Seq2seq Dependency Parsing

Zuchao Li[1,2], **Jiaxun Cai**[1,2], **Shexia He**[1,2], **Hai Zhao**[1,2*]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University
[2]Key Laboratory of Shanghai Education Commission for Intelligent Interaction
and Cognitive Engineering, Shanghai Jiao Tong University, Shanghai, China
{charlee, caijiaxun, heshexia}@sjtu.edu.cn, zhaohai@cs.sjtu.edu.cn

Abstract

This paper presents a sequence to sequence (seq2seq) dependency parser by directly predicting the relative position of head for each given word, which therefore results in a truly end-to-end seq2seq dependency parser for the first time. Enjoying the advantage of seq2seq modeling, we enrich a series of embedding enhancement, including firstly introduced subword and node2vec augmentation. Meanwhile, we propose a beam search decoder with tree constraint and sub-root decomposition over the sequence to furthermore enhance our seq2seq parser. Our parser is evaluated on benchmark treebanks, being on par with the state-of-the-art parsers by achieving 94.11% UAS on PTB and 88.78% UAS on CTB, respectively.

1 Introduction

Dependency parsing for syntactic structure can be unstrictly put into two categories in terms of searching strategies over parsing trees, graph-based and transition-based. The previous work (Eisner, 1996; Mcdonald et al., 2005) searched the entire tree space but limited to local features with higher computational costs, while (Nivre, 2003) could adopt rich features but subjected to limited searching space. Besides, ensemble or hybrid methods on both the basic models have been well studied (Nivre and Mcdonald, 2008; Zhang and Clark, 2008). Most traditional dependency parsers rely heavily on feature engineering, especially for graph-based parser, which suffers from poor efficiency and generalization ability. Recent tendency for dependency parsing is adopting neural networks due to their significant success in a wide range of applications. Specially, leveraging sequence-to-sequence (seq2seq) model for dependency parsing started to appear (Wiseman and Rush, 2016; Zhang et al., 2017b).

The recently proposed seq2seq parsers focus on predicting a transition sequence to build a dependency tree, which makes them actually fall back into the transition-based model constrained by the adopted transition parsing algorithm. In this paper, we propose a seq2seq parser with a novel parsing structure encoding independent of transition parsing operation sequence. The proposed parser first directly generates the head position for each word in an input sentence using a seq2seq model, and then employs a BiLSTM-CRF model (Huang et al., 2015) to predict the relation label for each determined dependency arc. Since the greatest challenge of the applying seq2seq model in dependency parsing is to guarantee the tree structure of the outputs, a beam search with tree constraint method is proposed to enforce a well-formed dependency tree in the decoder side. Besides, dependency parsing has to well handle long-distance parsing with corresponding to too long sequence learning bias in seq2seq models, we accordingly introduce a sub-root based sequence decomposition to effectively alleviate this issue.

Our proposed parser is evaluated on benchmark treebanks on both English and Chinese, which demonstrates that our model achieves the state-of-the-art scores among seq2seq parsing, and yields competitive performance with traditional transition- and graph-based parsers. In summary, this paper provides the following contributions:

*Corresponding author. This paper was partially supported by National Key Research and Development Program of China (No. 2017YFB0304100), National Natural Science Foundation of China (No. 61672343 and No. 61733011), Key Project of National Society Science Foundation of China (No. 15-ZDA041), The Art and Science Interdisciplinary Funds of Shanghai Jiao Tong University (No. 14JCRZ04).

• We propose a seq2seq model which does not rely on any transition sequence by directly predicting the head position for each word in the sentence, unlike all previous work that first predicted a transition sequence and then built dependency tree using a known transition parsing algorithm.

• We especially design a beam search algorithm in the decoder side, which always guarantees the output to be a well-formed dependency tree.

• To deal with the long-distance dependency, we introduce sub-root decomposition and it also alleviates the problem caused by too long seq2seq learning in the meantime.

2 Related Work

Early researches (Ma and Zhao, 2012; Martins et al., 2013) achieved comparable results with high-order feature. With the impressive success of deep neural networks in a wide range of NLP tasks (Cai and Zhao, 2017; Qin et al., 2017; He et al., 2018; Cai et al., 2018; Zhang et al., 2018; Bai and Zhao, 2018; Huang et al., 2018), neural models have been successfully applied to various dependency parsers (Li et al., 2018; Wang et al., 2017; Zhang et al., 2016). Dyer et al. (2015) introduced the stack LSTM to promote the transition-based parsing. Kiperwasser and Goldberg (2016) incorporated the bidirectional Long Short-Term Memory (BiLSTM) into both graph- and transition-based parsers. Andor et al. (2016) proposed globally normalized networks and achieved the best results of transition-based parsing, while the state-of-the-art result was reported in Dozat and Manning (2016), which proposed a deep biaffine model for graph-based parser.

Though previous neural parsers have achieved such inspiring progresses, the majority of them heavily rely on the primitive parsing framework. Usually, those neural parsers build a network for feature extracting and use the neural features in place of handcrafted ones to predict some discrete actions in a traditional parsing algorithm. Until recently, few work (Wiseman and Rush, 2016; Zhang et al., 2017b) considered the seq2seq model, resulting in an end-to-end parser. However, those seq2seq parsers focus on predicting the transition sequences and thus actually need some kinds of complicated post-processing to build well-formed dependency trees, which make them fall back to a transition-based parser with a seq2seq transition predictor.

In this work, we attempt to build a true seq2seq parser without transition parsing algorithm for post-processing. The proposed parser is different from previous proposed seq2seq parser in the way that it directly predicts the head position instead of the transition. Though both our parser and Zhang et al. (2017a) make direct head prediction during parsing, our parser essentially differs from their parser in two aspects. Firstly, their parser runs standard graph-based maximum spanning tree (MST) algorithm to guarantee the tree structure, and thus called head selection in their parser serves for the purpose of speeding up the graph-based parsing, while ours introduces a beam search with tree constraint only in the decoder side to enforce a well-formed tree and thus gets rid of relying on specific parsing algorithm. Secondly, each time the head selection parser selects a head, it relies on referring to each pair of words in the sentence. While our parser performs a fully generative decoding in one pass by using an attention layer to keep around the context of each word.

Another line of studies related to this work focus on beam search, which have attracted much interest and have been adopted to improve the performance of greedy parsers. Beam search commonly adopts approximate strategy for computational tractability, which aims at including more sophisticated features within reasonable cost. Zhang and Clark (2008) proposed a beam search based parser applied to integrate graph-based and transition-based parsers. Most transition-based neural models (Weiss et al., 2015; Zhou et al., 2015; Andor et al., 2016) incorporated the structured perceptron with beam search decoding, resulting in substantial improvement of accuracy. Additionally, beam search has been also incorporated in the seq2seq framework (Wiseman and Rush, 2016).

Following the previous works, we also adopt a beam search in our decoder. However, since our parser aims at predicting a structure instead of sequence, we especially design a beam search algorithm with several searching constraints, which enforces the tree structure of the outputs.

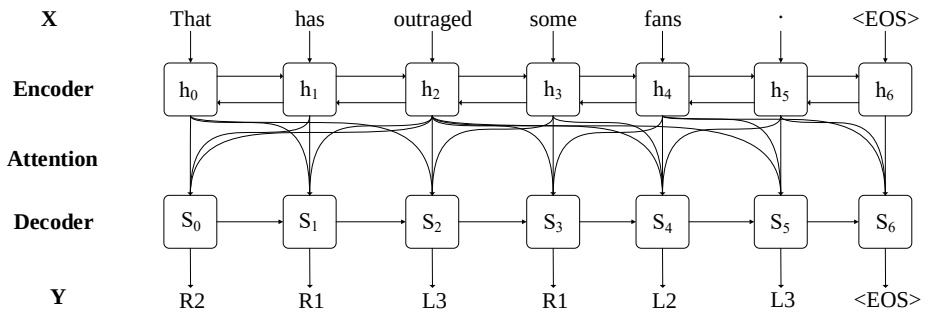

Figure 1: The framework of the proposed seq2seq model.

3 Seq2seq Parser

In this study, our parser is built upon a seq2seq model (Sutskever et al., 2014), which encodes the input sentence forward and backward, and predicts the head position for each word in the sentence. Our model contains three main components: (1) an encoder that processes the input sentence and maps it into some hidden states that lie in a low dimensional vector space $\mathbb{R}^h$, (2) a decoder that incorporates the hidden states and the previous prediction to generate head position of the current word, and (3) an attention layer that encodes the context for each focused word. Figure 1 illustrates our model.

Recurrent neural networks (RNNs) (Elman, 1990) is commonly used to build block for seq2seq model. Specifically, our model employs the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) variant of RNN, which uses numbers of gates to address the problem of vanishing or exploding gradient when trained with back-propagation through time. A standard LSTM unit consists of an input gate i_t, a forget gate f_t, an output gate o_t, a memory cell c_t and a hidden state h_t, where the subscript t denotes the time step.

3.1 Encoder

The encoder of our parser is a BiLSTM, which processes an input sequence in both directions by incorporating a stack of two distinct LSTMs.

Given an input sentence $X = \{w_1, \cdots, w_L\}$, the i-th element w_i is encoded by first feeding its embedding representation e_i in to two distinguish LSTMs: the forward LSTM and the backward LSTM, to obtain two hidden state vectors: h_i^f and h_i^b respectively, and then concatenating the two vectors as $h_i = [h_i^f; h_i^b]$.

3.1.1 Source Representation

Given a vocabulary W, each individual word $w_i \in W$ is mapped into a real-valued vector (word embedding) $w \in \mathbb{R}^m$ where m is the dimension. We employ the GloVe (Pennington et al., 2014) and Node2Vec (Grover and Leskovec, 2016) to generate the pre-trained word embedding, obtaining two distinct embedding for each word.

It is worth noting that Node2Vec is an embedding method that is capable of encoding the topological information of inside structure. To further incorporate the structural information of dependency tree, we perform a Node2Vec pre-training on the training set.

Besides, our model adopts subword which is obtained though BPE segmentation (Sennrich et al., 2015) broadly used in neural machine translation (NMT), and character augment embedding with AllenNLP toolkit according to (Gardner et al., 2017). A subword dictionary is built by running BPE on the Wikipedia, which segments each word into several subwords. Each subword in the dictionary is mapped into a real-valued vector (subword embedding). In our experiments, the subword embedding is randomly initialized and then trained jointly with other components of the network. To get the representation of the original word, we use an additional neural network component (Peters et al., 2018) to distill the character and subword embedding for representation. Our model also employs the part-of-speech (POS) tag embedding following previous works (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016). The

POS embedding is randomly initialized and jointly trained with other model parameters too.

The adopted word representation of our model is the concatenation of all above-mentioned embeddings: $e = [e^g; e^n; e^s; e^c; e^p]$, where e^g is the GloVe embedding, e^n is the Node2Vec embedding, e^s is the subword embedding, e^c is the character embedding and e^p is the POS tag embedding.

3.2 Decoder

In our model, an LSTM decoder predicts the head position for each word in the input sentence. Given a word w_i, the probability of each word in the vocabulary to be its head is given by:

$$p(\hat{y}_i|\hat{y}_1, \cdots, \hat{y}_{i-1}) = LSTM^{\mathcal{D}}(\hat{y}_{i-1}, h_i), \tag{1}$$

where $\hat{y}_{i-1}$ is the last output of the LSTM decoder, h_i is the hidden state of current word. The joint probability of the final output is then a product of conditional probability of each predicted head:

$$P(\hat{y}) = \prod_{i=0}^{M} p(\hat{y}_i|\{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_{i-1}\})$$

3.2.1 Target representation

In NMT context, the decoder output is sampled from the entire vocabulary using the probability computed by Eq.(1). However, in dependency parsing situation, the candidate set of the head words is subject to the words in the sentence. Thus, the predicted head would likely fall outside the input sentence when decoding with a full vocabulary upon corpus. To keep the generation of head within bounds, our model predicts the relative position instead of word form when assigning a head for each word. In the training process, we convert each head into a position representation, encoding relative distance between word and its head. When working, the parser picks a head for each word by predicting the relative position of the head. Our experiments show that the conversion is effective in bounding the output of our seq2seq parser.

Given a word w_i and its head word w_j, the relative position representation of w_j is obtained by:

$$R_{i,j} = \begin{cases} L_{j-i} & if \quad i < j, \\ R_{i-j} & if \quad i > j. \end{cases}$$

Figure 2 illustrates our relative position tag encoding for the output dependency structures.

3.3 Attention Mechanism

As usual, we employ an attention layer (Luong et al., 2015) to encode the context for each word. The context vector c_j at the j-step of the decoding process is calculated as the weighted sum of the hidden states of the input sequence:

$$c_j = \sum_{i=1}^{N} \alpha_j(i) h_i.$$

The attention weight $\alpha_j(i)$ between the i-th encoder hidden state h_i and the j-th decoder hidden state h_j is computed by a softmax function:

$$\alpha_j(i) = \frac{v^{\mathsf{T}} \tanh(W^{(a)}[h_i; h_j])}{\sum_{k=1}^{N} v^{\mathsf{T}} \tanh(W^{(a)}[h_k; h_j])},$$

where $[h_i; h_j]$ and $[h_k; h_j]$ denote the vector concatenation of the rows, $W^{(a)}$ and v are learnable parameters.

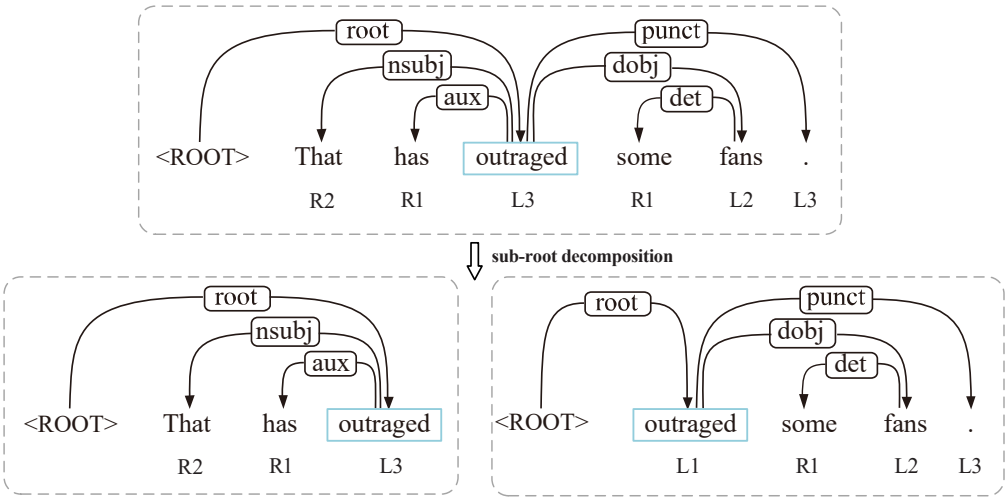

Figure 2: Illustration of target representation and sub-root segmentation for sentence ***That has outraged some fans .*** from PTB. The sequences below the word are the relative position tags which demonstrate the corresponding distance and direction (R: right, L: left) between the node and its syntactic head. The word **outraged** is defined as **sub-root**.

4 Sub-root Decomposition

It is well known that correctly recovering the long-distance dependency is the greatest challenge in dependency parsing. Even incorporating the BiLSTM encoder and the attention layer, the primitive seq2seq parser still performs poorly when encountering long sequences, as the long-distance dependencies heavily prevent the parser from building a correct tree.

To address the long-distance dependency problem and alleviate error propagation, we propose sub-root decomposition over the input parse tree. Such decomposition will cause the original dependency tree accordingly segmented into two or more subtrees, and the original input sentence segmented into shorter pieces. We define the notation **sub-root** as the direct syntactic children of the **ROOT** node. When working on 'nearly' projective dependency parsing, such decomposition would not break dependency arc in most situations.

Formally, given an input sentence sentence with length L: $X = \{w_1, w_2, \cdots, w_L\}$, for the i-th word w_i, we represent its head as $w_i.head$. Besides, we define a decomposition threshold β to determine whether an input sentence should be segmented into shorter ones. Only if the sentence length L is larger than the threshold β, we perform a decomposition on the sentence.

To perform the sub-root decomposition, we should first recognize the sub-root for a given sentence. We regard the sub-root recognition as a sequence tagging task, and employ a BiLSTM-CRF model which consists of 4-layer Bi-LSTM and a CRF layer to tag the sub-roots of a sentence. The output of this model is a binary indicator that distinguishes the sub-root nodes from others:

$$I(w_i) = \begin{cases} 0 & if \quad w_i.head \neq ROOT, \\ 1 & if \quad w_i.head = ROOT. \end{cases}$$

After recognizing the sub-root nodes, we decompose a sentence into several parts that come from collecting all nodes rooted by every sub-root, and perform the parsing on the sub-sentence. For example, given a sentence $X = \{w_1, w_2, \cdots, w_L\}$ with w_k as its sub-root, the sentence is decomposed into two sub-sentences: $X_1 = \{w_1, w_2, \cdots, w_k\}$ and $X_2 = \{w_k, w_{k+1}, \cdots, w_L\}$. Then we use the sentences X_1 and X_2 which are both rooted at w_k for training and parsing. It is worth noting that we use the gold sub-root to decompose a sentence in training, and predict the sub-roots for sentences in testing. Besides, the merge operation is necessary after decoding. The merge operation simply concatenates the sub-sentence together and set the head of sub-root node as ROOT.

Algorithm 1 Beam search with tree constraint

/* S decode scores; ts time step; L sentence length; $\mathcal{D}$ target vocabulary; Γ decode history;*/
procedure $step(S, ts, L, \mathcal{D}, \Gamma)$
Init empty valid target set $\mathcal{V}$ and visible history set Γ_v
if $ts > L$ **then**
  $S[\mathcal{D} \backslash \{\textbf{EOS}\}] \leftarrow -\infty$
else
  /* filter valid target by L and ts*/
  $\mathcal{V} \leftarrow valid\_target(\mathcal{D})$
  $\Gamma_v \leftarrow \Gamma[ts - \delta : ts - 1]$
  for $node \in \mathcal{D}$ **do**
    if $node \notin \mathcal{V}$ **then**
      $S[node] \leftarrow -\infty$
    else
      if $make\_circle(\Gamma_v, node)$ **then**
        $S[node] \leftarrow S[node] * \alpha_c$
      if $non\_projective(\Gamma_v, node)$ **then**
        $S[node] \leftarrow S[node] * \alpha_p$
$S \leftarrow top\mathbf{K}(S)$
$\Gamma \leftarrow \Gamma + \mathcal{D}[S]$
return S

4.1 Beam Search with Tree Constraint

Beam search is a popular approach to enlarge searching space in transition-based dependency parsing. The typical beam search algorithm keeps a fixed number (say K) of candidate states in the beam according to their scores (or probabilities). Each time a searching is performed, the algorithm explores all the possible next states for each states in the beam, and keeps the top-K states in beam again.

In inference stage, the beam B of time step ts is updated based on K states (hypotheses) of the last beam:

$$B_{ts} = topK[Y^k_{ts-1}, y^{k,n}_{ts}],$$

where $1 \leq k \leq K$ and $1 \leq n \leq N_k$. N_k is the number of branches of the k-th hypotheses in the last step. Y^k_{ts-1} represents the historical output and y_{ts} indicates the candidate target in the ts-th step. For each new exploring state, its score is computed by,

$$S(Y^k_{ts-1}, y^{k,n}_{ts}|x) = S(Y^k_{ts-1}|x) \ + \log p(y^{k,n}_{ts}|x, Y^k_{ts-1}).$$

Since our model directly predicts the head position for each word, its output might sometime violate the constraints of dependency tree structure. To enforce the model output to be a well-formed dependency tree, we introduce several constraints into the primitive beam search algorithm, resulting in the beam search with tree constraint.

The constraints that ensure a projective dependency tree are listed as follows:

• *Single headed*: A node should be assigned one and only one head. For our seq2seq parser, this also implies that the output sequence length should be exactly same as the input sequence length.

• *Acyclic*: In the decoding process, a newly generated head should not introduce a cycle in the dependency path.

• *Projective*: For the projective dependency parsing, the newly generated head should not cross the arc built by previous prediction (99.9% of PTB-SD, 100% of CTB on the training dataset are projective).

To enforce the above constraints, we introduce a weighted function $f_c(\cdot)$ that computes an additional weight for each candidate state to indicate whether the constraints are satisfied.

$$S(Y^k_{ts-1}, y^{k,n}_t|x) = S(Y^k_{ts-1}|x) \ + \log(p(y^{k,n}_t|x, Y^k_{ts-1}) \cdot f_c(y^{k,n}_{ts}|Y^k_{ts-1})),$$

$$f_c(y_{ts}^{k,n}|\cdot) = \begin{cases} 0, & ts > L, y_{ts} \neq \textbf{EOS}, \\ \alpha_c, & cycle(y_{ts-\delta}, \cdots, y_{ts}), \\ \alpha_p, & nproj(y_{ts-\delta}, \cdots, y_{ts}), \\ 1, & others. \end{cases}$$

where **EOS** is an additional symbol indicating the end of the predicted sequence, L is the length of the input sequence, $cycle(\cdot)$ is the cycle detected function which returns *True* if there exists a cycle on the input path, $nproj(\cdot)$ is an indicator of whether the resulting tree is projective. Note that for the sake of efficiency, we detect the violations of the *Acyclic* and *Projective* constraints by looking back a fixed number δ of historical predictions instead of enforcing them on the whole sequence. Thus, both $cycle(\cdot)$ and $nproj(\cdot)$ take a subsequence $y_{ts-\delta}, \cdots, y_{ts}$ instead of the whole sentence as input. By loosing the global constraints to local ones, the incorrect historical decisions that fall outside a window with size δ can never disturb a current prediction. In our experiments, δ is set to 8, both α_c and α_p are set to 0.8. Algorithm 1 shows the detail of the proposed beam search decoding with tree constraint.

5 Experiments

The proposed seq2seq parser is evaluated on the Penn Treebank (PTB) and the Chinese Treebank (CTB 5.1) with the unlabled attachment score (UAS) and the labeled attachment score (LAS) metrics (excluding punctuation). Follow the usual way processing the PTB and CTB:

For PTB, applying Stanford basic dependencies (SD) representation (De Marneffe et al., 2006), using sections 2-21 for training, section 22 for development and section 23 for testing as same as the standard splitting and tagging POS tag with the Stanford tagger (Toutanova et al., 2003).

For CTB, adopting Penn2Malt tool for conversion, splitting the dataset by sections 001-815, 1001-1136 for training, sections 886-931, 1148-1151 for development, and sections 816-885, 1137-1147 for testing as (Zhang and Clark, 2008) and using the golden segmentation and POS tags as (Chen and Manning, 2014).

5.1 Setup

In the experiments, our seq2seq parser implemented based on the OpenNMT-py project (Klein et al., 2017), in which employs a 4-layer Bi-LSTM as encoder and a 2-layer LSTM as decoder. The parameters are randomly initialized and optimized using the Adam algorithm with mini-batch size of 64. The initial learning rate is set to 0.001 (with $\beta_1 = 0.9, \beta_2 = 0.999$), and decays by 0.5 every epoch after running 20 epochs. In training phase, we adopt a dropout rate of 0.3. The size of the vocabulary in the target side is limited to 100 by setting the maximum relative position to 50, namely, the target side vocabulary is:

$$\mathcal{V}_t = \{L1, L2, \cdots, L50, R1, R2, \cdots, R50\}$$

In our experiments, the beam size is set to 5.

The representation of each input token is the concatenation of GloVe embedding, node2vec embedding and subword embedding. The GloVe embedding is pre-trained on Wikipedia 2014 and Gigaword 5 with 100 dimension (6B tokens). The node2vec embedding is pre-trained on the training set and the dimension is 128. It is trained with random walks in the syntactic graph (in order to make all the dependency tree form the graph, we add a dummy **ROOT** node). The walk length of node2vec is set to 30, per node visit(walk) times is 200 and the skip-gram window size is 10. To obtain subword segmentation for each word, we train the BPE model on the English and Chinese Wikipedia dataset respectively with number of merge operations set to 10K.

The sub-root tagging model consists of a 4-layer Bi-LSTM and a 1-layer CRF, taking the word and POS tag as inputs. Besides, after getting a dependency tree of the input sentence, we employ another BiLSTM-CRF model to predict the dependency relation of each arc. The predictor takes the word embedding, POS embeddings of head and dependent respectively as input, and performs a multiple labels tagging.

The code is available at `https://github.com/bcmi220/seq2seq_parser`.

5.2 Results

Table 1 shows comparison of our seq2seq parser with previously published parsers on PTB-SD and CTB. Unlike the transition-based parsers listed in ***Transition-NN*** block and the graph-based listed in the ***Graph-NN*** block, our model pursues a more simple way without any kind of transition- or graph-based algorithm. Our model outperforms most parsers that use either transition- or graph-based algorithm. Our model is competitive to the transition-based model of Andor et al. (2016) and slightly lower than the one of Zhu et al. (2015) which introduced a recursive convolutional neural network to encode the dependency tree and re-ranked the k-best trees. It is worth noting that Dozat and Manning (2016) used deep biaffine attention in the graph model (Kiperwasser and Goldberg, 2016), achieving the best results among traditional graph and transition parsing.

The parsers (Wiseman and Rush, 2016; Zhang et al., 2017b) are also seq2seq models, but they used transition sequence as target sequence encoding. The comparison shows our parser achieves state-of-the-art performance among seq2seq models. Kuncoro et al. (2016) also reported parsing result of 95.8% UAS on PTB. Since its result is converted from phrase-structure parsing and beyonds our focus, we excluded it from the table.

		PTB-YM	(%)	**PTB-LTH**	(%)	**PTB-SD**	(%)	**CTB**	(%)
System	**Method**	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
Non-NN:									
Ma and Zhao (2012)	4th order	-	93.4	-	-	-	-	-	87.4
Zhang and McDonald (2012)	cube pruning	-	93.1	-	-	-	-	-	86.9
Transition-NN:									
Dyer et al. (2015)	greedy	-	-	-	-	90.9	93.1	85.5	87.1
Kiperwasser and Goldberg (2016)	greedy	-	-	-	-	91.9	93.9	**86.1**	**87.6**
Andor et al. (2016)	beam	-	-	-	-	**92.79**	**94.61**	-	-
Zhu et al. (2015)	re-ranking	-	-	-	-	-	94.16	-	87.43
Graph-NN:									
Zhang and McDonald (2014)	3rd order	92.48	93.57	-	-	90.64	93.01	86.34	87.96
Zhang et al. (2016)	3rd order	92.23	93.31	90.07	93.14	91.29	93.42	86.17	87.65
Wang and Chang (2016)	1st order	92.45	93.51	-	-	91.82	94.08	86.23	87.55
Kiperwasser and Goldberg (2016)	1st order	-	-	-	-	90.90	93.0	84.9	86.5
Dozat and Manning (2016)	1st order	-	-	-	-	**94.08**	**95.74**	**88.23**	**89.30**
Zhang et al. (2017a)	1st order+re-ranking	-	-	-	-	91.90	94.10	86.15	87.84
Seq2seq:									
Wiseman and Rush (2016)	beam	-	-	-	-	87.26	91.57	-	-
Zhang et al. (2017b)	attention,single	-	-	-	-	91.60	93.71	85.40	87.41
This work	beam+sub-root	-	-	-	-	**92.08**	**94.11**	**86.23**	**88.78**

Table 1: Comparison of results on PTB and CTB test datasets.

5.3 Analysis

5.3.1 Sub-root Decomposition

This subsection gives an analysis on the effectiveness of the proposed sub-root decomposition algorithm. Figure 3 shows sentence length distribution on PTB and our sub-root tagging $F1$ score. For the sub-root tagging model, we get 96.73% and 96.01% $F1$ scores on development and test datasets, respectively.

To determine an optimal decomposition length threshold, we conduct experiments on PTB and the results are shown in Figure 4, which indicates that the threshold 40 performs best, though sub-root tagging model got the similar performance below the length 40. Meanwhile, all thresholds larger than 40 are superior to the case without decomposition, especially for the sentences longer than 40. We are surprised that all the other threshold settings are not better than the original case without any decomposition, which can be attributed to the dissatisfactory sub-root tagging $F1$ score.

Our experiments reveal that the sub-root decomposition method does work in improving the performance of our seq2seq parser, for its help on alleviating the long-distance dependencies and error propagation in the course of decoding.

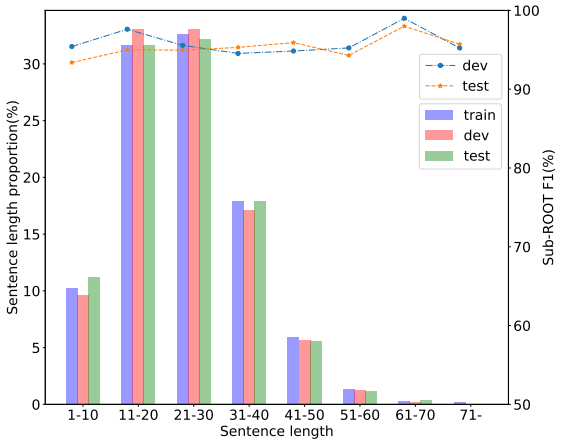

Figure 3: Sentence length distribution in PTB and the corresponding sub-root tagging accuracy.

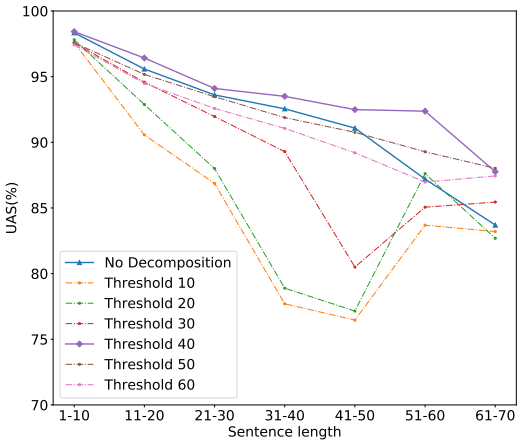

Figure 4: The UAS on PTB test dataset under different decomposition threshold.

5.3.2 Target Sequence

To make a direct comparison on the target encoding between our relative position sequence and previous transition sequence, we build a basic seq2seq model with the same source embedding, but without sub-root decomposition. The results are shown in Table 2, which indicates our relative position sequence is a better solution.

Since we limit the length of target sequence equal to the input sequence, the relative position sequence can also be predicted by sequence tagging model. We further compare the conventional sequence tagging model (Bi-LSTM+CRF with the same parameter settings as sub-root tagger) and the basic seq2seq model. The results demostrate that the seq2seq is necessarily better than the sequence tagging model for our concerned task.

Target encoding	**Model**	**Dev** (%) LAS	UAS	**Test** (%) LAS	UAS
Transition	seq2seq	83.56	87.34	82.77	87.49
Relative position	sequence tagging	83.81	87.58	81.37	87.60
Relative position	seq2seq	84.99	89.16	85.03	89.32

Table 2: Transition vs. relative position sequences in basic seq2seq model and sequence tagging vs. seq2seq model with relative position sequence (**without decomposition**).

5.3.3 Ablation Study

	Dev (%) LAS	UAS	**Test** (%) LAS	UAS
This work	91.86	93.84	92.08	94.11
-subword emb	91.35	93.76	91.77	93.95
-node2vec emb	91.60	93.71	91.85	94.01
-tree constraint	91.12	93.21	90.75	93.60

Table 3: Contribution of different components in our model.

In order to explore the contribution of the beam search with tree contraint and embeddings employed, we conduct a group of ablation experiments on PTB-SD dataset. The settings keep the same as before mentioned. Table 3 shows that the beam search with tree constraints substantially enhances the performance of our seq2seq parser. Meanwhile, both subword embedding and node2vec embedding contribute

to improving the parsing results.

6 Conclusion

In this paper, we propose a seq2seq parser with representation enhancement that directly predicts head positions without relying on transition sequence. To mitigate the long-distance dependency problem, we introduce the sub-root decomposition to shorten the sequence. To enforce a well-formed tree structure and alleviate error propagation, we employ beam search with constraints. Experiments on English and Chinese Penn Treebank verify the effectiveness of the proposed model. To the best of our knowledge, this is the first reported seq2seq parser with direct head prediction achieving promising performance.

References

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 the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August.

Hongxiao Bai and Hai Zhao. 2018. Deep enhanced representation for implicit discourse relation recognition. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*.

Deng Cai and Hai Zhao. 2017. *Pair-Aware Neural Sentence Modeling for Implicit Discourse Relation Classification*. IEA/AIE 2017, Part II, LNAI 10351.

Jiaxun Cai, Shexia He, Zuchao Li, and Hai Zhao. 2018. A full end-to-end semantic role labeler, syntactic-agnostic or syntactic-aware? In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454. Genoa Italy.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *Advances in Neural Information Processing Systems*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. pages 334–343.

Jason Eisner. 1996. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 79–86.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, (2):179–211.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

Shexia He, Zuchao Li, Hai Zhao, Hongxiao Bai, and Gongshen Liu. 2018. Syntax for semantic role labeling, to be, or not to be. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018)*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *Computer Science*.

Yafang Huang, Zuchao Li, Zhuosheng Zhang, and Hai Zhao. 2018. Moon IME: neural-based chinese pinyin aided input method with customizable association. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL 2018), System Demonstration*.

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.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. 2016. What do recurrent neural network grammars learn about syntax? *arXiv preprint arXiv:1611.05774*.

Haonan Li, Zhisong Zhang, Yuqi Ju, and Hai Zhao. 2018. Neural character-level dependency parsing for Chinese. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. pages 1412–1421, September.

Xuezhe Ma and Hai Zhao. 2012. Fourth-order dependency parsing. *Proceedings of COLING 2012: posters*, pages 785–796.

André FT Martins, Miguel B Almeida, and Noah A Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers.

Ryan Mcdonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Meeting on Association for Computational Linguistics*, pages 91–98.

Joakim Nivre and Ryan T. Mcdonald. 2008. Integrating graph-based and transition-based dependency parsers. In *ACL 2008, Proceedings of the Meeting of the Association for Computational Linguistics, June 15-20, 2008, Columbus, Ohio, Usa*, pages 950–958.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Lianhui Qin, Zhisong Zhang, Hai Zhao, Zhiting Hu, and Eric P. Xing. 2017. Adversarial connective-exploiting networks for implicit discourse relation classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL 2017)*, pages 1006–1017.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *Computer Science*.

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.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional lstm. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2306–2315.

Hao Wang, Hai Zhao, and Zhisong Zhang. 2017. A transition-based system for universal dependency parsing. In *CONLL 2017 Shared Task: Multilingual Parsing From Raw Text To Universal Dependencies (CONLL 2017)*, pages 191–197.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July.

Sam Wiseman and Alexander M. Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1296–1306, Austin, Texas, November.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 320–331. Association for Computational Linguistics.

Hao Zhang and Ryan McDonald. 2014. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 656–661.

Zhisong Zhang, Hai Zhao, and Lianhui Qin. 2016. Probabilistic graph-based dependency parsing with convolutional neural network. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, pages 1382–1392.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017a. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 665–676, Valencia, Spain, April.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2017b. Stack-based multi-layer attention for transition-based dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1677–1682, Copenhagen, Denmark, September.

Zhuosheng Zhang, Jiangtong Li, Pengfei Zhu, and Hai Zhao. 2018. Modeling multi-turn conversation with deep utterance aggregation. In *Proceedings of the 27th International Conference on Computational Linguistics (COLING 2018)*.

Hao Zhou, Yue Zhang, Shujian Huang, and Jiajun Chen. 2015. A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1213–1222, Beijing, China, July.

Chenxi Zhu, Xipeng Qiu, Xinchi Chen, and Xuanjing Huang. 2015. A re-ranking model for dependency parser with recursive convolutional neural network.