Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representation

Eliyahu Kiperwasser & Yoav Goldberg 2016
Presented by: Yaoyang Zhang
Outline

Simple and Accurate **Dependency Parsing** Using **Bidirectional LSTM Feature Representation**

- Background – **Bidirectional RNN**
- Background – **Dependency Parsing**
- Motivation – **Bidirectional RNN as feature functions**
- Model for transition-based parser
- Model for graph-based parser
- Results and conclusion
Bidirectional Recurrent Neural Network

- RNN has memory of the past up to time $i$, at step $i$
- What if we also have memory of the “future”? Since we are talking about text, the preceding and succeeding context should both carry some weight
- Use two RNNs, with different directions
- Each direction has its own set of parameters
- Use LSTM cells

[1]Figures borrowed from Stanford CS 244d notes
Bidirectional Recurrent Neural Network

- Why and how to use BiRNN for dependency parsing
  - Motivation: get a vector representation for each word in a sentence, which will later be used as feature input for parsing algorithm
  - One BiRNN per sentence
  - Will be trained jointly with a classifier/regressor depending on the parsing model

The brown fox jumped over the lazy dog
Bidirectional Recurrent Neural Network

• Input: words $w_1, w_2, ..., w_n$, POS tag $t_1, t_2, ..., t_n$
• Input to BiLSTM: $x_i = e(w_i) | e(t_i)$
  • $e()$: embedding of word/tag, jointly trained with the network
  • $|$: concatenation
• Output from BiLSTM: $v_i = \text{BiRNN}_\theta(x_{1:n}, i) = R_{NN_F}(x_{1:i}) \circ R_{NN_R}(x_{n:i})$
  • Feature representation
  • Output $\text{BiRNN}_\theta(x_{1:n}, i)$ is the concatenation of the outputs from two directions
Dependency Grammar

• A grammar model
• The syntactic structure of a sentence is described solely in terms of the words in a sentence and an associated set of directed binary grammatical relations that hold among the words\(^1\)
• TL; DR: Dependency grammar assumes that syntactic structure consists only of dependencies\(^2\)

---

[1] Speech and Language Processing, Chapter 14
[2] CS447 slide
Dependency Grammar

• There are other grammar models out there such as context-free grammar, but we are focusing on dependency grammar here
• Dependency parsing the process of getting the parse tree out of a sentence
• Dependency structures:
  • Each dependency is a directed edge from one word to another
  • Dependencies form a connected and acyclic graph over the words in a sentence
  • Every node (word) has at most one incoming edge
  • ➞ It is a rooted tree
• Universal dependencies: 37 syntactic relations for any language (with modification)
Parsing Algorithms

• Transition-based v.s. Graph-based

• Transition-based:
  • start with an initial state (empty stack, all words in a queue/buffer, empty dependencies)
  • greedily choose an action (shift, left-arc, right-arc) based on the current state
  • Repeat until reaching a terminal state (empty stack, empty queue, parse tree)

• Graph-based:
  • All the possible edges are associated with some scores
  • Different parse trees have different total scores
  • Use an (usually dynamic programming) algorithm to find the tree with the highest score
Transition-based Dependency Parsing

- **s:** sentence
- **w:** word
- **t:** transition (action)
- **c:** configuration (state)

**Initial:** empty stack, all words in the queue, empty dependencies

**Terminal:** empty stack, empty queue, dependency tree

**Legal:** shift, reduce, left-arc(label), right-arc(label)

**Scorer**($\phi(c), t$): given feature $\phi(c)$, outputs score for action $t$

---

**Algorithm 1** Greedy transition-based parsing

1. **Input:** sentence $s = w_1, \ldots, x_w, t_1, \ldots, t_n$, parameterized function $\text{SCORE}_\theta(\cdot)$ with parameters $\theta$.
2. $c \leftarrow \text{INITIAL}(s)$
3. **while not** $\text{TERMINAL}(c)$ **do**
4. $\hat{t} \leftarrow \arg \max_{t \in \text{LEGAL}(c)} \text{SCORE}_\theta(\phi(c), t)$
5. $c \leftarrow \hat{t}(c)$
6. **return** $\text{tree}(c)$
Transition-based Dependency Parsing

Economic news had little effect on financial markets.

[1] Borrowed from CS 447 slides
Transition-based Dependency Parsing - Motivation

• How to get feature $\phi(c)$, given the current state $c$?
  • Old-school: “Hand-crafted” features (templates) – can have as many as 72 templates
  • Now: Deep learning (Bidirectional LSTM)
    • $\phi(c)$ is actually a simple function of the BiRNN output vectors!

• Once we get the feature $\phi(c)$, the rest is straightforward
  • Train a classifier based on $\phi(c)$ and output $t$
Transition-based Dependency Parsing

- Output from BiLSTM: $v_i$
  - Feature representation

- Input to classifier (Multi-layer perceptron, MLP): $\phi(c)$
  - $c$: state at time $i$, $(\ldots|s_2|s_1|s_0, b_0|\ldots, T)$
  - $\phi(c) = v_{s_2} \circ v_{s_1} \circ v_{s_0} \circ v_{b_0}$

- Output from MLP: a vector of scores for all possible actions

- Objective (max-margin): Maximize the difference between the score of the correct action and the maximum score of all incorrect actions
  - $G$: correct (gold) actions
  - $A$: all actions

$$MLP_\theta(x) = W^2 \cdot \tanh(W^1 \cdot x + b^1) + b^2$$
Transition-based Dependency Parsing

• Put everything together:

Configuration:

```
<table>
<thead>
<tr>
<th>s2</th>
<th>s1</th>
<th>s0</th>
<th>b0</th>
<th>b1</th>
<th>b2</th>
<th>b3</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>jumped</td>
<td>over</td>
<td>the</td>
<td>lazy</td>
<td>dog</td>
<td>ROOT</td>
</tr>
</tbody>
</table>
```

Scoring:

```
(ScoreLeftArc, ScoreRightArc, ScoreShift)
```
Transition-based Dependency Parsing

• Other things to note:
  • Error exploration and dynamic oracle: a technique to explore wrong configurations to reduce overfitting, needs to redefine G (called dynamic oracle)
  • Aggressive exploration: with some (small) probability to follow the wrong configuration if the difference of scores between the correct and incorrect actions are small enough. Further reduces overfitting
Graph-based Dependency Parsing

- **Input:** sentence $s$, chooses a tree $y$ that score the highest (general form)
  - Score of a tree $y$ is the summation of scores of all its subtrees

\[
\text{predict}(s) = \arg\max_{y \in \mathcal{Y}(s)} \text{score}_{\text{global}}(s, y)
\]

\[
\text{score}_{\text{global}}(s, y) = \sum_{\text{part} \in y} \text{score}_{\text{local}}(s, \text{part})
\]
Graph-based Dependency Parsing

• Arc-factored graph: relaxes the assumption. Decompose the score of a tree into the sum of scores of arcs.
  • $\phi(s, h, m)$: feature function of edge $(h, m)$ in the sentence $s$

$$\text{parse}(s) = \arg \max_{y \in \mathcal{Y}(s)} \sum_{(h, m) \in y} \text{score}(\phi(s, h, m))$$

• Efficient DP algorithm to find the parse tree if $\phi(s, h, m)$ is given (Eisner’s decoding algorithm)

• Again, how to get the feature function $\phi(s, h, m)$?
  • Of course use vector representation from BiRNN. Concatenation of the two vectors for $h$ and $m$
Graph-based Dependency Parsing

Figure 14.12  Initial rooted, directed graph for *Book that flight.*

[1]Speech and Language Processing, Chapter 14
The Model (for Graph-Based)

• Output from BiLSTM: $v_i = \text{BiRNN}(x_{1:n}, i)$
  • Feature representation

• Input to regressor (Multi-layer perceptron, MLP): $\phi(s, h, m)$

$$
\phi(s, h, m) = \text{BiRNN}(x_{1:n}, h) \circ \text{BiRNN}(x_{1:n}, m)
$$

• Output from MLP: score for this edge

• Objective (max-margin, similar to transition-based):
  • $y$: correct tree, $y'$: incorrect tree

$$
max \left( 0, 1 - \max_{y' \neq y} \sum_{(h,m) \in y'} MLP(v_h \circ v_m) \right) + \sum_{(h,m) \in y} MLP(v_h \circ v_m)
$$
The Model (for Graph-Based)

• Put everything together:
The Model (for Graph-Based)

• Other things to note:
  • Labeled parsing: (similar to transition based)

\[
\text{label}(h, m) = \arg \max_{\ell \in \text{labels}} MLP_{LBL}(v_h \circ v_m)[\ell]
\]

• Loss augmented inference: prevent overfitting. Penalize trees that have high scores but are also VERY wrong

\[
\max(0, 1 + \text{score}(x, y) - \\
\max_{y' \neq y} \sum_{\text{part} \in y'} (\text{score}_{\text{local}}(x, \text{part}) + \mathbb{1}_{\text{part} \not\in y}))
\]
Experiment and Results

• Training:
  • Dataset: Stanford Dependency (SD) for English, Penn Chinese Treebank 5.1 (CTB5)
  • Word dropout: a word is replaced with an unknown symbol with probability proportional to the inverse of its frequency
  • 30 iterations
  • Hyper-parameters

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word embedding dimension</td>
<td>100</td>
</tr>
<tr>
<td>POS tag embedding dimension</td>
<td>25</td>
</tr>
<tr>
<td>Hidden units in $MLP$</td>
<td>100</td>
</tr>
<tr>
<td>Hidden units in $MLP_{LBL}$</td>
<td>100</td>
</tr>
<tr>
<td>BI-LSTM Layers</td>
<td>2</td>
</tr>
<tr>
<td>BI-LSTM Dimensions (hidden/output)</td>
<td>125 / 125</td>
</tr>
<tr>
<td>$\alpha$ (for word dropout)</td>
<td>0.25</td>
</tr>
<tr>
<td>$p_{agg}$ (for exploration training)</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 2: Hyper-parameter values used in experiments
## Experiment and Results

- **UAS**: unlabeled attachment score, **LAS**: labeled attachment score
- **Model** much simpler but very competitive results

<table>
<thead>
<tr>
<th>System</th>
<th>Method</th>
<th>Representation</th>
<th>Emb</th>
<th>PTB-YM UAS</th>
<th>PTB-YM LAS</th>
<th>PTB-SD UAS</th>
<th>PTB-SD LAS</th>
<th>CTB UAS</th>
<th>CTB LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>graph, 1st order</td>
<td>2 BiLSTM vectors</td>
<td>YES</td>
<td>93.0</td>
<td>90.9</td>
<td>86.5</td>
<td>84.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>transition (greedy, dyn-oracle)</td>
<td>4 BiLSTM vectors</td>
<td>YES</td>
<td>93.6</td>
<td>91.5</td>
<td>87.4</td>
<td>85.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>This work</td>
<td>transition (greedy, dyn-oracle)</td>
<td>11 BiLSTM vectors</td>
<td>YES</td>
<td>93.9</td>
<td>91.9</td>
<td>87.6</td>
<td>86.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZhangNivre11</td>
<td>transition (beam)</td>
<td>large feature set (sparse)</td>
<td>YES</td>
<td>93.0</td>
<td>90.9</td>
<td>86.5</td>
<td>84.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martins13 (TurboParser)</td>
<td>graph, 3rd order+</td>
<td>large feature set (sparse)</td>
<td>YES</td>
<td>93.1</td>
<td>90.9</td>
<td>86.5</td>
<td>84.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pei15</td>
<td>graph, 2nd order</td>
<td>large feature set (dense)</td>
<td>YES</td>
<td>93.0</td>
<td>90.9</td>
<td>86.5</td>
<td>84.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dyer15</td>
<td>transition (greedy)</td>
<td>Stack-LSTM + composition</td>
<td></td>
<td>92.7</td>
<td>90.6</td>
<td>86.1</td>
<td>84.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ballesteros16</td>
<td>transition (greedy, dyn-oracle)</td>
<td>Stack-LSTM + composition</td>
<td></td>
<td>92.4</td>
<td>90.0</td>
<td>85.7</td>
<td>84.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weiss15</td>
<td>transition (greedy)</td>
<td>large feature set (dense)</td>
<td>YES</td>
<td>93.2</td>
<td>91.2</td>
<td>86.5</td>
<td>84.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Weiss15</td>
<td>transition (beam)</td>
<td>large feature set (dense)</td>
<td>YES</td>
<td>93.2</td>
<td>91.2</td>
<td>86.5</td>
<td>84.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pei15</td>
<td>graph, 2nd order</td>
<td>large feature set (dense)</td>
<td>YES</td>
<td>93.3</td>
<td>90.9</td>
<td>87.1</td>
<td>85.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dyer15</td>
<td>transition (greedy)</td>
<td>Stack-LSTM + composition</td>
<td>YES</td>
<td>93.1</td>
<td>90.9</td>
<td>87.1</td>
<td>85.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ballesteros16</td>
<td>transition (greedy, dyn-oracle)</td>
<td>Stack-LSTM + composition</td>
<td>YES</td>
<td>93.6</td>
<td>91.4</td>
<td>87.6</td>
<td>86.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LeZuidema14</td>
<td>reranking /blend</td>
<td>inside-outside recursive net</td>
<td>YES</td>
<td>93.1</td>
<td>91.5</td>
<td>86.1</td>
<td>83.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zhu15</td>
<td>reranking /blend</td>
<td>recursive conv-net</td>
<td>YES</td>
<td>93.8</td>
<td>91.5</td>
<td>86.1</td>
<td>83.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>