Graph-based Dependency Parsing with Graph Neural Networks

Tao Ji , Yuanbin Wu , and Man Lan

Department of Computer Science and Technology,
East China Normal University
{taoji.cs}@gmail.com {ybwu,mlan}@cs.ecnu.edu.cn

Abstract

We investigate the problem of efficiently incorporating high-order features into neural graph-based dependency parsing. Instead of explicitly extracting high-order features from intermediate parse trees, we develop a more powerful dependency tree node representation which captures high-order information concisely and efficiently. We use graph neural networks (GNNs) to learn the representations and discuss several new configurations of GNN's updating and aggregation functions. Experiments on PTB show that our parser achieves the best UAS and LAS on PTB (96.0%, 94.3%) among systems without using any external resources.

1 Introduction

In recent development of dependency parsers, learning representations is gaining in importance. From observed features (words, positions, POS tags) to latent parsing states, building expressive representations is shown to be crucial for getting accurate and robust parsing performances.

Here we focus on graph-based dependency parsers. Given a sentence, a parser first scores all word pairs about how possible they hold valid dependency relations, and then use decoders (e.g., greedy, maximum spanning tree) to generate a full parse tree from the scores. The score function is a key component in graph-based parses. Commonly, a neural network is assigned to learn low dimension vectors for words (i.e., nodes of parse trees), and the score function depends on vectors of the word pair (e.g., inner products). The main task of this paper is to explore effective encoding systems for dependency tree nodes.

Two remarkable prior works on node representation are recurrent neural networks (RNNs) (Kiperwasser and Goldberg, 2016b) and biaffine mappings (Dozat and Manning, 2017). RNNs are powerful tools to collect sentence-level information, but the representations ignore features related to dependency structures. The biaffine mappings improve vanilla RNNs via a key observation: the representation of a word should be different regarding whether it is a head or a dependent (i.e., dependency tree edges are directional). Therefore, Dozat and Manning (2017) suggest distinguishing head and dependent vector of a word. Following this line of thought, it is natural to ask whether we can introduce more structured knowledge into node representations. In other words, if biaffine mappings encode the first order parent-children relations, can we incorporate other high-order relations (such as grandparents and siblings)?

In this work, we propose to use graph neural networks (GNNs) for learning dependency tree node representations. Given a weighted graph, a GNN embeds a node by recursively aggregating node representations of its neighbours. For the parsing task, we build GNNs on weighted complete graphs which are readily obtained in graph-based parsers. The graphs could be fixed in prior or revised during the parsing process. By stacking multiple layers of GNNs, the representation of a node gradually collects various high-order information and bring global evidence into decoders' final decision.

Comparing with recent approximate high-order parsers (Kiperwasser and Goldberg, 2016b; Zheng, 2017; Ma et al., 2018), GNNs extract high-order information in a similar incremental manner: node representations of a GNN layer are computed based on outputs of former layers. However, the main difference is that, instead of extracting high-order features on only one intermediate tree, the update of GNN node vectors is able to inspect all intermediate trees. Thus, it may reduce the influence of a suboptimal intermediate parsing result.

Comparing with the syntactic graph network

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2475–2485
Florence, Italy, July 28 - August 2, 2019.

(Marcheggiani and Titov, 2017; Bastings et al., 2017; Zhang et al., 2018b) which runs GNNs on dependency trees given by external parsers, we use GNNs to build the parsing model. And instead of using different weight matrices for outgoing and ingoing edges, our way of handling directional edges is based on the separation of head and dependent representations, which requires new protocols for updating nodes.

We discuss various configurations of GNNs, including strategies on neighbour vector aggregations, synchronized or asynchronized node vector update and graphs with different edge weights. Experiments on the benchmark English Penn Treebank 3.0 and CoNLL2018 multilingual parsing shared task show the effectiveness of the proposed node representations, and the result parser is able to achieve state-of-the-art performances.

To summarize, our major contributions include:

1. introducing graph neural networks to dependency parsing, which aims to efficiently encode high order information in dependency tree node representations.

2. investigating new configurations of GNNs for handling direct edges and nodes with multiple representations.

3. achieving state-of-the-art performances on PTB 3.0 (96.0% UAS, 94.3% LAS).

2 Basic Node Representations

In this section, we review word encoding systems used in recurrent neural networks and biaffine mappings. Our GNN encoder (Section 3) will base on these two prior works. [1]

Given a sentence $s = w_1, \ldots, w_n$, we denote a dependency tree of s to be $T = (V, E)$, where the node set V contains all words and a synthetic root node 0, and the edge set E contains node pairs (i, j, r) which represents a dependency relation r between w_i (the head) and w_j (the dependent). Following the general graph-based dependency parsing framework, for every word pair (i, j), a function $\sigma(i, j)$ assigns it a score which measures how possible is w_i to be the head of w_j. [2] We denote G to be the directed complete graph in which all nodes in V are connected with weights given by σ. The correct tree T is obtained from G using a decoder (e.g., dynamic programming (Eisner, 1996), maximum spanning tree (McDonald et al., 2005), and greedy algorithm (Zhang et al., 2017)).

In neural-network-based models, the score function $\sigma(i, j)$ usually relies on vector representations of nodes (words) i and j. How to get informative encodings of tree nodes is important for training the parser. Basically, we want the tree node encoder to explore both the surface form and deep structure of the sentence.

To encode the surface form of s, we can use recurrent neural networks (Kiperwasser and Goldberg, 2016b). Specifically, we apply a bidirectional long short-term memory network (biLSTM, (Hochreiter and Schmidhuber, 1997)). At each sentence position i, a forward LSTM chain (with parameter $\overrightarrow{\theta}$) computes a hidden state vector $\overrightarrow{\boldsymbol{c}}_i$ by collecting information from the beginning of s to the current position i. Similarly, a backward LSTM chain ($\overleftarrow{\theta}$) collects information $\overleftarrow{\boldsymbol{c}}_i$ from the end of s to the position i:

$$\overrightarrow{\boldsymbol{c}}_i = \text{LSTM}(\boldsymbol{x}_i, \overrightarrow{\boldsymbol{c}}_{i-1}; \overrightarrow{\theta}),\ \overleftarrow{\boldsymbol{c}}_i = \text{LSTM}(\boldsymbol{x}_i, \overleftarrow{\boldsymbol{c}}_{i+1}; \overleftarrow{\theta}),$$

where $\boldsymbol{x}_i$ is the input of a LSTM cell which includes a randomly initialized word embedding $\boldsymbol{e}(w_i)$, a pre-trained word embedding $\boldsymbol{e}'(w_i)$ from Glove (Pennington et al., 2014) and a trainable embedding of w_i's part-of-speech tag $\boldsymbol{e}(pos_i)$,

$$\boldsymbol{x}_i = \big(\boldsymbol{e}(w_i) + \boldsymbol{e}'(w_i)\big) \oplus \boldsymbol{e}(pos_i).$$

Then, a context-dependent node representation of word i is the concatenation of the two hidden vectors,

$$\boldsymbol{c}_i = \overrightarrow{\boldsymbol{c}}_i \oplus \overleftarrow{\boldsymbol{c}}_i. \tag{1}$$

With the node representations, we can define the score function σ using a multi-layer perceptron $\sigma(i, j) = \text{MLP}(\boldsymbol{c}_i \oplus \boldsymbol{c}_j)$ (Pei et al., 2015), or using a normalized bilinear function ($A, \boldsymbol{b}_1, \boldsymbol{b}_2$ are parameters),

$$\begin{aligned}\sigma(i, j) &= \text{Softmax}_i\left(\boldsymbol{c}_i^{\intercal} A \boldsymbol{c}_j + \boldsymbol{b}_1^{\intercal} \boldsymbol{c}_i + \boldsymbol{b}_2^{\intercal} \boldsymbol{c}_j\right) \\ &\triangleq P(i|j),\end{aligned} \tag{2}$$

[1] Following the convention of (Dozat and Manning, 2017), we use lowercase italic letters for scalars and indices, lowercase bold letters for vectors, uppercase italic letters for matrices.

[2] We will focus on the unlabelled parsing when illustrating our parsing models. For predicting labels, we use the identical setting in (Dozat and Manning, 2017).

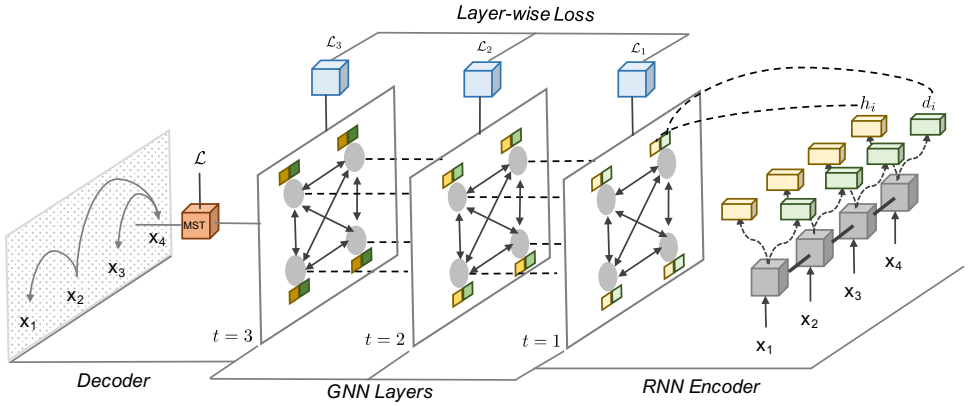


Figure 1: The GNN architecture. "RNN Encoder"+"Decoder" is equal to the Biaffine parser. For the "GNN Layers", each layer is based on a complete weighted graph, and the weights are supervised by the layer-wise loss.

which is actually a distribution on j's head words.

We note that from the RNN encoder, a node only obtains one vector representation. But as the dependency tree edges have directions, a word plays a different role regarding it is the head or the dependent in an edge. Thus, instead of using one vector representation, we employ two vectors to distinguish the two roles (Dozat and Manning, 2017). Concretely, based on $\boldsymbol{c}_i$, we use two multi-layer perceptrons to generate two different vectors,

$$\boldsymbol{h}_i = \text{MLP}_h(\boldsymbol{c}_i), \quad \boldsymbol{d}_i = \text{MLP}_d(\boldsymbol{c}_i).$$

The score funcion in Equation 2 now becomes

$$\sigma(i,j) = \text{Softmax}_i\left(\boldsymbol{h}_i^\intercal A \boldsymbol{d}_j + \boldsymbol{b}_1^\intercal \boldsymbol{h}_i + \boldsymbol{b}_2^\intercal \boldsymbol{d}_j\right) \quad (3)$$

The main task we will focus on in following sections is to further encode deep structure of s to node vectors $\boldsymbol{h}_i$ and $\boldsymbol{d}_i$. Specifically, besides the parent-child relation, we would like to consider high-order dependency relations such as grandparents and siblings in the score function σ.

3 Node Representation with GNNs

3.1 The GNN Framework

We first introduce the general framework of graph neural network. The setting mainly follows the graph attention network (Velikovi et al., 2018). [3] Given a (undirected) graph G, a GNN is a multi-layer network. At each layer, it maintains a set of node representations by aggregating information from their neighbours.

[3]There are other variants of GNNs. See (Battaglia et al., 2018) for a more general definition.

Formally, let $\mathcal{N}(i)$ be neighbours of node i in G. We denote $\boldsymbol{v}_i^t$ to be the vector representation of i at the t-th GNN layer. $\boldsymbol{v}_i^t$ is obtained by

$$\boldsymbol{v}_i^t = g\left(W \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^t \boldsymbol{v}_j^{t-1} + B\boldsymbol{v}_i^{t-1}\right), \quad (4)$$

where g is a non-linear activation function (we use LeakyReLU with negative input slope 0.1), W and B are parameter matrices. We use different edge weights α_{ij}^t, which is a function of $\boldsymbol{v}_i^{t-1}$ and $\boldsymbol{v}_j^{t-1}$, to indicate different contributions of node j in building $\boldsymbol{v}_i^t$. The update Equation 4 reads that the new representation $\boldsymbol{v}_i^t$ contains both the previous layer vector $\boldsymbol{v}_i^{t-1}$ and a weighted aggregation of neighbour vectors $\boldsymbol{v}_j^{t-1}$.

We can see that the GNN naturally catches multi-hop (i.e., high-order) relations. Taking the first two layers for example, for every node i at the second layer, $\boldsymbol{v}_i^2$ contains information of its 1-hop neighbours $\boldsymbol{v}_j^1$. Since $\boldsymbol{v}_j^1$ has already encoded its own 1-hop neighbours at the first layer, $\boldsymbol{v}_i^2$ actually encodes information of its 2-hop neighbours. Inspired by this observation, we think GNNs may help parsing with high-order features.

On the other side, to parse with GNNs, instead of encoding one vector for each node, we need to handle the head representation $\boldsymbol{h}_i$ and the dependent representation $\boldsymbol{d}_i$ simultaneously on a directed graph G.

Furthermore, to approximate the exact high-order parsing (Eisner, 1996; McDonald and Pereira, 2006), we need each GNN layer to have a concrete meaning regarding parsing the sentence. For example, we could consider complete graphs

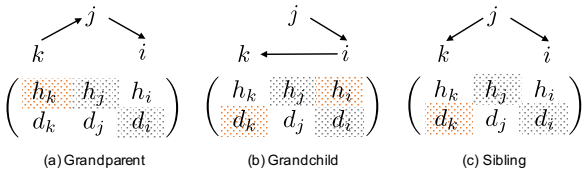


Figure 2: Three types of high-order information integrated in the parent-child pair (j, i). The grey shadows indicate which node representations already exist in first order feature. The orange shadows indicate which node representations should to be included for each high-order feature. Notice that k is actually a weighted sum of all applicable nodes (*soft*). Subfigure (a) helps to understand Equation 6. Since k acts as parent of j, to capture grandparent feature, $\boldsymbol{h}_j$ should additionally contains information of $\boldsymbol{h}_k$. Subfigure (c) helps to understand Equation 7. Since k acts as child of j, to capture sibling feature, $\boldsymbol{h}_j$ should additionally contains information of $\boldsymbol{d}_k$.

(i.e., all nodes are connected) and set edge weights using conditional probabilities,

$$\alpha_{ij}^t = \sigma^t(i, j) = P^t(i|j), \tag{5}$$

which is Equation 3 evaluated at layer t. [4] Thus, the graph at each layer appears as a "soft" parse tree, and the aggregated information would approximate high-order features on that tree. Comparing with existing incremental parsers which maintain only one intermediate tree ("hard"), the "soft" trees represented by GNN layers contain more information. In fact, the graphs keep all information to derive any intermediate parse trees. Therefore, it may reduce the risk of extracting high-order features on suboptimal intermediates. We detail the GNN model in the following.

3.2 High-order Information

Given a node i, we mainly focus on three types of high-order information, namely, grandparents, grandchildren and siblings. We need to adapt the general GNN update formula to properly encode them into node representations.

First, for incorporating grandparent information (Figure 2.a), we expect $\sigma^t(j, i)$, which depends on the head vector of j and the dependent vector of i, not only considers the parent-child pair (j, i), but also consults the ("soft") parent of j suggested by the previous layer (denoted by k). Specifically, the new head representation of node j should examine representations of its neighbors when they act as parents of j. In other word, we will update $\boldsymbol{h}_j^t$ using $\boldsymbol{h}_k^{t-1}$. Similarly, for encoding grandchildren of j in $\sigma^t(j, i)$ (also denoted by k), we need the new dependent representation of node i examine its neighbors when they act as children of i. Thus, we will update $\boldsymbol{d}_i^t$ using $\boldsymbol{d}_k^{t-1}$. It suggests the following protocol,

$$\begin{cases} \boldsymbol{h}_i^t = g\Big(W_1 \sum\limits_{j \in \mathcal{N}(i)} \alpha_{ji}^t \boldsymbol{h}_j^{t-1} + B_1 \boldsymbol{h}_i^{t-1}\Big) \\ \boldsymbol{d}_i^t = g\Big(W_2 \sum\limits_{j \in \mathcal{N}(i)} \alpha_{ij}^t \boldsymbol{d}_j^{t-1} + B_2 \boldsymbol{d}_i^{t-1}\Big). \end{cases} \tag{6}$$

Note that we use α_{ji}^t in updating $\boldsymbol{h}_i^t$ and α_{ji}^t in updating $\boldsymbol{d}_i^t$ which is according to the probabilistic meaning of the weights.

On the other side, for extracting siblings of i (again denoted by k) in (j, i) (Figure 2.c), the new head representation of node j should examine representations of its neighbors when they act as dependents of j. We expect the update of $\boldsymbol{h}_j^t$ involving $\boldsymbol{d}_k^{t-1}$ It suggests our second update protocol [5],

$$\begin{cases} \boldsymbol{h}_i^t = g\Big(W_1 \sum\limits_{j \in \mathcal{N}(i)} \alpha_{ij}^t \boldsymbol{d}_j^{t-1} + B_1 \boldsymbol{h}_i^{t-1}\Big) \\ \boldsymbol{d}_i^t = g\Big(W_2 \sum\limits_{j \in \mathcal{N}(i)} \alpha_{ji}^t \boldsymbol{h}_j^{t-1} + B_2 \boldsymbol{d}_i^{t-1}\Big). \end{cases} \tag{7}$$

We can integrate Equation 6 and 7 in a single update which handles grandparents, grandchildren and siblings in an uniform way,

$$\begin{cases} \boldsymbol{h}_i^t = g\Big(W_1 \sum\limits_{j \in \mathcal{N}(i)} (\alpha_{ji}^t \boldsymbol{h}_j^{t-1} + \alpha_{ij}^t \boldsymbol{d}_j^{t-1}) + B_1 \boldsymbol{h}_i^{t-1}\Big) \\ \boldsymbol{d}_i^t = g\Big(W_2 \sum\limits_{j \in \mathcal{N}(i)} (\alpha_{ij}^t \boldsymbol{h}_j^{t-1} + \alpha_{ji}^t \boldsymbol{d}_j^{t-1}) + B_2 \boldsymbol{d}_i^{t-1}\Big). \end{cases} \tag{8}$$

Comparing with the general GNNs, above node vector updates are tailored to the parsing task using high-order feature rules. We think exploring the semantics of representations and graph weights would provide useful guidance in design of GNNs for specific tasks. Finally, besides the default synchronized setting, we also investigate asynchronized version of Equation 8,

$$\begin{cases} \boldsymbol{h}_i^{t-\frac{1}{2}} = g\Big(W_1 \sum\limits_{j \in \mathcal{N}(i)} (\alpha_{ji}^t \boldsymbol{h}_j^{t-1} + \alpha_{ij}^t \boldsymbol{d}_j^{t-1}) + B_1 \boldsymbol{h}_i^{t-1}\Big) \\ \boldsymbol{d}_i^t = g\Big(W_2 \sum\limits_{j \in \mathcal{N}(i)} (\alpha_{ij}^t \boldsymbol{h}_j^{t-\frac{1}{2}} + \alpha_{ji}^t \boldsymbol{d}_j^{t-1}) + B_2 \boldsymbol{d}_i^{t-1}\Big), \end{cases} \tag{9}$$

where we first update $\boldsymbol{h}$, and then use the updated $\boldsymbol{h}$ to update $\boldsymbol{d}$.

[4] The model adds layer-wise loss functions to approach Equation 5, see Section 3.5.

[5] The update of $\boldsymbol{d}_i^t$ in Equation 7 tries to include knowledge of other candidate heads of i. It does not correspond to a high-order feature, but for building a symmetric formula, we just include it in that way.

3.3 Graph Weights

In the graph-based parsing, the topology structure of G is mainly determined by edge weights α_{ij}^t. In fact, we usually work on a complete graph to obtain a parse tree. Thus, how to design α_{ij}^t is important to apply GNNs. As mentioned above, we can set α_{ij}^t equals to probability $P^t(i|j)$. In this section, we explore more settings on α_{ij}^t.

First, instead of using the "soft" tree setting, we can assign $\{0, 1\}$ values to α_{ij}^t to obtain a sparse graph,

$$\alpha_{ij}^t = \begin{cases} 1, & i = \arg\max_{i'} P^t(i'|j) \\ 0, & \text{otherwise} \end{cases}, \quad (10)$$

In this setting, a node only looks at the head node with the highest probability.

An extension of Equation 10 is to consider top-k head nodes, which could include more neighbourhood information. Defining $\mathcal{N}_k^t(j)$ be a set of nodes with top-k $P^t(i|j)$ for node j, we re-normalize Equation 3 on this set and assign them to α_{ij}^t,

$$\alpha_{ij}^t = \begin{cases} \text{Softmax}_i\left(\boldsymbol{h}_i^\intercal A \boldsymbol{d}_j + \boldsymbol{b}_1^\intercal \boldsymbol{h}_i + \boldsymbol{b}_2^\intercal \boldsymbol{d}_j\right), i \in \mathcal{N}_k^t(j) \\ 0, \quad \text{otherwise} \end{cases} \quad (11)$$

Finally, for comparison, one can ignore $P^t(i|j)$ and see each neighbour equally at each layer,

$$\alpha_{ij}^t = \frac{1}{n}, \quad \forall j \in V, i \in V/\{j\}. \quad (12)$$

3.4 Decoding

Given node representations and $P(i|j)$, to build the final parse tree, we can either greedily set the head of w_j to $\arg\max_i P(i|j)$ which is fast for decoding but may output an ill-formed tree, or use a MST algorithm on all word pairs with weight $P(i|j)$, which forms a valid tree but could be slower.

To predict labels of dependency edges, we introduce $P(r|i, j)$ which measures how possible a tree (i, j) holds a dependency relation r using another MLP. The setting is identical to the biaffine parser (Dozat and Manning, 2017).

3.5 Training

Given the gold standard tree T, the training objective consists of two parts. First, we have a decoder behind the final GNN layer (denote by τ) which will perform decoding on both tree structures (using $P^\tau(i|j)$) and edge labels (using $P(r|i, j)$). The loss from the final classifier is negative log-likelihood of T,

$$\mathcal{L}_0 = -\frac{1}{n} \sum_{(i,j,r)\in T} \left(\log P^\tau(i|j) + \log P(r|i, j)\right).$$

Second, as mentioned in Section 3.1, we can provide supervision on $P^t(i|j)$ from each GNN layer (only on the tree structure, intermediate loss on labels are ignored). The *layer-wise* loss is

$$\mathcal{L}' = \sum_{t=1}^{\tau} \mathcal{L}_t = \sum_{t=1}^{\tau} -\frac{1}{n} \sum_{(i,j,r)\in T} \log P^t(i|j).$$

The objective is to minimize a weighted combination of them $L = \lambda_1 \mathcal{L}_0 + \lambda_2 \mathcal{L}'$.

4 Experiments

We evaluate the proposed framework on the Stanford Dependency (SD) conversion of the English Penn Treebank (PTB 3.0) and the Universal Dependencies (UD 2.2) (Nivre et al., 2018) treebanks used in CoNLL 2018 shared task(Zeman et al., 2018). For English, we use the standard train/dev/test splits of PTB (train=§2-21, dev=§22, test=§23), POS tags were assigned using the Stanford tagger with 10-way jackknifing of the training corpus (accuracy ≈ 97.3%). For 12 languages selected from UD 2.2, we use CoNLL 2018 shared task's official train/dev/test splits, POS tags were assigned by the UDPipe (Straka et al., 2016).

Parsing performance is measured with five metrics. We report unlabeled (UAS) and labeled attachment scores (LAS), unlabeled (UCM) and labeled complete match (LCM), and label accuracy score (LA). For evaluations on PTB, following (Chen and Manning, 2014), five punctuation symbols (" " : , .) are excluded from the evaluation. For CoNLL 2018 shared task, we use the official evaluation script.

All basic hyper-parameters are the same as those reported in Dozat and Manning (2017), which means that our baseline system without GNN layers is a re-implementation of the Biaffine parser. For GNN models, the only new parameters are matrices in $P^t(i|j)$ and matrices in GNN units. The λ_1, λ_2 in objective L is set to $\lambda_1 = 1, \lambda_2 = 0.5$. The hyper-parameters of our default settings are summarized in Appendix A.

The default setting for our final parser is a 2-layer GNN model that uses $hd \triangleright h$ (Equation 8)

Parser		Test UAS	Test LAS
(Chen and Manning, 2014)	T	91.8	89.6
(Dyer et al., 2015)	T	93.1	90.9
(Ballesteros et al., 2016)	T	93.56	92.41
(Weiss et al., 2015)	T	94.26	91.42
(Andor et al., 2016)	T	94.61	92.79
(Ma et al., 2018) §	T	95.87	94.19
(Kiperwasser and Goldberg, 2016a) §	G	93.0	90.9
(Kiperwasser and Goldberg, 2016b)	G	93.1	91.0
(Wang and Chang, 2016)	G	94.08	91.82
(Cheng et al., 2016)	G	94.10	91.49
(Kuncoro et al., 2016)	G	94.26	92.06
(Zheng, 2017) §	G	95.53	93.94
(Dozat and Manning, 2017)	G	95.74	94.08
Baseline	G	95.68	93.96
Our Model §	G	**95.97**	**94.31**

Table 1: Results on the English PTB dataset. The § indicates parsers using high-order features. "T" represents transition-based parser, and "G" represents a graph-based parser.

GNN Layer	GNN Model	Dev UAS	Dev LAS	Test UAS	Test LAS
$l = 0$	Baseline	95.58	93.74	95.68	93.96
$l = 1$	$d \triangleright h$	95.75	93.84	95.83	94.15
	$h \triangleright h$	**95.78**	93.80	**95.91**	94.12
	$hd \triangleright h$	95.77	**93.87**	95.88	**94.23**
$l = 2$	$d \triangleright h$	**95.80**	93.85	95.88	94.17
	$h \triangleright h$	95.77	93.83	95.85	94.13
	$hd \triangleright h$	95.79	**93.90**	**95.92**	**94.24**
$l = 3$	$d \triangleright h$	95.74	93.78	95.87	94.14
	$h \triangleright h$	**95.75**	93.80	95.90	94.15
	$hd \triangleright h$	95.71	**93.82**	**95.93**	**94.22**

Table 2: Impact of l and different high-order information integration methods on PTB dataset. "$d \triangleright h$" corresponds to the Equation 7, "$h \triangleright h$" corresponds to the Equation 6, "$hd \triangleright h$" corresponds to the Equation 8.

aggregating function and "H-first" asynchronous update method (Equation 9). [6]

4.1 Main Results

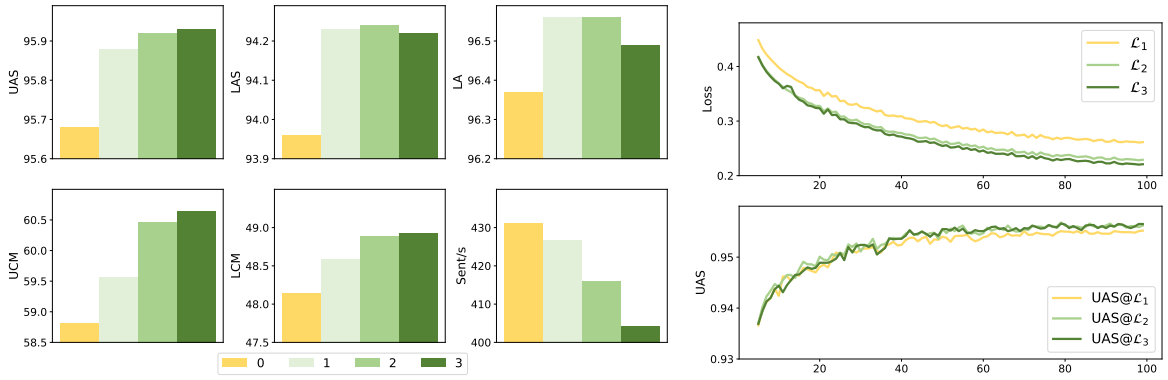
Firstly, we compare our method with previous work (Table 1). The first part contains transition-based models, the second part contains graph-based models and the last part includes three models with integrated hard high-order features. In general, our proposed method achieves significant improvements over our baseline biaffine parser and matches state-of-the-art models. In particular, it achieves 0.29 percent UAS and 0.35 percent LAS improvement over the baseline parser, and 0.1 percent UAS and 0.12 percent LAS improvement over the strong transition-based parser (Ma et al., 2018). It shows that our method can boost the performance of graph-based dependency parser using the global and soft high-order information by the GNN architecture.

Secondly, we analyze different aggregating functions when capturing high-order information. (Table 2). We have some observations regarding this results. Model $hd \triangleright h$ (Equation 8) integrates high-order information of grandparents, grandchildren and siblings. Under all layer settings (1 to 3), its LAS is always better than $h \triangleright h$ (Equation 6) model and $d \triangleright h$ (Equation 7) model, which separately describe high-order information. However, UAS is not sensitive to different ways of aggregating.

[6] Our implementation is publicly available at: https://github.com/AntNLP/gnn-dep-parsing

Thirdly, we analyze the contributions and effects of the number of GNN layers (Figure 3 (a)). From the computation of GNNs, the more layers, the higher order of information is captured. The experimental results show that the 1-layer model significantly outperforms 0-layer model on all five scoring metrics. But continuing to increase the number of layers does not significantly improve performance. Previous work (Zheng, 2017) has shown that the introduction of more than second-order information does not significantly improve parsing performances. Our results also present a consistent conclusion. Specifically, on UAS, LAS and LA, the 2-layer model has the highest sum of scores. On UCM and LCM, performance increases as the number of layers increases, showing the superiority of using high-order information in complete sentence parsing. In addition to parsing performance, we also focus on the speed. We observe that adding one layer of GNN slows down the prediction speed by about 2.1%. The 2-layer model can process 415.9 sentences per second on a single GPU. Its impact on the training process is also slight, increasing from 3 minutes to 3.5 minutes per epoch.

We futher examine different performance of each layer in a 3-layer model (Figure 3 (b)). We observe that, as we move to a higher layer, the average loss decreases during the training process ($\mathcal{L}_3 < \mathcal{L}_2 < \mathcal{L}_1$). The figure shows that the introduction of high-order information leads to more accurate graph weights. We also do the MST decoding directly based on the graph weights on each layer and compare their development set UAS performances. From the layer-wise UAS



(a) (b)

Figure 3: (a) Parsing performance and speed of different layers of our $hd \triangleright h$ model on the test set. (b) Layer-wise training loss and development set's UAS of our 3-layer $hd \triangleright h$ model.

GNN Layer	GNN Model	Dev		Test	
		UAS	LAS	UAS	LAS
$l = 2$	Synch	95.79	93.90	95.92	94.24
	H-first	**95.88**	**93.94**	**95.97**	**94.31**
	D-first	95.78	93.91	95.95	94.27

Table 3: Impact of different GNN update methods on PTB dataset. "Synch" is our default synchronized setting (Equation 8). "H-first" is an asynchronous update method that first updates head word representation (Equation 9). Similarly, the "D-first" model first updates dependent word representation.

GNN Layer	GNN Model	Dev		Test	
		UAS	LAS	UAS	LAS
$l = 2$	All=1	95.71	93.73	95.76	94.07
	Hard-1	95.69	93.70	95.80	94.13
	Hard-2	95.73	93.78	**95.90**	**94.20**
	Hard-3	**95.81**	**93.88**	95.88	**94.20**
$l = 2$	Soft	**95.88**	**93.94**	**95.97**	**94.31**

Table 4: Impact of different kinds of graph weights on PTB dataset. "All=1" means setting all weights to 1 (Equation 12), "Hard-k" means renormalization at the top-k weights of each node (Equation 11), "Soft" is our default model setting (Equation 8).

results, we observe that the difference between 2-layer and 3-layer is not obvious, but both are higher than the 1-layer.

Fourthly, we present the influences of synchronized/asynchronized GNN update methods (Table 3). We first compare the synchronous update and asynchronous update methods. It shows that the later one works better without adding extral parameters. The reason may be that asynchronous methods aggregate high-order information earlier. The H-first model (Equation 9) is slightly better than the D-first model. This may indicate that dependent representation is more important than head representation, since the first updated representation will improve the representation of the late update,

Fifthly, we experiment with unweighted graph (all set to 1) and hard weight graph (renormalized at top-k) (Table 4). A GNN based on completely unweighted graph is equivalent to uniformly incorporating representations of all neighbors for each node in the sentence, and similar to incorporating sentence embedding. Experiments show that this approach will hurt the performance of the parser. For the Hard-k model (Equation 11), when k is equal to 1, it is equivalent to a GNN based on greedy decoding results, when k is equal to the sentence length, it is equivalent to our soft method. Experiments show that as k increases from 1 to 3, the performance of the Hard-k model is gradually improved. We also observe that hard weights affect the training stability of the parser.

Finally, we report the results of our model on partial UD treebanks on the CoNLL 2018 shared task (Table 5). Our model uses only word and XPOS tag (predict by UDPipe), without any cross lingual features. [7] We use FastText multilingual pretrained vectors instead of Glove vectors. [8] The results show that our GNN parser performs better on 10 UD 2.2 treebanks. For *bg*, our parser does not improve performance. For *nl*, our parser improves 0.22 UAS, although LAS is slightly lower

[7] The results should not compare with the shared task's official results.

[8] https://github.com/facebookresearch/fastText

UD 2.2	Baseline Parser		GNN Parser	
	UAS	LAS	UAS	LAS
bg	**91.69**	88.25	91.64	**88.28**
ca	92.08	89.75	**92.12**	**89.90**
cs	91.22	88.73	**92.00**	**89.85**
de	86.11	81.86	**86.47**	**81.96**
en	83.72	81.07	**83.83**	**81.16**
es	90.95	88.65	**91.28**	**88.93**
fr	86.46	83.15	**86.82**	**83.73**
it	90.70	88.80	**90.81**	**88.91**
nl	87.72	**84.85**	**87.94**	84.82
no	88.27	85.97	**88.57**	**86.33**
ro	89.07	84.18	**89.11**	**84.44**
ru	88.67	86.29	**88.94**	**86.62**
Avg.	88.89	85.96	**89.13**	**86.24**

Table 5: UAS and LAS F1 scores on 12 UD 2.2 test sets from CoNLL 2018 shared task.

than the baseline parser. For average performance, it achieves 0.24 percent UAS and 0.28 percent LAS improvement over the baseline parser.

4.2 Error Analysis

Following McDonald and Nivre (2011); Ma et al. (2018), we characterize the errors made by the baseline biaffine parser and our GNN parser. Analysis shows that most of the gains come from the difficult cases (e.g. long sentences or long-range dependencies), which represents an encouraging sign of the proposed method's benefits.

Sentence Length. Figure 4 (a) shows the accuracy relative to sentence length. Our parser significantly improves the performance of the baseline parser on long sentence, but is slightly worse on short sentence (length ≤ 10).

Dependency Length. Figure 4 (b) shows the precision and recall relative to dependency length. Our parser comprehensively and significantly improves the performance of the baseline parser in both precision and recall.

Root Distance. Figure 4 (c) shows the precision and recall relative to the distance to the root. Our parser comprehensively and significantly improves baseline parser's recall. But for precision, the baseline parser performs better over long distances (≥ 6) than our parser.

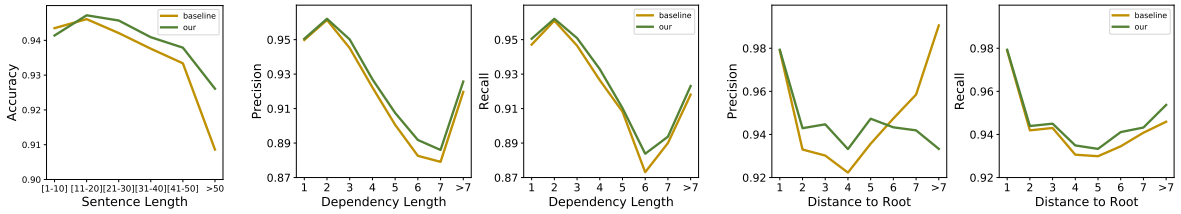
5 Related Work

Graph structures have been extended to model text representation, giving competitive results for a number of NLP tasks. By introducing context neighbors, the graph structure is added to the sequence modeling tool LSTMs, which improves performance on text classification, POS tagging and NER tasks (Zhang et al., 2018a). Based on syntactic dependency trees, DAG LSTMs (Peng et al., 2017) and GCNs (Zhang et al., 2018b) are used to improve the performance of relation extraction task. Based on the AMR semantic graph representation, graph state LSTMs (Song et al., 2018), GCNs (Bastings et al., 2017) and gated GNNs (Beck et al., 2018) are used as encoder to construct graph-to-sequence learning. To our knowledge, we are the first to investigate GNNs for dependency parsing task.

The design of the node representation network is a key problem in neural graph-based parsers. Kiperwasser and Goldberg (2016b) use BiRNNs to obtain node representation with sentence-level information. To better characterize the direction of edge, Dozat and Manning (2017) feed BiRNNs outputs to two MLPs to distinguish word as head or dependent, and then construct a biaffine mapping for prediction. It also performs well on multilingual UD datasets (Che et al., 2018).

Given a graph, a GNN can embed the node by recursively aggregating the node representations of its neighbors (Battaglia et al., 2018). Based on a biaffine mapping, GNNs can enhance the node representation by recursively integrating neighbors' information. The message passing neural network (MPNN) (Gilmer et al., 2017) and the non-local neural network (NLNN) (Wang et al., 2018) are two popular GNN methods. Due to the convenience of self-attention in handling variable sentence length, we use a GAT-like network (Velikovi et al., 2018) belonging to NLNN. Then, we further explore its aggregating functions and update methods on special task.

Apply the GAT to a directed complete graph similar to the Transformer encoder (Vaswani et al., 2017). But the transformer framework focuses only on head-dep-like dependency, we further explore it to capture high-order information on dependency parsing. Several works have investigated high-order features in neural parsing. Kiperwasser and Goldberg (2016b) uses a bottom-up tree-encoding to extract *hard* high-order features from an intermediate predicted tree. Zheng (2017) uses an incremental refinement framework to extract *hard* high-order features from a whole predicted tree. Ma et al. (2018) uses greedy decoding to replace the MST decoding and extract *local* 2-order features at the current decoding time.



(a) (b) (c)

Figure 4: Parsing performance of baseline and our best parser relative to length and graph factors.

Comparing with the previous work, GNNs can efficiently capture *global* and *soft* high-order features.

6 Conclusions

We propose a novel and efficient dependency parser using the Graph Neural Networks. By recursively aggregating the neighbors' information, our parser can obtain node representation that incorporates high-order features to improve performance. Experiments on PTB and UD2.2 datasets show the effectiveness of our proposed method.

Acknowledgement

The authors wish to thank the reviewers for their helpful comments and suggestions and Ziyin Huang, Yufang Liu, Meng Zhang and Qi Zheng for their comments on writing. This research is (partially) supported by STCSM (18ZR1411500). The corresponding authors are Yuanbin Wu and Man Lan.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack LSTM parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2005–2010.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1957–1967.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 273–283.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.

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 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 740–750.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2204–2214.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 334–343.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1263–1272.

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

Eliyahu Kiperwasser and Yoav Goldberg. 2016a. Easy-first dependency parsing with hierarchical tree lstms. *TACL*, 4:445–461.

Eliyahu Kiperwasser and Yoav Goldberg. 2016b. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL*, 4:313–327.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1744–1753.

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, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1403–1414.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515. Association for Computational Linguistics.

Ryan T. McDonald, Koby Crammer, and Fernando C. N. Pereira. 2005. Online large-margin training of dependency parsers. In *ACL 2005, 43rd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, 25-30 June 2005, University of Michigan, USA*, pages 91–98.

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

Ryan T. McDonald and Fernando C. N. Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL 2006, 11st Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, April 3-7, 2006, Trento, Italy*.

Joakim Nivre et al. 2018. Universal Dependencies 2.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, `http://hdl.handle.net/11234/1-1983xxx`.

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.

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017. Cross-sentence n-ary relation extraction with graph lstms. *TACL*, 5:101–115.

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

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1616–1626.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, Portoro, Slovenia. European Language Resources Association.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural*

Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA, pages 6000–6010.

Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*.

Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local neural networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7794–7803.

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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 323–333.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–20, Brussels, Belgium. Association for Computational Linguistics.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. 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. Association for Computational Linguistics.

Yue Zhang, Qi Liu, and Linfeng Song. 2018a. Sentence-state LSTM for text representation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 317–327.

Yuhao Zhang, Peng Qi, and Christopher D. Manning. 2018b. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2205–2215.

Xiaoqing Zheng. 2017. Incremental graph-based neural dependency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1655–1665.

A Hyper-Parameters

Layer	Hyper-parameter	Value
Input	Word	100
	POS tag	100
	Glove	100
LSTM	encoder layers	3
	encoder size	400
MLP	arc MLP size	500
	rel MLP size	100
Dropout	embeddings	0.33
	hidden states	0.33
	inputs states	0.33
	MLP	0.33
Trainer	optimizer	Adam
	learning rate	0.002
	(β_1, β_2)	(0.9, 0.9)
	decay rate	0.75
	decay step length	5000
GNN	graph layers	2

Table 6: Hyper-parameters for experiments.