.53	40.69±0.55	90.45±0.15	89.14±0.14	45.71±0.46	40.80±0.26
	10	90.18±0.16	88.87±0.14	**46.05±0.58**	40.69±0.55	90.46±0.16	89.16±0.15	45.71±0.46	40.80±0.26
+sib	1	89.91±0.07	88.59±0.10	45.50±0.50	40.40±0.48	90.25±0.10	88.94±0.12	45.42±0.52	40.54±0.69
	5	89.99±0.05	88.70±0.09	45.55±0.36	40.37±0.14	90.41±0.07	89.12±0.07	45.76±0.46	40.69±0.52
	10	90.00±0.04	88.72±0.09	45.58±0.32	40.37±0.14	90.43±0.09	89.15±0.10	45.75±0.44	40.68±0.50
Full	1	90.21±0.15	88.85±0.15	45.83±0.52	40.54±0.60	90.36±0.16	89.05±0.15	45.60±0.33	40.73±0.23
	5	90.23±0.13	88.89±0.14	46.00±0.54	**40.75±0.64**	90.58±0.12	89.27±0.11	**46.20±0.26**	**41.25±0.22**
	10	**90.29±0.13**	**88.95±0.13**	46.03±0.54	**40.75±0.64**	**90.59±0.12**	**89.29±0.11**	**46.20±0.26**	**41.25±0.22**
		German							
		Dev				**Test**			
Model	beam	UAS	LAS	UCM	LCM	UAS	LAS	UCM	LCM
BiAF	–	**93.60±0.13**	**91.96±0.13**	58.79±0.25	49.59±0.19	**93.85±0.07**	**92.32±0.06**	60.60±0.38	52.46±0.24
Basic	1	93.35±0.14	91.58±0.17	59.64±0.78	49.75±0.64	93.39±0.09	91.85±0.09	61.08±0.31	52.21±0.53
	5	93.49±0.14	91.72±0.16	59.99±0.69	49.82±0.54	93.61±0.09	92.07±0.08	61.38±0.30	52.51±0.43
	10	93.48±0.14	91.71±0.17	**60.02±0.69**	49.84±0.54	93.59±0.09	92.06±0.08	61.38±0.30	52.51±0.43
+gpar	1	93.39±0.07	91.66±0.13	59.59±0.54	49.81±0.42	93.44±0.07	91.91±0.11	61.73±0.47	52.84±0.48
	5	93.47±0.09	91.75±0.10	59.81±0.55	50.05±0.39	93.68±0.04	92.16±0.04	62.09±0.44	53.13±0.42
	10	93.48±0.08	91.76±0.09	59.89±0.59	50.09±0.40	93.68±0.05	92.16±0.03	62.10±0.42	**53.14±0.4**
+sib	1	93.43±0.07	91.73±0.08	59.68±0.25	49.93±0.30	93.55±0.07	92.00±0.08	61.90±0.50	52.79±0.22
	5	93.53±0.05	91.83±0.07	59.95±0.23	50.14±0.39	93.75±0.09	92.20±0.08	62.21±0.38	53.03±0.18
	10	93.55±0.06	91.84±0.07	59.96±0.24	**50.15±0.40**	93.76±0.09	92.21±0.08	**62.21±0.38**	53.03±0.18
Full	1	93.33±0.13	91.60±0.16	59.78±0.32	49.78±0.29	93.50±0.04	91.91±0.11	61.80±0.28	52.95±0.37
	5	93.42±0.11	91.69±0.12	59.90±0.27	49.94±0.35	93.64±0.03	92.10±0.06	61.89±0.21	53.06±0.36
	10	93.40±0.11	91.67±0.12	59.90±0.27	49.94±0.35	93.64±0.03	92.11±0.05	61.89±0.21	53.06±0.36

Table 7: Parsing performance of different variations of our model on both the development and test sets for three languages, together with the BIAF parser as the baseline. Best results are highlighted with bold print.