Semantic Dependency Parsing with Edge GNNs

Songlin Yang, Kewei Tu*
School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
{yangsl,tukw}@shanghaitech.edu.cn

Abstract

Second-order neural parsers have obtained high accuracy in semantic dependency parsing. Inspired by the factor graph representation of second-order parsing, we propose edge graph neural networks (E-GNNs). In an E-GNN, each node corresponds to a dependency edge, and the neighbors are defined in terms of sibling, co-parent, and grandparent relationships. We conduct experiments on SemEval 2015 Task 18 English datasets, showing the superior performance of E-GNNs.

1 Introduction

Traditional syntactic dependency parsing aims to produce a tree structure for a given sentence, which has been well-studied. However, tree-structured representation is ill-suited for producing meaning representation, which motivates the proposal of semantic dependency parsing (SDP) (Oepen et al., 2014). SDP aims to produce a directed acyclic graph (DAG) instead of a tree to enable representing more complex semantic relationships.

Graph-based methods have obtained high accuracy in SDP (Peng et al., 2017; Dozat and Manning, 2018; Wang et al., 2019). Notably, Wang et al. (2019) propose a second-order neural CRF parser and show superior performance compared to the first-order Biaffine Parser (Dozat and Manning, 2018). To optimize the intractable CRF objective, they leverage approximate inference algorithms such as loopy belief propagation (LBP), unrolling several inference steps as recurrent neural networks (Zheng et al., 2015) for end-to-end approximation-aware training (Gormley et al., 2015). However, their model suffers from the following two problems: (i) Second-order dependency parsing can be formulated in terms of factor graphs (Smith and Eisner, 2008). However, the corresponding factor graph for second-order parsing is highly loopy (Fig. 1). It is known that on loopy graphs, LBP can easily get stuck on bad local optima, leading to sub-optimal results and thus undermining the parsing performance. (ii) First-order and second-order scores are produced based solely on contextualized word representations, which is deemed to be sub-optimal (Gan et al., 2022).

In this work, we propose edge GNNs (E-GNNs) to address the aforementioned limitation of Wang et al. (2019). Inspired by factor graph representations of second-order SDP where each variable node corresponds to a dependency edge (Fig. 1), we take edges as GNN nodes and define neighbors in terms of sibling, co-parent, and grandparent relationships. The benefit is shown as follows.

*Corresponding Author

1Our code is publicly available at https://github.com/sustcsonglin/gnn-sdp.
(i) Previous work suggests that GNNs outperform LBP on loopy graphs (Yoon et al., 2018; Satorras and Welling, 2021). Thus we can expect that using E-GNNs will improve the inference quality and thereby result in better parsing performance. (ii) E-GNNs are more expressive since they incorporate edge-level features instead of just word-level features as in Wang et al. (2019). Edge nodes propagate features among neighbors via GNN layers, iteratively refining their representations to be more context-aware, and thereby capturing more information regarding long-range dependencies, which is shown experimentally.

We conduct experiments on SemEval 2015 Task 18 English datasets of SDP, showing superior performance compared with Wang et al. (2019).

2 Model

Word representations. Given a sentence \( w = w_0 \cdots w_n \) (\( w_0 \) is the root), we feed it into BERT (Devlin et al., 2019) to obtain contextualized word embeddings and apply mean-pooling to the last layer of BERT to obtain word-level embedding \( c = c_0 \cdots c_n \). We concatenate \( c \) with POS tag and lemma embeddings

\[
    e_i = c_i \oplus e_i^{\text{pos}} \oplus e_i^{\text{lemma}}
\]

and then feed \( e_0 \cdots e_n \) into a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) (BiLSTM):

\[
    \cdots, (b_i^-, b_i^+), \cdots = \text{BiLSTM}([..., e_i, ...])
\]

The final word representation is \( \overrightarrow{x_i} = b_i^- \oplus b_i^+ \).

Initial edge representation. To obtain initial edge representations, we adopt low-rank bilinear pooling (Kim et al., 2017) in order to capture pairwise interactions of parent and child word representations:

\[
    e_{ij}^0 = U(\sigma(Vx_i) \circ \sigma(Wx_j))
\]

where \( \sigma \) is the activation function and we choose \( \text{tanh} \) in this work; \( \circ \) is Hadamard (element-wise) product; \( U, W, V \) are linear layers (bias terms are omit for brevity).

E-GNN encoding. Inspired by second-order parsing, we take edges as GNN nodes and define neighbors in terms of sibling (sib), co-parent (cop), grand-parent (grdp) and grandchild (grds) relationships. We define \( \text{rel}(i,j) \), the neighbor set\(^2\) of edge \((i, j)\) with respect to relationship \( \text{rel} \in \{\text{sib}, \text{cop}, \text{grdp}, \text{grds}\} \) as follows:

\[
    \begin{align*}
        \text{sib}(i,j) & := \{(i,k)\}_{k} \quad \text{cop}(i,j) := \{(k,j)\}_{k} \\
        \text{grdp}(i,j) & := \{(k,i)\}_{k} \quad \text{grds}(i,j) := \{(j,k)\}_{k}
    \end{align*}
\]

For each \( \text{rel} \), we use a deep biaffine scoring function (Dozat and Manning, 2017) to compute the un-normalized attention scores from edge \((i, j)\) to its neighbor \((m,n) \in \text{rel}(i,j)\):

\[
    \begin{align*}
        e_{\text{rel},a/b}^{ij} & = ML^{\text{rel},a/b}(e_{ij}) \\
        s_{\text{rel}}^{ij,mn} & = [e_{\text{rel},a}^{ij}; 1]^T W^{\text{rel},b} [e_{\text{rel},b}^{mn}; 1];
    \end{align*}
\]

Note that we do not need to compute scores for every pair of \((i, j)\) and \((m, n)\), which needs \( O(n^4) \) time. We only need to compute scores for adjacent edges under specific relationship and thereby need only \( O(n^3) \) time. The normalized attention scores for each relation types are computed as follows:

\[
    \begin{align*}
        \alpha^{\text{sib}}_{ij,k} & = \frac{\exp(s^{\text{sib}}_{ij,k})}{\sum_{k'} \exp(s^{\text{sib}}_{ij,k'})} \\
        \alpha^{\text{cop}}_{ij,k} & = \frac{\exp(s^{\text{cop}}_{ij,k})}{\sum_{k'} \exp(s^{\text{cop}}_{ij,k'})} \\
        \alpha^{\text{grds}}_{ij,k} & = \frac{\exp(s^{\text{grds}}_{ij,k})}{\sum_{k'} \exp(s^{\text{grds}}_{ij,k'})} \\
        \alpha^{\text{grdp}}_{ij,k} & = \frac{\exp(s^{\text{grdp}}_{ij,k})}{\sum_{k'} \exp(s^{\text{grdp}}_{ij,k'})}
    \end{align*}
\]

We compute the feature aggregated from neighbors as:

\[
    t_{ij}^m = \sum_k \left( \alpha^{\text{sib}}_{ij,ik} e_{ij,k}^{m-1} + \alpha^{\text{cop}}_{ij,kj} e_{ij,k}^{m-1} + \alpha^{\text{grds}}_{ij,jk} e_{ij,jk}^{m-1} + \alpha^{\text{grdp}}_{ij,ki} e_{ij,ki}^{m-1} \right)
\]

Next, we update GNN node representations based on their last iteration’s representations and the aggregated feature:

\[
    e_{ij}^m = \text{ReLU}(\text{Linear}(e_{ij}^{m-1}) + t_{ij}^m)
\]

\(^2\)We include the edge itself in its neighbor set because during E-GNN computation, an edge may not want to attend to any neighbors and in that case it can put most of the attention weight on itself.
Table 1: Labeled F1 scores on three formalisms of SemEval 2015 Task 18. +char and +lemma means using character and lemma embeddings. Pointer: Fernández-González and Gómez-Rodríguez (2020). †: our re-implementation.

<table>
<thead>
<tr>
<th>Parser</th>
<th>ID</th>
<th>OOD</th>
<th>ID</th>
<th>OOD</th>
<th>ID</th>
<th>OOD</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dozat and Manning (2017)</td>
<td>93.7</td>
<td>88.9</td>
<td>93.9</td>
<td>90.6</td>
<td>81.0</td>
<td>79.4</td>
<td>89.5</td>
</tr>
<tr>
<td>Kurita and Søgaard (2019)</td>
<td>92.0</td>
<td>87.2</td>
<td>92.8</td>
<td>88.8</td>
<td>79.3</td>
<td>77.7</td>
<td>88.0</td>
</tr>
<tr>
<td>Wang et al. (2019) (MF)</td>
<td>94.0</td>
<td>89.7</td>
<td>94.1</td>
<td>91.3</td>
<td>81.4</td>
<td>79.6</td>
<td>89.8</td>
</tr>
<tr>
<td>Wang et al. (2019) (LBP)</td>
<td>93.9</td>
<td>89.5</td>
<td>94.2</td>
<td>91.3</td>
<td>81.4</td>
<td>79.5</td>
<td>89.8</td>
</tr>
<tr>
<td>Pointer</td>
<td>93.9</td>
<td>89.6</td>
<td>94.2</td>
<td>91.2</td>
<td>81.8</td>
<td>79.8</td>
<td>90.0</td>
</tr>
<tr>
<td>Zhang et al. (2019)</td>
<td>92.2</td>
<td>87.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Lindemann et al. (2019)</td>
<td>94.1</td>
<td>90.5</td>
<td>94.7</td>
<td>92.8</td>
<td>82.1</td>
<td>81.6</td>
<td>90.3</td>
</tr>
<tr>
<td>Lindemann et al. (2020)</td>
<td>93.9</td>
<td>90.4</td>
<td>94.7</td>
<td>92.7</td>
<td>81.9</td>
<td>81.6</td>
<td>90.2</td>
</tr>
<tr>
<td>Pointer</td>
<td>94.4</td>
<td>91.0</td>
<td>95.1</td>
<td>93.4</td>
<td>82.6</td>
<td>82.0</td>
<td>90.7</td>
</tr>
<tr>
<td>LBP† (baseline)</td>
<td>94.9</td>
<td>91.7</td>
<td>95.2</td>
<td>93.5</td>
<td>82.6</td>
<td>82.3</td>
<td>90.9</td>
</tr>
<tr>
<td>E-GNN (ours)</td>
<td>95.0</td>
<td>92.0</td>
<td>95.5</td>
<td>93.9</td>
<td>82.9</td>
<td>82.4</td>
<td>91.1</td>
</tr>
</tbody>
</table>

**Training loss.** After \( l \) iterations of GNN update, we obtain \( e_{ij}^{l} \) for each edge. We use an MLP to map \( e_{ij}^{l} \) into a \( q \)-dimensional vector \( d_{ij} \), where \( q \) is the label set size (including the special NULL label). We can associate each edge with a label index, which is either the index of NULL if the edge does not exist in the gold SDP graph, or the index of the gold edge label. We denote this label index as \( y_{ij}^{l} \). Then we use cross-entropy to compute the loss:

\[
L = - \sum_{ij} \log \frac{\exp\{d_{ij}(y_{ij}^{l})\}}{\sum_{y} \exp\{d_{ij}(y)\}}
\]

### 3 Experiment

#### 3.1 Setup

We conduct experiments on the SemEval 2015 Task 18 English datasets (Oepen et al., 2015). Sentences in the datasets are annotated with three formalisms: DM (Flickinger et al., 2012), PAS (Miyao and Tsujii, 2004), and PSD (Hajič et al., 2012). We use the standard data splitting as used in previous works (Martins and Almeida, 2014; Du et al., 2015), which contains 33,964 sentences in the training set, 1,692 sentences in the development set, 1,410 sentences in the in-domain (ID) test set and 1,849 sentences in the out-of-domain (OOD) test set from the Brown Corpus (Francis and Kucera, 1982). We use bert-base-cased (Devlin et al., 2019) to obtain contextualized word embedding. The number of GNN layers is set to 3. Other hyperparameters can be found in App. A. We report the labeled F1 scores (LF1) in the ID and OOD test sets for each formalism. The reported results are averaged over three runs with different random seeds.

#### 3.2 Main result

Table 1 shows the experimental results. We reimplement the LBP-based second-order parser of Wang et al. (2019) as the baseline (LBP hereafter for short), using the same neural encoder and the same settings (e.g., hyper-parameters) as E-GNN for fair comparison. As we can see, LBP has already surpassed Pointer (Fernández-González and Gómez-Rodríguez, 2020), a strong model, by 0.2 and 0.4 average F1 scores in ID test sets and OOD test sets. E-GNN further outperforms LBP by 0.2 average F1 scores on both ID and OOD test datasets.

#### 3.3 Ablation study

We conduct ablation studies on PAS. First, we study the importance of using different relationship types to define neighbors in GNNs. As we can see from Table 2, removing sib/cop/grd (both grdp and gdrd) results in 0.17, 0.20, 0.18 LF1 score drops, respectively, showing that all these relationships are beneficial to the final performance, which is consistent with the intuition in second-order parsing. Second, we conduct an ablation study on the effect of the number of GNN layers. Table 2 shows that using 0/1/2 layers leads to 0.32/0.18/0.15 F1 score drops, respectively, showing that all these relationships are beneficial to the final performance, which is consistent with the intuition in second-order parsing.

#### 3.4 Error analysis

Fig. 2 shows the change of LF1 scores with the length of dependency edges. We can see that when...
the edge length is small (1-5), LBP and E-GNN have almost identical performance. However, when the edge length is large (>10), E-GNN outperforms LBP by a large margin, especially when the edge length is larger than 20. We hypothesize that E-GNN can model long-range dependencies more effectively. Neural encoders such as BiLSTMs have difficulty in capturing long-range dependencies, so relying solely on word representations to produce first/second-order edge scores, as in Wang et al. (2019), would have difficulty in predicting long edges. In comparison, for E-GNN, although the initial edge representation may also have difficulty in capturing long-range dependencies, during iterative GNN update, a long edge can gather information from all its neighbors, refining its representation to be more context-aware and thus capturing more long-range information.

4 Related work

Dependency parsing with GNNs. Ji et al. (2019) used GNNs for dependency parsing. However, they view words instead of edges as GNN nodes. Consequently, it is tricky to define neighbors and thus tricky to design node vector update schemes. In our model, we view edges as GNN nodes, so we can define neighbors and design node vector update schemes more naturally by following second-order dependency relationships. In addition, our model captures edge-level features and thus is more expressive.

Algorithmic alignment. One can view the GNN layers of our model as a learnable inference decoder, which mimics the behavior of LBP. Xu et al. (2020) propose the concept of algorithmic alignment, finding that neural networks—whose structures resembling classical algorithms for certain problems—are easier to train and have better performance. The design of our model follows the principle of algorithmic alignment, as E-GNN nodes resemble variable nodes in the factor graph of second-order parsing, and the message passing mechanism of the GNN resembles LBP inference steps. We can find other successful models complying with the algorithmic alignment principle in the field of NLP. Taking DIORA (Drozdov et al., 2019) for example, it mimics the classical inside-outside
algorithm to design the network and achieves good performance in unsupervised constituency parsing.

5 Conclusion

We proposed E-GNNs in the spirit of the factor graph representation of second-order dependency parsing. Experiments and ablation studies on SemEval 2015 Task 18 English datasets of SDP validated the effectiveness of E-GNNs.

Limitations

E-GNN needs $O(n^3)$ time to update edge representations in each GNN layer, while the Biaffine Parser only needs $O(n^2)$ time to score all edges. Besides, E-GNN needs to store $O(n^2)$ edge embeddings in each GNN layer, consuming more GPU memories than the Biaffine Parser.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (61976139).

References


### Architecture hyper-parameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS/Lemma dimension</td>
<td>100</td>
</tr>
<tr>
<td>Embeddings dropout</td>
<td>0.33</td>
</tr>
<tr>
<td>BiLSTM encoder size</td>
<td>1000</td>
</tr>
<tr>
<td>BiLSTM layers dropout</td>
<td>0.33</td>
</tr>
<tr>
<td>MLP layers dropout</td>
<td>0.33</td>
</tr>
<tr>
<td>BiAffine hidden size</td>
<td>300</td>
</tr>
<tr>
<td>GNN hidden size</td>
<td>500</td>
</tr>
<tr>
<td>GNN layer</td>
<td>3</td>
</tr>
</tbody>
</table>

### Training-related hyper-parameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT learning rate</td>
<td>$5 \times 10^{-5}$</td>
</tr>
<tr>
<td>Other learning rate</td>
<td>$1.5 \times 10^{-3}$</td>
</tr>
<tr>
<td>Optimizer</td>
<td>AdamW</td>
</tr>
<tr>
<td>Scheduler</td>
<td>linear warmup</td>
</tr>
<tr>
<td>Warmup rate</td>
<td>0.5</td>
</tr>
<tr>
<td>Gradient clipping</td>
<td>5.0</td>
</tr>
<tr>
<td>Tokens per batch</td>
<td>3000</td>
</tr>
<tr>
<td>Maximum training sentence length</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 3: Summary of hyper-parameters.

## A Hyperparameter details

The hyperparameter configuration is summarized in Table 3. Besides, the number of total training epoch is set to 30 for DM; 20 for PAS and PSD. The number of BiLSTM encoder layer is 1 for DM, and 2 for PAS and PSD.