Dependency Parsing as MRC-based Span-Span Prediction

Leilei Gan[♦], Yuxian Meng[♣], Kun Kuang[♦], Xiaofei Sun[♦♣]
Chun Fan[♠], Fei Wu[♦♥] and Jiwei Li[♦♣]
[♦]College of Computer Science and Technology, Zhejiang University, [♣]Shannon.AI
[♥]Shanghai Institute for Advanced Study of Zhejiang University, [♥]Shanghai AI Laboratory
[♠]Peng Cheng Laboratory, [♠]National Biomedical Imaging Center, Peking University
[♠] Computer Center, Peking University
{leileigan, kunkuang, wufei, jiwei_li}@zju.edu.cn
{yuxian_meng, xiaofei_sun}@shannonai.com, fanchun@pku.edu.cn

Abstract

Higher-order methods for dependency parsing can partially but not fully address the issue that edges in dependency trees should be constructed at the text span/subtree level rather than word level. In this paper, we propose a new method for dependency parsing to address this issue. The proposed method constructs dependency trees by directly modeling span-span (in other words, subtree-subtree) relations. It consists of two modules: the *text span proposal module* which proposes candidate text spans, each of which represents a subtree in the dependency tree denoted by (root, start, end); and the *span linking module*, which constructs links between proposed spans. We use the machine reading comprehension (MRC) framework as the backbone to formalize the span linking module, where one span is used as query to extract the text span/subtree it should be linked to. The proposed method has the following merits: (1) it addresses the fundamental problem that edges in a dependency tree should be constructed between subtrees; (2) the MRC framework allows the method to retrieve missing spans in the span proposal stage, which leads to higher recall for eligible spans. Extensive experiments on the PTB, CTB and Universal Dependencies (UD) benchmarks demonstrate the effectiveness of the proposed method. [1] [2]

1 Introduction

Dependency parsing is a basic and fundamental task in natural language processing (NLP) (Eisner, 2000; Nivre, 2003; McDonald et al., 2005b). Among existing efforts for dependency parsers, graph-based models (McDonald et al., 2005a; Pei et al., 2015) are a widely used category of models, which cast the task as finding the optimal maximum spanning tree in the directed graph. Graph-based models provide a more global view than shift-reduce models (Zhang and Nivre, 2011; Chen and Manning, 2014), leading to better performances.

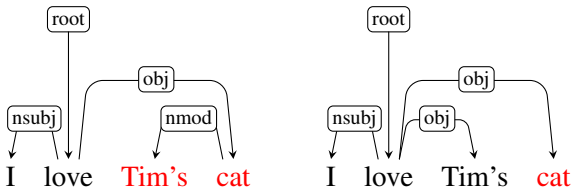

Figure 1: Two possible dependency trees for sentence "I love Tim's cat". For the tree on the right hand side, if we look at the token-token level, 'love" being linked to "cat" is correct. But at the subtree-subtree level, the linking is incorrect since the span/subtree behind "cat" is incorrect.

Graph-based methods are faced with a challenge: they construct dependency edges by using word pairs as basic units for modeling, which is insufficient because dependency parsing performs at the span/subtree level. For example, Figure 1 shows two possible dependency trees for the sentence "I love Tim's cat". In both cases, at the token level, "love" is linked to "cat". If we only consider token-token relations, the second case of 'love" being linked to "cat" is correct. But if we view the tree at the subtree-subtree level, the linking is incorrect since the span/subtree behind "cat" is incorrect. Although higher-order methods are able to alleviate this issue by aggregating information across adjacent edges, they can not fully address the issue. In nature, the token-token strategy can be viewed as a coarse simplification of the span-span (subtree-subtree) strategy, where the root token in the token-token strategy can be viewed as the **average** of all spans covering it. We would like an approach that directly models span-span relations using **exact** subtrees behind tokens, rather than the average of all spans covering it.

To address this challenge, in this work, we propose a model for dependency parsing that directly

[1]Chun Fan is the corresponding author.
[2]The code is available at https://github.com/ShannonAI/mrc-for-dependency-parsing

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 2427 - 2437

operates at the span-span relation level. The proposed model consists of two modules: (1) the *text span proposal module* which proposes eligible candidate text spans, each of which represents a subtree in the dependency tree denoted by (root, start, end); (2) and the *span linking module*, which constructs links between proposed spans to form the final dependency tree. We use the machine reading comprehension (MRC) framework as the backbone to formalize the span linking module in an MRC setup, where one span is used as a query to extract the text span/subtree it should be linked to. In this way, the proposed model is able to directly model span-span relations and build the complete dependency tree in a bottom-up recursive manner.

The proposed model provides benefits in the following three aspects: (1) firstly, it naturally addresses the shortcoming of token-token modeling in vanilla graph-based approaches and directly performs at the span level; (2) with the MRC framework, the left-out spans in the span proposal stage can still be retrieved at the span linking stage, and thus the negative effect of unextracted spans can be alleviated; and (3) the MRC formalization allows us to take advantage of existing state-of-the-art MRC models, with which the model expressivity can be enhanced, leading to better performances.

We are able to achieve new SOTA performances on PTB, CTB and UD benchmarks, which demonstrate the effectiveness of the proposed method.

2 Related Work

Transition-based dependency parsing incrementally constructs a dependency tree from input words through a sequence of shift-reduce actions (Zhang and Nivre, 2011; Chen and Manning, 2014; Zhou et al., 2015; Dyer et al., 2015; Yuan et al., 2019; Han et al., 2019; Mohammadshahi and Henderson, 2020a). Graph-based dependency parsing searches through the space of all possible dependency trees for a tree that maximizes a specific score (Pei et al., 2015; Wang et al., 2018; Zhang et al., 2019).

Graph-based dependency parsing is first introduced by McDonald et al. (2005a,b). They formalized the task of dependency parsing as finding the maximum spanning tree (MST) in directed graphs and used the large margin objective (Crammer et al., 2006) to efficiently train the model. Zhang et al. (2016) introduced a probabilistic convolutional neural network (CNN) for graph-based dependency parsing to model third-order dependency information Wang and Chang (2016); Kiperwasser and Goldberg (2016) proposed to employ LSTMs as an encoder to extract features, which are then used to score dependencies between words. Zhang and Zhao (2015); Zhang et al. (2019); Wang and Tu (2020) integrated higher-order features across adjacent dependency edges to build the dependency tree. Ji et al. (2019) captured high-order dependency information by using graph neural networks. The biaffine approach (Dozat and Manning, 2016) is a particular kind of graph-based method improving upon vanilla scoring functions in graph-based dependency parsing. Ma et al. (2018) combined biaffine classifiers and pointer networks to build dependency trees in a top-down manner. Jia et al. (2020); Zhang et al. (2020) extended the biaffine approach to the conditional random field (CRF) framework. Mrini et al. (2020) incorporated label information into the self-attention structure (Vaswani et al., 2017b) for biaffine dependency parsing.

3 Method

3.1 Notations

Given a sequence of input tokens $\mathbf{s} = (w_0, w_1, ..., w_n)$, where n denotes the length of the sentence and w_0 is a dummy token representing the root of the sentence, we formalize the task of dependency parsing as finding the tree with the highest score among all possible trees rooted at w_0 .

$$\hat{T} = \arg\max_{T_{w_0}} \text{score}(T_{w_0}) \quad (1)$$

Each token w_i in the input sentence corresponds to a subtree T_{w_i} rooted at w_i within in the full tree T, and the subtree can be characterized by a text span, with the index of its leftmost token being $T_{w_i}.s$ in the original sequence, and the index of its rightmost token being $T_{w_i}.e$ in the original sequence. As shown in the first example of Figure 1, the span covered by the subtree T_{love} is the full sentence "I love Tim's cat", and the span covered by the subtree T_{cat} is "Tim's cat". Each directional arc $w_i \rightarrow w_j$ in T represents a parent-child relation between T_{w_i} and T_{w_j}, where T_{w_j} is a subtree of T_{w_i}. This implies that the text span covered by T_{w_j} is fully contained by the text span covered by T_{w_i}.

It is worth noting that the currently proposed paradigm can only handle the projective situation. We will get back to how to adjust the current paradigm to non-projective situation in Section 3.5.

Sentence: *root I love Tim's cat*	
Spans	**Links**
T_{root}: root I love Tim's cat	$T_{\text{root}} \rightarrow T_{\text{love}}$
T_{I}: I	$T_{\text{love}} \rightarrow T_{\text{I}}$
T_{love}: I love Tim's cat	$T_{\text{love}} \rightarrow T_{\text{cat}}$
$T_{\text{Tim's}}$: Tim's	$T_{\text{cat}} \rightarrow T_{\text{Tim's}}$
T_{cat}: Tim's cat	

Table 1: Spans and links for the left tree in Figure 1.

3.2 Scoring Function

With notations defined in the previous section, we now illustrate how to compute the score(T_{w_0}) in Eq.(1). Since we want to model the span-span relations inside a dependency tree, where the tree is composed by spans and the links between them, we formalize the scoring function as:

$$\begin{aligned}\text{score}(T_{w_0}) = &\sum_{i=1}^{n} \text{score}_{\text{span}}(T_{w_i}) \\ &+ \lambda \sum_{(w_i \rightarrow w_j) \in T_{w_0}} \text{score}_{\text{link}}(T_{w_i}, T_{w_j})\end{aligned} \tag{2}$$

where $\text{score}_{\text{span}}(T_{w_i})$ represents how likely the subtree rooted at w_i covers the text span from $T.s$ to $T.e$. $\text{score}_{\text{link}}(T_{w_i}, T_{w_j})$ represents how likely tree T_{w_j} is a subtree of T_{w_i}, i.e. there is an arc from w_i to w_j, and λ is a hyper-parameter to balance $\text{score}_{\text{span}}$ and $\text{score}_{\text{link}}$. We will illustrate the details how to compute $\text{score}_{\text{span}}(T)$ and $\text{score}_{\text{link}}(T_1, T_2)$ in the following sections. Table 1 shows all the spans and links for the left tree in Figure 1.

3.3 Span Proposal Module

In this section, we introduce the span proposal module. This module gives each tree T_{w_i} a score $\text{score}_{\text{span}}(T_{w_i})$ in Eq.(2), which represents how likely the subtree rooted at w_i covers the text span from $T_{w_i}.s$ to $T_{w_i}.e$. The score can be decomposed into two components – the score for the left half span from w_i to $T_{w_i}.s$, and the score for the right half span from w_i to $T_{w_i}.e$, given by:

$$\begin{aligned}\text{score}_{\text{span}}(T_{w_i}) = &\ \text{score}_{\text{start}}(T_{w_i}.s|w_i) \\ &+ \text{score}_{\text{end}}(T_{w_i}.e|w_i)\end{aligned} \tag{3}$$

We propose to formalize $\text{score}_{\text{start}}(T_{w_i}.s|w_i)$ as the score for the text span starting at $T_{w_i}.s$, ending at w_i, by transforming the task to a text span extraction problem. Concretely, we use the biaffine function (Dozat and Manning, 2016) to score the text span by computing $\text{score}_{\text{start}}(j|i)$ – the score of the tree rooted at at w_i and staring at w_j:

$$\text{score}_{\text{start}}(j|i) = \mathbf{x}_i^\top U_{\text{start}} \mathbf{x}_j + \mathbf{w}_{\text{start}}^\top \mathbf{x}_j \tag{4}$$

where $U \in \mathbb{R}^{d \times d}$ and $\mathbf{w} \in \mathbb{R}^d$ are trainable parameters, $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{x}_j \in \mathbb{R}^d$ are token representations of w_i and w_j respectively. To obtain $\mathbf{x}_i$ and $\mathbf{x}_j$, we pass the sentence s to pretrained models such as BERT (Devlin et al., 2018). $\mathbf{x}_i$ and $\mathbf{x}_j$ are the last-layer representations output from BERT for w_i and w_j. We use the following loss to optimize the left-half span proposal module:

$$\mathcal{L}_{\text{span}}^{\text{start}} = -\sum_{i=1}^{n} \log \frac{\exp(\text{score}_{\text{start}}(T_{w_i}.s|i))}{\sum_{j=1}^{n} \exp(\text{score}_{\text{start}}(j|i))} \tag{5}$$

This objective enforces the model to find the correct span start $T_{w_i}.s$ for each word w_i. We ignore loss for w_0, the dummy root token.

$\text{score}_{\text{end}}(T_{w_i}.e|w_i)$ can be computed in the similar way, where the model extracts the text span rooted at index w_i and ending at $T_{w_i}.e$:

$$\text{score}_{\text{end}}(j|i) = \mathbf{x}_i^\top U_{\text{end}} \mathbf{x}_j + \mathbf{w}_{\text{end}}^\top \mathbf{x}_j \tag{6}$$

The loss to optimize the right-half span proposal module:

$$\mathcal{L}_{\text{span}}^{\text{end}} = -\sum_{i=1}^{n} \log \frac{\exp(\text{score}_{\text{end}}(T_{w_i}.e|i))}{\sum_{j=1}^{n} \exp(\text{score}_{\text{end}}(j|i))} \tag{7}$$

Using the left-half span score in Eq.(4) and the right-half span score in Eq.(6) to compute the full span score in Eq.(3), we are able to compute the score for any subtree, with text span starting at $T_{w_i}.s$, ending at $T_{w_i}.e$ and rooted at w_i.

3.4 Span Linking Module

Given two subtrees T_{w_i} and T_{w_j}, the span linking module gives a score – $\text{score}_{\text{link}}(T_{w_i}, T_{w_j})$ to represent the probability of T_{w_j} being a subtree of T_{w_i}. This means that T_{w_i} is the parent of T_{w_j}, and that the span associated with T_{w_j}, i.e., $(T_{w_j}.s, T_{w_j}.e)$ is fully contained in the span associated with T_{w_i}, i.e., $(T_{w_i}.s, T_{w_i}.e)$.

We propose to use the machine reading comprehension framework as the backbone to compute this score. It operates on the triplet {context (X), query (q) and answer (a)}. The context X is the original sentence $\mathbf{s}$. The query q is the child span $(T_{w_j}.s, T_{w_j}.e)$. And we wish to extract the answer, which is the parent span $(T_{w_i}.s, T_{w_i}.e)$ from the context input sentence s. The basic idea here is

that using the child span to query the full sentence gives direct cues for identifying the corresponding parent span, and this is more effective than simply feeding two extracted spans and then determining whether they have the parent-child relation.

Constructing Query Regarding the query, we should consider both the span and its root. The query is thus formalized as follows:

$$\begin{aligned}&\texttt{<sos>}, T_{w_j}.s, T_{w_j}.s+1, ..., T_{w_j}-1, \texttt{<sor>},\\ &T_{w_j}, \texttt{<eor>}, T_{w_j}+1, ...,\\ &T_{w_j}.e-1, T_{w_j}.e, \texttt{<eos>}\end{aligned} \quad (8)$$

where <sos>, <sor>, <eor>, and <eos> are special tokens, which respectively denote the start of span, the start of root, the end of root, and the end of span. One issue with the way above to construct query is that the position information of T_{w_j} is not included in the query. In practice, we turn to a more convenient strategy where the query is the original sentence, with special tokens <sos>, <sor>, <eor>, and <eos> used to denote the position of the child. In this way, position information for child T_{w_j} can be naturally considered.

Answer Extraction The answer is the parent, with the span $T_{w_i}.s, T_{w_i}.e$ rooted at T_{w_i}. We can directly take the framework from the MRC model by identifying the start and end of the answer span, respectively denoted by $\text{score}^s_{\text{parent}}(T_{w_i}.s|T_{w_j})$ and $\text{score}^e_{\text{parent}}(T_{w_i}.e|T_{w_j})$. We also wish to identify the root T_{w_i} from the answer, which is characterized by the score of w_i being the root of the span, denoted by $\text{score}^r_{\text{parent}}(w_i|T_{w_j})$. Furthermore, since we also want to identify the relation category between the parent and the child, the score signifying the relation label l is needed to be added, which is denoted by $\text{score}^l_{\text{parent}}(l|T_{w_j}, w_i)$.

For quadruple $(T_{w_i}.s, T_{w_i}.e, T_{w_j}, l)$, which denotes the span $T_{w_i}.s, T_{w_i}.e$ rooted at w_i, the final score for it being the answer to T_{w_j}, and the relation between the subtrees is l, is given by:

$$\begin{aligned}&\text{score}_{\text{parent}}(T_{w_i}|T_{w_j}) =\\ &\text{score}^r_{\text{parent}}(w_i|T_{w_j}) + \text{score}^s_{\text{parent}}(T_{w_i}.s|T_{w_j}) +\\ &\text{score}^e_{\text{parent}}(T_{w_i}.e|T_{w_j}) + \text{score}^l_{\text{parent}}(l|T_{w_j}, w_i)\end{aligned} \quad (9)$$

In the MRC setup, the input is the concatenation of the query and the context, denoted by {<cls>, query, <sep>, context}, where <cls> and <sep> are special tokens. The input is fed to BERT, and we obtain representations for each input token. Let $\mathbf{h}_t$ denote the representation for the token with index t output from BERT. The probability of t^{th} token being the root of the answer, which is denoted by $\text{score}^r_{\text{parent}}(w_t|T_{w_j})$ is the softmax function over all constituent tokens in the context:

$$\text{score}^r_{\text{parent}}(w_t|T_{w_j}) = \frac{\exp\left(\mathbf{h}^\top_{\text{root}} \times \mathbf{h}_t\right)}{\sum_{t' \in \text{context}} \exp(\mathbf{h}^\top_{\text{root}} \times \mathbf{h}_{t'})} \quad (10)$$

where $\mathbf{h}^\top_{\text{root}}$ is trainable parameter. $\text{score}^s_{\text{parent}}$ and $\text{score}^e_{\text{parent}}$ can be computed in the similar way:

$$\begin{aligned}\text{score}^s_{\text{parent}}(w_t|T_{w_j}) &= \frac{\exp\left(\mathbf{h}^\top_{\text{start}} \times \mathbf{h}_t\right)}{\sum_{t' \in \text{context}} \exp(\mathbf{h}^\top_{\text{start}} \times \mathbf{h}_{t'})}\\ \text{score}^e_{\text{parent}}(w_t|T_{w_j}) &= \frac{\exp\left(\mathbf{h}^\top_{\text{end}} \times \mathbf{h}_t\right)}{\sum_{t' \in \text{context}} \exp(\mathbf{h}^\top_{\text{end}} \times \mathbf{h}_{t'})}\end{aligned} \quad (11)$$

For $\text{score}^l_{\text{parent}}(l|T_{w_j}, w_i)$, which denotes the relation label between T_{w_i} and T_{w_j}, we can compute it in a simple way. Since $\mathbf{h}_{w_i}$ already encodes information for $\mathbf{h}_{w_j}$ through self-attentions, the representation $\mathbf{h}_{w_i}$ for w_i is directly fed to the softmax function over all labels in the label set $\mathcal{L}$:

$$\text{score}^l_{\text{parent}}(l|T_{w_j}, w_i) = \frac{\exp\left(\mathbf{h}^\top_l \times \mathbf{h}_{w_i}\right)}{\sum_{l' \in \mathcal{L}} \exp(\mathbf{h}^\top_{l'} \times \mathbf{h}_{w_i})} \quad (12)$$

Mutual Dependency A closer look at Eq.(9) reveals that it only models the uni-directional dependency relation that T_{w_i} is the parent of T_{w_j}. This is suboptimal since if T_{w_i} is a parent answer of T_{w_j}, T_{w_j} should be a child answer of T_{w_i}. We thus propose to use T_{w_i} as the query q and T_{w_j} as the answer a.

$$\begin{aligned}&\text{score}_{\text{child}}(T_{w_j}|T_{w_i}) =\\ &\text{score}^r_{\text{child}}(w_j|T_{w_i}) + \text{score}^s_{\text{child}}(T_{w_j}.s|T_{w_i}) +\\ &\text{score}^e_{\text{child}}(T_{w_j}.e|T_{w_i}) + \text{score}^l_{\text{child}}(l|T_{w_i}, T_{w_j})\end{aligned} \quad (13)$$

The final score $\text{score}_{\text{link}}$ is thus given by:

$$\begin{aligned}\text{score}_{\text{link}}(T_{w_i}, T_{w_j}) = &\text{score}_{\text{child}}(T_{w_j}|T_{w_i})\\ &+ \text{score}_{\text{parent}}(T_{w_i}|w_j)\end{aligned} \quad (14)$$

Since one tree may have multiple children but can only have one parent, we use the multi-label cross entropy loss $\mathcal{L}_{\text{parent}}$ for $\text{score}_{\text{parent}}(T_{w_i}|T_{w_j})$ and use the binary cross entropy loss $\mathcal{L}_{\text{child}}$ for $\text{score}_{\text{child}}(T_{w_j}|T_{w_i})$. We jointly optimize these two losses $\mathcal{L}_{\text{link}} = \mathcal{L}_{\text{parent}} + \mathcal{L}_{\text{child}}$ for span linking.

3.5 Inference

Given an input sentence $\mathbf{s} = (w_0, w_1, w_2, ..., w_n)$, the number of all possible subtree spans $(w_i, T_{w_i}.s, T_{w_i}.e)$ is $\mathcal{O}(n^3)$, and therefore running MRC procedure for every candidate span is computationally prohibitive. A naive solution is to use the span proposal module to extract top-k scored spans rooted at each token. This gives rise to a set of span candidates $\mathcal{T}$ with size $1 + n \times k$ (the root token w_0 produces only one span), where each candidate span is associated with its subtree span score $\text{score}_{\text{span}}(\cdot)$. Then we construct the optimal dependency tree based only on these extracted spans by linking them. This strategy obtains a local optimum for Eq.(2), because we want to compute the optimal solution for the first part $\sum_{i=1}^{n} \text{score}_{\text{span}}(T_{w_i})$ depending on the second part of Eq.(2), i.e., $\sum_{(w_i \to w_j) \in T_{w_0}} \text{score}_{\text{link}}(T_{w_i}, T_{w_j})$. But in this naive strategy, the second part is computed after the first part.

It is worth noting that the naive solution of using only the top-k scored spans has another severe issue: spans left out at the span proposal stage can never be a part of the final prediction, since the span linking module only operates on the proposed spans. This would not be a big issue if top-k is large enough to recall almost every span in ground-truth. However, span proposal is intrinsically harder than span linking because the span proposal module lacks the triplet span information that is used by the span linking module. Therefore, we propose to use the span linking module to retrieve more correct spans. Concretely, for every span T_{w_j} proposed by the span proposal module, we use $\arg\max \text{score}_{\text{parent}}(T_{w_i}|T_{w_j})$ to retrieve its parent with the highest score as additional span candidates. Recall that span proposal proposed $1 + n \times k$ spans. Added by spans proposed by the span linking module, the maximum number of candidate spans is $1 + 2 \times n \times k$. The MRC formalization behind the span linking module improves the recall rate as missed spans at the span proposal stage can still be retrieved at this stage.

Projective Decoding Given retrieved spans harvested in the proposal stage, we use a CKY-style bottom-up dynamic programming algorithm to find the projective tree with the highest score based on Eq.(2). The algorithm is present in Algorithm 1. The key idea is that we can generalize the definition of $\text{score}(T_{w_0})$ in Eq.(2) to any w by the following definition:

Algorithm 1: Projective Inference

Input : Input sentence $\mathbf{s}$, span candidates $\mathcal{T}$, span scores $\text{score}_{\text{span}}(T), \forall T \in \mathcal{T}$
Output : Highest score of every span $\text{score}(T), \forall T \in \mathcal{T}$

```
/* Compute linking scores based on Eq.(14) */
score_link(T1, T2) ← score_parent(T1|T2) + score_child(T2|T1), ∀(T1, T2) ∈ T
/* Compute score(T), ∀T ∈ T */
for len ← 0 to n do
    for T ← T do
        if T.e − T.s = len then
            /* T covers a single word */
            if len = 0 then
                score(T) ← score_span(T)
            else
                /* C is a set of direct subtrees composing T */
                score(T) ← score_span(T) + max_C(T)(Σ_{Ti∈C}[score(Ti) + λscore_link(T, Ti)])
            end
        end
    end
end
```

$$\begin{aligned} \text{score}(T_w) = &\sum_{T_{w_i} \subseteq T_w} \text{score}_{\text{span}}(T_{w_i}) \\ &+ \lambda \sum_{(w_i \to w_j) \in T_w} \text{score}_{\text{link}}(T_{w_i}, T_{w_j}) \end{aligned} \tag{15}$$

where $\{T_{w_i} \mid T_{w_i} \subseteq T_w, i = 0, 1, ..., n\}$ is all subtrees inside T_w, i.e. there is a path in T_w like

$$w \to w_{i_1} \to ..., \to w_i$$

Using this definition, we can rewrite $\text{score}(T_w)$ in a recursive manner:

$$\begin{aligned} &\text{score}(T_w) = \text{score}_{\text{span}}(T_w) \\ &+ \sum_{T_{w_j} \in \mathcal{C}(T_w)} [\text{score}(T_{w_j}) + \lambda \, \text{score}_{\text{link}}(T_w, T_{w_j})] \end{aligned} \tag{16}$$

where $\mathcal{C}(T_w) = \{T_{w_i} \mid (w \to w_i) \in T_w, i = 0, 1, ...n\}$ is the set of all direct subtrees of T_w.

Non-Projective Decoding It is noteworthy that effectively finding a set of subtrees composing a tree T requires trees to be projective (the projective property guarantees every subtree is a continuous span in text), and experiments in Section 4 show that this algorithm performs well on datasets where most trees are projective, but performs worse when

a number of trees are non-projective. To address this issue, we adapt the proposed strategy to the MST (Maximum Spanning Tree) algorithm (McDonald et al., 2005b). The key point of MST is to obtain the score for each pair of tokens w_i and w_j (rather than spans) , denoted by $\text{score}_{\text{edge}}(w_i, w_j)$. We propose that the score to link w_i and w_j is the highest score achieved by two spans respectively rooted at w_i and w_j:

$$\begin{aligned}\text{score}_{\text{edge}}(w_i, w_j) = \max_{T_{w_i}, T_{w_j}} [&\text{score}_{\text{span}}(T_{w_i}) \\ &+ \text{score}_{\text{span}}(T_{w_j}) + \lambda \text{score}_{\text{link}}(T_{w_i}, T_{w_j})]\end{aligned} \quad (17)$$

The final score for tree T is given by:

$$\text{score}(T) = \sum_{(w_i \rightarrow w_j) \in T} \text{score}_{\text{edge}}(w_i, w_j) \quad (18)$$

Here, MST can be readily used for decoding.

4 Experiments

4.1 Datasets and Metrics

We carry out experiments on three widely used dependency parsing benchmarks: the English Penn Treebank v3.0 (PTB) dataset (Marcus et al., 1993), the Chinese Treebank v5.1 (CTB) dataset (Xue et al., 2002) and the Universal Dependency Treebanks v2.2 (UD) (Nivre et al., 2016) where we select 12 languages for evaluation. We follow Ma et al. (2018) to process all datasets. The PTB dataset contains 39832 sentences for training and 2416 sentences for test. The CTB dataset contains 16091 sentences for training and 1910 sentences for test. The statistics for 12 languages in UD dataset are the same with Ma et al. (2018). We use the unlabeled attachment score (UAS) and labeled attachment score (LAS) for evaluation. Punctuations are ignored in all datasets during evaluation.

4.2 Experiment Settings

We compare the proposed model to the following baselines: (1) **Biaffine**, (2) **StackPTR**, (3)**GNN**, (4) **MP2O**, (5) **CVT**, (6) **LRPTR**, (7) **HiePTR**, (8) **TreeCRF**, (9) **HPSG**, (10) **HPSG+LA**, (11) **MulPTR**, (12) **SynTr**. The details of these baselines are left to the supplementary materials due to page limitation. We group experiments into three categories: without pretrained models, with BERT and with RoBERTa. To implement a span-prediction parsing model without pretrained models, we use the QAnet (Yu et al., 2018) for span prediction. To enable apple-to-apple comparisons, we implement our proposed model, the Biaffine model, MP2O (Wang and Tu, 2020) based on $\text{BERT}_{\text{large}}$ (Devlin et al., 2018) and $\text{RoBERTa}_{\text{large}}$ (Liu et al., 2019) for PTB, BERT and RoBERTa-$\text{wwm}_{\text{large}}$ (Cui et al., 2019) for CTB, BERTBase-Multilingual-Cased and XLM-$\text{RoBERTa}_{\text{large}}$ for UD. We apply both projective decoding and non-projective MST decoding for all datasets. For all experiments, we concatenate 100d POS tag embedding with 1024d pretrained token embeddings, then project them to 1024d using a linear layer. Following Mrini et al. (2020), we further add 1-3 additional encoder layers on top to let POS embeddings well interact with pretrained token embeddings. POS tags are predicted using the Stanford NLP package (Manning et al., 2014). We tried two different types of additional encoders: Bi-LSTM (Hochreiter and Schmidhuber, 1997) and Transformer (Vaswani et al., 2017a). For Bi-LSTM, the number of hidden size is 1024d. For Transformer, the number of attention heads and hidden size remain the same as pretrained models (16 for attention heads and 1024d for hidden size). We use 0.1 dropout rate for pretrained models and 0.3 dropout rate for additional layers. We use Adam (Kingma and Ba, 2014) as optimizer. The weight parameter λ is tuned on the development set. The code is implemented by PyTorch 1.6.0 and MindSpore.

4.3 Main Results

Table 2 compares our model to existing state-of-the-art models on PTB/CTB test sets. As can be seen, for models without pretrained LM, the proposed span-prediction model based on QAnet outperforms all baselines, illustrating the effectiveness of the proposed span-prediction framework for dependency parsing. For BERT-based models, the proposed span-prediction models outperform Biaffine model based on BERT, along with other competitive baselines. On PTB, performances already outperform all previous baselines, except on the LAS metric in comparison to HiePTR (95.46 vs. 95.47) on PTB, but underperform RoBERTa-based models. On CTB, the proposed span-prediction model obtains a new SOTA performance of 93.14% UAS. For RoBERTa-based models, the proposed model achieves a new SOTA performance of 97.24% UAS and 95.49% LAS on PTB. As PTB and CTB contain almost only projective trees, the projective decoding strategy significantly outperforms the non-

	PTB		CTB	
	UAS	LAS	UAS	LAS
with additional labelled constituency parsing data				
MulPTR[♭]	96.06	94.50	90.61	89.51
MulPTR+BERT[♭]	96.91	95.35	92.58	91.42
HPSG[♭]	97.20	95.72	-	-
HPSG+LA[♭]	97.42	96.26	94.56	89.28
without Pretrained Models				
Biaffine	95.74	94.08	89.30	88.23
StackPTR	95.87	94.19	90.59	89.29
GNN	95.87	94.15	90.78	89.50
LRPTR	96.04	94.43	-	-
HiePTR	96.18	94.59	90.76	89.67
TreeCRF	96.14	94.49	-	-
Ours-Proj	**96.42**	**94.71**	**91.15**	**89.68**
Ours-Nproj	96.33	94.60	90.12	89.55
with Pretrained Models				
with BERT				
Biaffine	96.78	95.29	92.58	90.70
MP2O	96.91	95.34	92.55	90.69
SynTr+RNGTr	96.66	95.01	92.98	91.18
HiePTR	97.05	**95.47**	92.70	**91.50**
Ours-Proj	**97.18**	95.46	**93.14**	91.27
Ours-Nproj	97.09	95.35	93.06	91.21
with RoBERTa				
Biaffine	96.87	95.34	92.45	90.48
MP2O	96.94	95.37	92.37	90.40
Ours-Proj	**97.24**	**95.49**	**92.68**	**90.91**
Ours-Nproj	97.14	95.39	92.58	90.83

Table 2: Results for different models on PTB and CTB. [♭]: These approaches utilized both dependency and constituency information in their approach, thus is not comparable to ours.

projective MST algorithm. It is worth noting that, since MulPTR, HPSG and HPSG+LA rely on additional labeled data of constituency parsing, results for HPSG are not comparable to ours. We list them here for reference purposes.

Table 3 compares our model with existing state-of-the-art methods on UD test sets. Other than es, where the proposed model slightly underperforms the SOTA model by 0.02, the proposed model enhanced with XLM-RoBERTa achieves SOTA performances on all other 11 languages, with an average performance boost of 0.3. As many languages in UD have a notable portion of non-projective trees, MST decoding significantly outperforms projective decoding, leading to new SOTA performances in almost all language sets.

5 Ablation Study and Analysis

We use PTB to understand behaviors of the proposed model. As projective decoding works best for PTB, scores reported in this section are all from projective decoding.

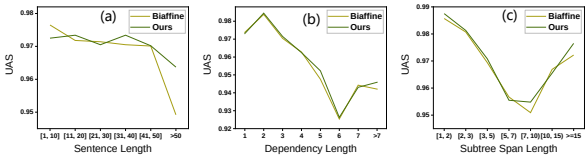

Figure 2: Performances on PTB test dev of Biaffine and our parser w.r.t. sentence length, dependency length and subtree span length.

5.1 Effect of Candidate Span Number

We would like to study the effect of the number of candidate spans proposed by the span proposal module, i.e., the value of k. We vary the value of k from 1 to 25. As shown in Table 4, increasing values of k leads to higher UAS, and the performance stops increasing once k is large enough ($k > 15$). More interestingly, even though k is set to 1, which means that only one candidate span is proposed for each word, the final UAS score is 96.94, a score that is very close to the best result 97.24 and surpasses most existing methods as shown in Table 2. These results verify that the proposed approach can accurately extract and link the dependency spans.

5.2 Effect of Span Retrieval by Span Linking

As shown in Table 5, span recall significantly improves with the presence of the span linking stage. This is in line with our expectation, since spans missing at the proposal module can be retrieved by QA model in the span linking stage. Recall boost narrows down when k becomes large, which is expected as more candidates are proposed at the proposal stage. The span linking stage can improve computational efficiency by using a smaller number of proposed spans while achieving the same performance.

5.3 Effect of Scoring Functions

We study the effect of each part of the scoring functions used in the proposed model. Table 6 shows the results. We have the following observations:
(1) **token(query)-token(answer)**: we simplify the model by only signifying root token in queries (child) and extract the root token in the context (parent). The model actually degenerates into a model similar to Biaffine by working at the token-token level. We observe significant performance decreases, 0.57 in UAS and 0.34 in LAS.
(2) **token(query)-span(answer)**: signifying only token in queries (child) and extracting span in answers (parent) leads to a decrease of 0.13 and 0.08

projective%	bg 99.8	ca 99.6	cs 99.2	de 97.7	en 99.6	es 99.6	fr 99.7	it 99.8	nl 99.4	no 99.3	ro 99.4	ru 99.2	Avg. 99.4
GNN	90.33	92.39	90.95	79.73	88.43	91.56	87.23	92.44	88.57	89.38	85.26	91.20	89.37
						+BERT							
(Sun et al., 2020)	90.88	93.67	91.52	86.64	90.60	93.01	90.65	94.05	92.81	91.05	86.59	92.35	91.15
MP2O	91.30	93.60	92.09	82.00	90.75	92.62	89.32	93.66	91.21	91.74	86.40	92.61	91.02
Biaffine	92.72	93.88	92.70	83.65	91.31	91.89	90.87	94.02	92.24	93.50	88.11	94.37	91.61
Ours-Proj	93.42	93.85	92.90	84.88	91.74	92.05	91.50	94.62	92.42	93.98	88.52	94.50	92.03
Ours-NProj	93.61	94.22	93.48	85.14	91.77	92.50	91.52	94.60	92.82	94.24	88.48	94.73	92.30
						+XLM-RoBERTa							
MP2O	91.42	93.75	92.15	82.20	90.91	92.60	89.51	93.79	91.45	91.95	86.50	92.81	90.75
Biaffine	93.04	94.15	93.57	84.84	91.93	**92.64**	91.64	94.07	92.78	94.17	88.66	94.91	92.15
Ours-Proj	93.61 (+0.57)	94.04 (-0.11)	93.1 (-0.47)	84.97 (+0.13)	91.92 (-0.01)	92.32 (-0.32)	91.69 (+0.05)	**94.86 (+0.79)**	92.51 (-0.27)	94.07 (-0.10)	**88.76 (+0.10)**	94.66 (-0.25)	92.21 (+0.06)
Ours-NProj	**93.76 (+0.72)**	**94.38 (+0.23)**	**93.72 (+0.15)**	**85.23 (+0.39)**	**91.95 (+0.02)**	92.62 (-0.02)	**91.76 (+0.12)**	94.79 (+0.72)	**92.97 (+0.19)**	**94.50 (+0.33)**	88.67 (+0.01)	**95.00 (+0.09)**	**92.45 (+0.30)**

Table 3: LAS for different model on UDv2.2. We use ISO 639-1 codes to represent languages from UD.

k	1	2	5	10	15	20
UAS	96.94	97.10	97.22	97.23	97.24	97.23

Table 4: Effect of number of span candidates

k	1	2	5	10	15
recall w/o link	97.09	98.88	99.57	99.77	99.86
recall w/ link	97.76	99.14	99.69	99.85	99.92

Table 5: Span recall with/without span linking module

Model	UAS	LAS
Full	97.24	95.49
token(query)-token(answer)	96.67(-0.57)	95.15(-0.34)
span(query)-token(answer)	97.17(-0.07)	95.44(-0.05)
token(query)-span(answer)	97.11(-0.13)	95.41(-0.08)
-mutual	97.18(-0.06)	95.43(-0.06)

Table 6: The effect of removing different parts from scoring function

respectively for UAS and LAS.

(3) **span(query)-token(answer)**: signifying spans in queries (child) but only extracting token in answers (parent) leads to a decrease of 0.07 and 0.05 respectively for UAS and LAS. (1), (2) and (3) demonstrate the necessity of modeling span-span rather than token-token relations in dependency parsing: replacing span-based strategy with token-based strategy for either parent or child progressively leads to performance decrease.

(4) Removing the **Mutual Dependency** module which only uses child $\rightarrow$ parent relation and ignores parent $\rightarrow$ child relation also leads to performance decrease.

5.4 Analysis

Following Ma et al. (2018); Ji et al. (2019), we analyze performances of the Biaffine parser and the proposed method with respect to sentence length, dependency length, and subtree span length. Results are shown in Figure 2.

Sentence Length. As shown in Figure 2(a), the proposed parser achieves better performances on long sentences compared with Biaffine. Specially, when sentence length is greater than 50, the performance of the Biaffine parser decreases significantly, while the proposed parser has a much smaller drop (from 0.97 to 0.964).

Dependency Length. Figure 2(b) shows the results with respect to dependency length. The proposed parser shows its advantages on long-range dependencies. We suppose span-level information is beneficial for long-range dependencies.

Subtree Span Length. We further conduct experiments on subtree span length. We divide the average lengths of the two spans in the span linking module into seven buckets. We suppose our parser should show advantages on long subtree span, and the results in Figure 2(c) verify our conjecture.

In summary, the span-span strategy works significantly better than the token-token strategy, especially for long sequences. This explanation is as follows: the token-token strategy can be viewed as a coarse simplification of the span-span strategy, where the root token in the token-token strategy can be viewed as the **average** of all spans covering it, while in the span-span strategy, it represents the exact span, rather than the average. The deviation from the average is relatively small from the extract when sequences are short, but becomes larger as sequence length grows, since the number of spans covering the token exponentially grows with length. This makes the token-token strategy work significantly worse for long sequences.

6 Conclusion

In this paper, we propose to construct dependency trees by directly modeling span-span instead of

token-token relations. We use the machine reading comprehension framework to formalize the span linking module, where one span is used as a query to extract the text span/subtree it should be linked to. Extensive experiments on the PTB, CTB and UD benchmarks show the effectiveness of the proposed method.

Acknowledgement

This work is supported by the Science and Technology Innovation 2030 - "New Generation Artificial Intelligence" Major Project (No. 2021ZD0110201), the Key R & D Projects of the Ministry of Science and Technology (2020YFC0832500) and CAAI-Huawei MindSpore Open Fund. We would like to thank anonymous reviewers for their comments and suggestions.

References

Danqi Chen and Christopher D 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.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Ziqing Yang, Shijin Wang, and Guoping Hu. 2019. Pre-training with whole word masking for chinese bert. *arXiv preprint arXiv:1906.08101*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *arXiv preprint arXiv:1505.08075*.

Jason Eisner. 2000. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. *arXiv preprint arXiv:1903.08445*.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Multitask pointer network for multi-representational parsing. *arXiv preprint arXiv:2009.09730*.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2021. Dependency parsing with bottom-up hierarchical pointer networks. *arXiv preprint arXiv:2105.09611*.

Wenjuan Han, Yong Jiang, and Kewei Tu. 2019. Enhancing unsupervised generative dependency parser with contextual information. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5315–5325, Florence, Italy. Association for Computational Linguistics.

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

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy. Association for Computational Linguistics.

Zixia Jia, Youmi Ma, Jiong Cai, and Kewei Tu. 2020. Semi-supervised semantic dependency parsing using CRF autoencoders. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6795–6805, Online. Association for Computational Linguistics.

Diederik P 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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*

Papers), pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60.

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

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of human language technology conference and conference on empirical methods in natural language processing*, pages 523–530.

Alireza Mohammadshahi and James Henderson. 2020a. Graph-to-graph transformer for transition-based dependency parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3278–3289, Online. Association for Computational Linguistics.

Alireza Mohammadshahi and James Henderson. 2020b. Recursive non-autoregressive graph-to-graph transformer for dependency parsing with iterative refinement. *arXiv preprint arXiv:2003.13118*.

Khalil Mrini, Franck Dernoncourt, Quan Hung Tran, Trung Bui, Walter Chang, and Ndapa Nakashole. 2020. Rethinking self-attention: Towards interpretability in neural parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 731–742, Online. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Wenzhe Pei, Tao Ge, and Baobao Chang. 2015. An effective neural network model for graph-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 313–322, Beijing, China. Association for Computational Linguistics.

Kailai Sun, Zuchao Li, and Hai Zhao. 2020. Cross-lingual universal dependency parsing only from one monolingual treebank. *arXiv preprint arXiv:2012.13163*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017a. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017b. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

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)*, pages 2306–2315, Berlin, Germany. Association for Computational Linguistics.

Wenhui Wang, Baobao Chang, and Mairgup Mansur. 2018. Improved dependency parsing using implicit word connections learned from unlabeled data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2857–2863, Brussels, Belgium. Association for Computational Linguistics.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. Association for Computational Linguistics.

Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated Chinese corpus. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*.

Yunzhe Yuan, Yong Jiang, and Kewei Tu. 2019. Bidirectional transition-based dependency parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7434–7441.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

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*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics.

Zhisong Zhang, Xuezhe Ma, and Eduard Hovy. 2019. An empirical investigation of structured output modeling for graph-based neural dependency parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5592–5598, Florence, Italy. Association for Computational Linguistics.

Zhisong Zhang and Hai Zhao. 2015. High-order graph-based neural dependency parsing. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 114–123.

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 (Volume 1: Long Papers)*, pages 1382–1392, Berlin, Germany. Association for Computational Linguistics.

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.

Junru Zhou and Hai Zhao. 2019. Head-Driven Phrase Structure Grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

A Baselines

We use the following baselines:

- **Biaffine**: Dozat and Manning (2016) fed pairs of words into a biaffine classifier to determine the dependency relations between them.
- **StackPTR**: Ma et al. (2018) combined pointer network with transition-based method to make it benefits from the information of whole sentence and all previously derived subtree structures.
- **GNN**: Ji et al. (2019) used graph neural networks (GNN) to learn token representations for graph-based dependency parsing.
- **MP2O**: Wang and Tu (2020) used message passing to integrate second-order information to biaffine backbone.
- **CVT**: Clark et al. (2018) proposed Cross-View Training, a semi-supervised approach to improve model performance.
- **LRPTR**: Fernández-González and Gómez-Rodríguez (2019) also took advantage of pointer networks to implement transition-based parser, which contains only n actions and is more efficient than StackPTR.
- **HiePTR**: Fernández-González and Gómez-Rodríguez (2021) introduced structural knowledge to the sequential decoding of the left-to-right dependency parser with Pointer Networks.
- **TreeCRF**: Zhang et al. (2020) presented a second-order TreeCRF extension to the biaffine parser.
- **HPSG**: Zhou and Zhao (2019) used head-driven phrase structure grammar to jointly train constituency and dependency parsing.
- **HPSG+LA**: Mrini et al. (2020) added a label attention layer to HPSG to improve model performance. HPSG+LA also relies on the additional constituency parsing dataset.
- **MulPTR**: Fernández-González and Gómez-Rodríguez (2020) jointly trained two separate decoders responsible for constituent parsing and dependency parsing.
- **SynTr**: Mohammadshahi and Henderson (2020b) proposed recursive non-autoregressive graph-to-graph Transformers for the iterative refinement of dependency graphs conditioned on the complete graph.