Supertagging Combinatory Categorial Grammar
with Attentive Graph Convolutional Networks

Yuanhe Tian♥, Yan Song♠♥†, Fei Xia♥
♥University of Washington  ♠The Chinese University of Hong Kong (Shenzhen)
♦Shenzhen Research Institute of Big Data
♥{yhtian, fxia}@uw.edu ♠songyan@cuhk.edu.cn

Abstract

Supertagging is conventionally regarded as an important task for combinatory categorial grammar (CCG) parsing, where effective modeling of contextual information is highly important to this task. However, existing studies have made limited efforts to leverage contextual features except for applying powerful encoders (e.g., bi-LSTM). In this paper, we propose attentive graph convolutional networks to enhance neural CCG supertagging through a novel solution of leveraging contextual information. Specifically, we build the graph from chunks (n-grams) extracted from a lexicon and apply attention over the graph, so that different word pairs from the contexts within and across chunks are weighted in the model and facilitate the supertagging accordingly. The experiments performed on the CCGbank demonstrate that our approach outperforms all previous studies in terms of both supertagging and parsing. Further analyses illustrate the effectiveness of each component in our approach to discriminatively learn from word pairs to enhance CCG supertagging.

1 Introduction

Combinatory categorial grammar (CCG) is a lexicalized grammatical formalism, where the lexical categories (also known as supertags) of the words in a sentence provide informative syntactic and semantic knowledge for text understanding. Therefore, CCG parse often provides useful information for many downstream natural language processing (NLP) tasks such as logical reasoning (Yoshikawa et al., 2018) and semantic parsing (Beschke, 2019). To perform CCG parsing in different languages, most studies conducted a supertagging-parsing pipeline (Clark and Curran, 2007; Kummerfeld et al., 2010; Song et al., 2012; Lewis and Steedman, 2014b; Huang and Song, 2015; Xu et al., 2015; Lewis et al., 2016; Vaswani et al., 2016; Yoshikawa et al., 2017), in which their main focus is the first step, and they generated the CCG parse trees directly from supertags with a few rules afterwards.

Building an accurate supertagger in a sequence labeling process requires a good modeling of contextual information. Recent neural approaches to supertagging mainly focused on leveraging powerful encoders with recurrent models (Lewis et al., 2016; Vaswani et al., 2016; Clark et al., 2018), with limited attention paid to modeling extra contextual features such as word pairs with strong relations. Graph convolutional networks (GCN) is demonstrated to be an effective approach to model such contextual information between words in many NLP tasks (Marcheggiani and Titov, 2017; Huang and Carley, 2019; De Cao et al., 2019; Huang et al., 2019); thus we want to determine whether this approach can also help CCG supertagging.

However, we cannot directly apply conventional GCN models to CCG supertagging because in most of the previous studies the GCN models are built over the edges in the dependency tree of an input sentence. As high-quality dependency parsers are not always available, we do not want our CCG supertaggers to rely on the existence of dependency parsers. Thus, we need another way to extract useful word pairs to build GCN models. For that, we propose to obtain word pairs from frequent chunks (n-grams) in the corpus, because those chunks are easy to identify with co-occurrence counts. To appropriately learn from n-grams, one requires the GCN to be able to distinguish different word pairs because such information in n-grams are not explicitly structured as that in dependency parses. Because existing GCN models are limited in treating all word pairs equally, we propose an adaptation of conventional GCN for CCG supertagging.

1Corresponding author.
2Our code and models for CCG supertagging are released at https://github.com/cuhksz-nlp/NeST-CCG.
We treat CCG supertagging as a sequence labeling within and across chunks and an attention mechanism is applied to GCN to weight those edges. Since CCG supertagging is also a parsing task, we propose to construct the graph based on the chunks (n-grams) extracted from a pre-constructed n-gram lexicon $\mathcal{N}$. Specifically, the lexicon consists of n-grams automatically extracted from raw corpora by unsupervised methods. The overall architecture of our tagger is illustrated in Figure 1, with an input sentence and corresponding supertagging and parsing output. The details of the main components in the architecture are provided below.

2 The Approach

We treat CCG supertagging as a sequence labeling task, where the input is a sentence with $n$ words $\mathcal{X} = x_1 x_2 \cdots x_i \cdots x_n$, and the output is a sequence of supertags $\mathcal{Y} = \hat{y}_1 \hat{y}_2 \cdots \hat{y}_i \cdots \hat{y}_n$. Our approach uses attentive GCN (A-GCN) to incorporate information of word pairs through a graph; the graph is built based on n-grams in the input sentence that appear in a lexicon $\mathcal{N}$. This lexicon is built based on chunks (n-grams) extracted from the lexicon $\mathcal{N}$, with the chunks illustrated in the red boxes.

In this paper, we propose attentive GCN (A-GCN) for CCG supertagging, where its input graph is built based on chunks (n-grams) extracted with unsupervised methods. In detail, two types of edges in the graph are introduced to model word relations within and across chunks and an attention mechanism is applied to GCN to weight those edges. In doing so, different contextual information are discriminatively learned to facilitate CCG supertagging without requiring any external resources. The validity of our approach is demonstrated by experimental results on the CCGBank (Hockenmaier and Steedman, 2007), where state-of-the-art performance is obtained for both tagging and parsing.

2.1 GCN

Normal GCN models with $L$ layers learn from word pairs suggested by the dependency parsing results of the input sentence $\mathcal{X}$, where the edges between all pairs of words $x_i$ and $x_j$ are represented by an adjacency matrix $A = \{a_{i,j}\}_{n \times n}$. In $A$, $a_{i,j} = 1$ if there is a dependency edge between $x_i$ and $x_j$ or $i = j$ (the direction of the edge is ignored), and $a_{i,j} = 0$ otherwise. Based on the adjacency matrix, for each $x_i$, the $l$-th GCN layer finds all $x_j$ associated with $x_i$ (where $a_{i,j} = 1$), takes their hidden vectors $h_j^{(l-1)}$ from the $(l-1)$-th layer, and computes the output for $x_i$ by

$$ h_i^{(l)} = \sigma(LN(\sum_{j=1}^{n} a_{i,j} (W^{(l)} \cdot h_j^{(l-1)} + b^{(l)}))) $$

where $W^{(l)}$ and $b^{(l)}$ are trainable matrix and bias for the $l$-th GCN layer, $LN$ refers to layer normalization and $\sigma$ the ReLU activation function. Therefore, in normal GCN, for each $x_i$, all the $x_j$ that connect to $x_i$ are treated exactly the same.

2.2 Graph Construction based on Chunks

Since CCG supertagging is also a parsing task, we do not want our approach to rely on the existence of a dependency parser. Without such a parser, we need an alternative for finding good word pairs to build the graph in A-GCN (which is equivalent to build the adjacency matrix $A$). Inspired by the studies that leverage chunks (n-grams) as effective features to carry contextual information and enhance model performance (Song et al., 2009; Song and Xia, 2012; Ishiwatari et al., 2017; Yoon et al., 2018; Zhang et al., 2019; Tian et al., 2020a,c,b), we propose to construct the graph based on the chunks (n-grams) extracted from a pre-constructed n-gram lexicon $\mathcal{N}$. Specifically, the lexicon is constructed by computing the PMI of any two adjacent words $s'$, $s''$ in the training set by

$$ PMI(s', s'') = \log \frac{p(s's'')}{p(s')p(s'')} $$

where $p$ is the probability of an n-gram (i.e., $s'$, $s''$ and $s's''$) in the training set; then a high PMI

Figure 1: The architecture of our CCG supertagger with A-GCN and an example input sentence with its corresponding supertagging and parsing output. The overall architecture of our tagger is illustrated in Figure 1, with an input sentence and corresponding supertagging and parsing output. The details of the main components in the architecture are provided below.
Figure 2: Examples of the two types of edges for building the graph in an input sentence, in which chunks (n-grams) extracted from the lexicon \( \mathcal{N} \) are highlighted in green; example in-chunk and cross-chunk edges are marked in blue and red color, respectively.

The score suggests that the two words co-occur a lot in the dataset and are more likely to form a n-gram. For each pair of adjacent words \( s_{i-1}, s_i \) in a sentence \( \mathcal{S} = s_1 s_2 \cdots s_{i-1} s_i \cdots s_n \), we compute the PMI score of the two words and use a threshold to determine whether a delimiter should be inserted between them. As a result, the sentence \( \mathcal{S} \) is segmented into pieces of n-grams and we extract all n-grams from all sentences to form the lexicon \( \mathcal{N} \).

Then for graph building, given an input sentence \( \mathcal{X} \), we find all the n-grams in \( \mathcal{X} \) that appear in \( \mathcal{N} \). A chunk is either a n-gram that does not overlap with other n-grams or a text span that covers multiple overlapping n-grams. For example, in Figure 2, we find four chunks (i.e., “all students”, “are required to”, “finish”, and “in two hours”) in the example sentence according to the lexicon \( \mathcal{N} \) (the chunks are highlighted in green). In these chunks, “all students”, “finish”, and “in two hours” are non-overlapping n-grams included in the lexicon and “are required to” is a text span that covers the overlapping n-grams “are required” and “required to”.

In most cases, the adjacent words within the same chunk tend to have a strong word-word relation in terms of co-occurrence, and thus we can build the graph and its adjacency matrix accordingly.

Based on the chunks, we construct the graph by two types of edges, i.e., the in-chunk and cross-chunk ones: the first type is to model local word pairs, and the graph includes edges between any two adjacent words within the same chunk. For example, as shown in Figure 2, the in-chunk edges (blue lines) for the chunk “in two hours” are “(in, two)” and “(two hours)”. The second type is to model cross chunk word pairs, which are built from any two adjacent chunks with the starting and ending words in the two chunks connected. The motivation of using the starting and ending words is that English phrases tend to be head-initial (e.g., verb phrase such as “buy some books”) or head-final (e.g., adjective phrase such as “red apples”) in many cases. E.g., for the two chunks “all students” and “are required to” in Figure 2, the corresponding cross-chunk edges (red lines) are “(all, are)”, “(all, to)”, “(students, are)”, and “(students, to)”. The graph is equivalent to the adjacency matrix \( A \), where \( a_{i,j} = 1 \) if there is an edge between \( x_i \) and \( x_j \) in the graph or \( i = j \), and \( a_{i,j} = 0 \) otherwise.

### 2.3 The Attentive GCN

When learning from a graph, conventional GCN models treat all word pairs from the graph equally, and thus are unable to account for the possibility that the contribution of different \( x_j \) on \( x_i \) may vary. Particularly for our graph built from chunks, it is important to be able to distinguish different word pairs because all the chunks and the graph are constructed automatically without a dependency parser. Therefore, we apply an attention mechanism to the adjacency matrix and adapt Eq. (1) used in the normal GCN for our A-GCN by replacing the \( a_{i,j} \in \{0, 1\} \) by a weight \( p_{i,j}^{(l)} \) for each \( x_i \) and all its associated \( x_j \), the weight \( p_{i,j}^{(l)} \) for this word pair is computed by

\[
p_{i,j}^{(l)} = \frac{a_{i,j} \cdot \exp(h_i^{(l-1)} \cdot W_{pos}^{(l)} \cdot h_j^{(l-1)})}{\sum_{j=1}^{n} a_{i,j} \cdot \exp(h_i^{(l-1)} \cdot W_{pos}^{(l)} \cdot h_j^{(l-1)} )}
\]

where \( W_{pos}^{(l)} \) models the positional relation (i.e., left, right, or self) between \( x_i \) and \( x_j \) and it has three choices, i.e., \( W_{left}^{(l)}, W_{right}^{(l)} \), and \( W_{self}^{(l)} \) for different \( i \) and \( j \).

### 2.4 Supertagging with A-GCN

To conduct supertagging with A-GCN, we firstly obtain the hidden vector \( h_i^{(l)} \) for \( x_i \) from BERT (Devlin et al., 2019) to feed into the first GCN layer. Upon receiving the encoding results from A-GCN, the following supertagging process is straightforward: each \( h_i^{(L)} \) is obtained from the last A-GCN layer and aligned with the output by

\[
o_i = W_d \cdot h_i^{(L)}
\]

where \( W_d \) is a trainable matrix

---

1For example, a sentence can be segmented into \( \mathcal{S} = s_1/s_2 s_3 \cdots s_4/s_5 \) ("/" refers to a delimiter) if the PMI of \( s_1, s_2 \) and \( s_4, s_5 \) are lower than the threshold and the PMI of \( s_2, s_3 \) and \( s_3, s_4 \) are greater than the threshold; we thus obtain three n-grams, i.e., \( s_1, s_2 s_3 s_4, \) and \( s_5 \) from this sentence.

2We do not distinguish the two types of edges in \( A \).

3For example, \( W_{pos}^{(l)} = W_{left}^{(l)}, \) if \( j < i \).
### Table 1: The train/dev/test splits of English CCGBank and the statistics of sentences and words in them.

<table>
<thead>
<tr>
<th>Section No.</th>
<th>Train</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-21</td>
<td>23</td>
<td>0</td>
<td>929,552</td>
</tr>
<tr>
<td>Sentence #</td>
<td>39,604</td>
<td>1,913</td>
<td>2,407</td>
</tr>
<tr>
<td>Word #</td>
<td>929,552</td>
<td>45,422</td>
<td>55,371</td>
</tr>
</tbody>
</table>

### Hyper-parameters

<table>
<thead>
<tr>
<th>Hyper-parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Size</td>
<td>16, 32</td>
</tr>
<tr>
<td>Drop-out Rate</td>
<td>0.2</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>3e-5, 2e-5, 1e-5, 5e-6</td>
</tr>
<tr>
<td>Max Sentence Length</td>
<td>300</td>
</tr>
<tr>
<td>Random Seed</td>
<td>42</td>
</tr>
<tr>
<td>Training Epoch</td>
<td>50</td>
</tr>
<tr>
<td>Warm-up Rate</td>
<td>0.1, 0.2</td>
</tr>
</tbody>
</table>

### Table 2: The list of hyper-parameters tested in our experiments. We run all models with the combination of those hyper-parameters and use the one achieving the highest supertagging results in our final experiments.

### Table 3: Results (supertagging accuracy and labeled F-scores) of different models with BERT-Large encoder on the development set of CCGBank. “PARM” is the number of trainable parameters in the models; “Full” uses the fully connected graph and “Chunk” uses the graph built based on chunks.

<table>
<thead>
<tr>
<th>Models</th>
<th>PARM</th>
<th>TAG</th>
<th>LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT-Cased</td>
<td>335M</td>
<td>96.04</td>
<td>90.31</td>
</tr>
<tr>
<td>+ A-GCN (Full)</td>
<td>343M</td>
<td>95.93</td>
<td>90.13</td>
</tr>
<tr>
<td>+ A-GCN (Chunk)</td>
<td>343M</td>
<td>96.11</td>
<td>90.49</td>
</tr>
<tr>
<td>BERT-Uncased</td>
<td>337M</td>
<td>96.06</td>
<td>90.37</td>
</tr>
<tr>
<td>+ A-GCN (Full)</td>
<td>345M</td>
<td>95.99</td>
<td>90.21</td>
</tr>
<tr>
<td>+ A-GCN (Chunk)</td>
<td>345M</td>
<td>96.17</td>
<td>90.60</td>
</tr>
</tbody>
</table>

### 3 Experiments

#### 3.1 Settings

We run experiments on the English CCGBank (Hockenmaier and Steedman, 2007)\(^5\) and follow Clark and Curran (2007) to split it into train/dev/test sets, whose statistics (sentence and word numbers) are reported in Table 1. To construct n-gram lexicon \(N\) for building the edges in our graph, we perform PMI on the training set of CCGBank to extract n-grams whose length is between \([1, 5]\), with the threshold of the PMI score set to 0. For the encoder, we try both cased and uncased BERT-Large (Devlin et al., 2019) with their default settings (e.g., 24 layers of self-attentions in 1024 dimensional hidden vectors)\(^6\) and used two layers for A-GCN. To obtain CCG parse from the generated supertags, we adopt the parsing algorithm used in EasyCCG (Lewis and Steedman, 2014a). We follow previous studies (Lewis and Steedman, 2014a; Lewis et al., 2016; Yoshikawa et al., 2017) to use the most frequent 425 supertags as the tag set and evaluate our model on both the tagging accuracy and the labeled F-scores (LF) of the dependencies converted from CCG parse\(^7\).

For other hyper-parameter settings, we test their values as shown in Table 2 when training our models. We tried all combinations of them for each model and use the one achieving the highest supertagging results in our final experiments. Note that, with the best hyper-parameters, the best performance is achieved with warm-up rate 0.1, batch size 16, and learning rate 1e-5.

#### 3.2 Results

To explore the effectiveness of our approach, we run CCG taggers with and without A-GCN, and try two ways to construct the graph: one is a fully connected GCN where edges are built between every two words; the other is our proposed approach with the chunk-based graph. Experimental results on supertagging accuracy (TAG) and labeled F-scores (LF) for parsing on the development set of CCGBank are reported in Table 3, with the number of trainable parameters of all models also presented.

The experiments show that, for both cased and uncased BERT encoders, the proposed chunk A-GCN works the best in terms of both supertagging accuracy and parsing results. In contrast, Full A-GCN has inferior performance to the BERT baselines. This contrast shows the importance of appropriate construction of the graphs fed into A-GCN, since the fully connected graph with all words associated...
Table 4: Comparison of our models with uncased BERT encoder and previous studies on the test set of CCGbank. Models with “†” use the EasyCCG parser to generate CCG parse trees from the predicted supertags.

Table 5: Experimental results of models with uncased BERT-Large encoder on the test set of CCGbank, where the in-chunk, cross-chunk edges or the attention mechanism in our A-GCN module is ablated.

### 3.3 Ablation Study

We conduct an ablation study to explore the effect of the two types of edges and the attention mechanism on our best model. The supertagging and parsing results of models with different configurations are reported in Table 5, where the results are categorized into four groups. The first group (ID 1) is the results of the best performing model where all settings are activated; the second (ID 2-3) is the ablation of either in-chunk or cross-chunk edges with attention; the third (ID 4-6) is the result of using normal GCN without the attention mechanism; and the last group (ID 7) is the baseline model where none of the three settings is activated.

The results show that the model performance drops when either part is ablated (ID 1 vs. ID 2-6). Specifically, removing attention significantly hurts the performance, where all configurations without attention (ID 4-6) shows worse-than-baseline (ID 7) results; this observation confirms the importance of applying attention on GCN. One possible explanation to this phenomenon could be that considerable noises are introduced to the graph because the edges in our graph are derived from chunks and they do not follow syntax in most cases; thus, it is crucial to assign weights to the edges and not treat them equally. Interestingly, comparing the two types of edges, models with cross-chunk edges yield much higher results than the ones with in-chunk edges only when the attention is not used (ID 5 vs. ID 6), while it is slightly better when attention is applied (ID 2 vs. ID 3). This comparison suggests that in-chunk edges could introduce more noise than cross-chunk edges. So that when the attention is not used (ID 6), the model fails to weight the edges and results in a significant drop on its performance; On the contrary, when the attention is applied (ID 3), our model is able to even the performance of models with in-chunk and cross-chunk edges, which confirms that weighting is essential in selecting useful information for CCG supertagging.

### 4 Conclusion

In this paper, we propose A-GCN for CCG supertagging, with its graph built from chunks extracted from a lexicon. We use two types of edges for the graph, namely, in-chunk and cross-chunk edges for word pairs within and across chunks, respectively, and propose an attention mechanism to distinguish the important word pairs according to their contribution to CCG supertagging. Experimental results and the ablation study on the English CCGbank demonstrate the effectiveness of our approach to CCG supertagging, where state-of-the-art performance is obtained on both CCG supertagging and parsing. Further analysis is performed to investigate using different types of edges, which reveals their quality and confirms the necessity of introducing attention to GCN for CCG supertagging.
Acknowledgements

This work is supported by The Chinese University of Hong Kong (Shenzhen) under University Development Fund UDF01001809.

References


Appendix A: Example Sentences with Extracted Chunks

Figure 3: Example sentences with the chunks extracted from the lexicon $\mathcal{N}$ highlighted in green.

In the main experiments, we use the lexicon obtained from the training set of the English CCG-bank to extract chunks in each sentence, where the chunks are used to build the graph. Figure 3 shows five example sentences in which the extracted chunks are highlighted in green. We report more examples in the supplemental materials.

Appendix B: Example Suppertagging and Parsing Results

Figure 4 shows the CCG supertagging and parsing results of EasyCCG and our approach (i.e., BERT + A-GCN (Chunk)) on two example sentences. In the figure, the correct and incorrect supertags are represented by green and red color, respectively. We report more CCG parsing results of our approach in the supplemental materials.

We use the implementation from CCGweb (Evang et al., 2019) at https://ccgweb.phil.hhu.de/.
EasyCCG

The Dow Jones industrials closed at 2569.26

Our Approach

The Dow Jones industrials closed at 2569.26

EasyCCG

The Dow Jones industrials closed at 2569.26

Our Approach

The Dow Jones industrials closed at 2569.26

Figure 4: The CCG supertagging and parsing results of EasyCCG and our approach (i.e., BERT + A-GCN (Chunk)) on the examples, where the correct and incorrect supertags are represented in green and red color, respectively.