NLP Programming Tutorial 12 - Dependency Parsing

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Interpreting Language is Hard!

I saw a girl with a telescope

- “Parsing” resolves structural ambiguity in a formal way
Two Types of Parsing

- **Dependency**: focuses on relations between words

  ![Dependency Diagram](image)

  *I saw a girl with a telescope*

- **Phrase structure**: focuses on identifying phrases and their recursive structure

  ![Phrase Structure Diagram](image)

  *I saw a girl with a telescope*
Dependencies Also Resolve Ambiguity

I saw a girl with a telescope

I saw a girl with a telescope
Dependencies

- **Typed**: Label indicating relationship between words
  
  ```
  nsubj prep dobj 
  det det pobj
  ```
  
  I saw a girl with a telescope

- **Untyped**: Only which words depend
  
  ```
  nsubj det det dobj pobj
  ```
  
  I saw a girl with a telescope
Dependency Parsing Methods

- **Shift-reduce**
  - Predict from left-to-right
  - Fast (linear), but slightly less accurate?
  - MaltParser

- **Spanning tree**
  - Calculate full tree at once
  - Slightly more accurate, slower
  - MSTParser, Eda (Japanese)

- **Cascaded chunking**
  - Chunk words into phrases, find heads, delete non-heads, repeat
  - CaboCha (Japanese)
Maximum Spanning Tree

- Each dependency is an edge in a directed graph
- Assign each edge a score (with machine learning)
- Keep the tree with the highest score

(Chu-Liu-Edmonds Algorithm)
Cascaded Chunking

- Works for Japanese, which is strictly head-final
- Divide sentence into chunks, head is rightmost word

私は望遠鏡で女の子を見た
Shift-Reduce

- Process words one-by-one left-to-right
- Two data structures
  - **Queue**: of unprocessed words
  - **Stack**: of partially processed words
- At each point choose
  - **shift**: move one word from queue to stack
  - **reduce left**: top word on stack is head of second word
  - **reduce right**: second word on stack is head of top word
- Learn how to choose each action with a classifier
Shift Reduce Example

**Stack**

<table>
<thead>
<tr>
<th>I saw a girl</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift</td>
</tr>
<tr>
<td>I saw a girl</td>
</tr>
<tr>
<td>shift</td>
</tr>
<tr>
<td>I saw</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>saw a girl</td>
</tr>
<tr>
<td>shift</td>
</tr>
<tr>
<td>saw a girl</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>I</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Queue**

<table>
<thead>
<tr>
<th>saw a girl</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
</tr>
<tr>
<td>shift</td>
</tr>
<tr>
<td>saw</td>
</tr>
<tr>
<td>r left</td>
</tr>
<tr>
<td>saw</td>
</tr>
<tr>
<td>r right</td>
</tr>
<tr>
<td>saw</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>girl</td>
</tr>
<tr>
<td>a</td>
</tr>
</tbody>
</table>
Classification for Shift-Reduce

- Given a state:

  \[
  \begin{array}{c|c}
  \text{Stack} & \text{Queue} \\
  \text{saw a} & \text{girl} \\
  \text{l} &
  \end{array}
  \]

- Which action do we choose?
  - shift
  - r left
  - r right

- Correct actions $\rightarrow$ correct tree
Classification for Shift-Reduce

- We have a weight vector for “shift” “reduce left” “reduce right”
  \[
  \mathbf{W} = \begin{bmatrix} W_s & W_l & W_r \end{bmatrix}
  \]

- Calculate feature functions from the queue and stack
  \[\phi(\text{queue}, \text{stack})\]

- Multiply the feature functions to get scores
  \[s_s = w_s \cdot \phi(\text{queue}, \text{stack})\]

- Take the highest score
  \[s_s > s_l \land s_s > s_r \rightarrow \text{do shift}\]
Features for Shift Reduce

- Features should generally cover at least the last stack entries and first queue entry

<table>
<thead>
<tr>
<th>Word</th>
<th>POS</th>
<th>stack[-2]</th>
<th>stack[-1]</th>
<th>queue[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>saw</td>
<td>VBD</td>
<td></td>
<td>a</td>
<td>girl</td>
</tr>
<tr>
<td>a</td>
<td>DET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>girl</td>
<td>NN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\phi_{W-2saw,W-1a} = 1 \quad \phi_{W-1a,W0girl} = 1 \\
\phi_{W-2saw,P-1DET} = 1 \quad \phi_{W-1a,P0NN} = 1 \\
\phi_{P-2VBD,W-1a} = 1 \quad \phi_{P-1DET,W0girl} = 1 \\
\phi_{P-2VBD,P-1DET} = 1 \quad \phi_{P-1DET,P0NN} = 1
\]
Algorithm Definition

• The algorithm **ShiftReduce** takes as input:
  • Weights $w_s, w_l, w_r$
  • A queue $= [(1, \text{word}_1, \text{POS}_1), (2, \text{word}_2, \text{POS}_2), ...]$  
• starts with a stack holding the special ROOT symbol:
  • $stack = [ (0, \text{"ROOT"}, \text{"ROOT"}) ]$
• processes and returns:
  • $heads = [ -1, head_1, head_2, ... ]$
**Shift Reduce Algorithm**

**ShiftReduce**(queue)

`make` list `heads`

`stack = [(0, "ROOT", "ROOT")]

while |queue| > 0 or |stack| > 1:

    `feats = MAKEFEATS(stack, queue)`
    `s_s = w_s * feats`  # Score for "shift"
    `s_l = w_l * feats`  # Score for "reduce left"
    `s_r = w_r * feats`  # Score for "reduce right"

    if `s_s >= s_l` and `s_s >= s_r` and |queue| > 0:
        `stack.push(queue.popleft())`  # Do the shift
    elif `s_l >= s_r`:
        `heads[ stack[-2].id ] = stack[-1].id`
        `stack.remove(-2)`
    else:
        `heads[ stack[-1].id ] = stack[-2].id`
        `stack.remove(-1)`
Training Shift-Reduce

- Can be trained using perceptron algorithm
- Do parsing, if correct answer $corr$ different from classifier answer $ans$, update weights
- e.g. if $ans = \text{SHIFT}$ and $corr = \text{LEFT}$

\[ w_s \leftarrow \varphi(queue,stack) \]
\[ w_l \rightarrow \varphi(queue,stack) \]
Keeping Track of the Correct Answer (Initial Attempt)

- Assume we know correct head of each stack entry:
  
  \[
  \text{stack}[-1].\text{head} == \text{stack}[-2].\text{id} \quad \text{(left is head of right)}
  \]
  \[
  \rightarrow \text{corr} = \text{RIGHT}
  \]
  
  \[
  \text{stack}[-2].\text{head} == \text{stack}[-1].\text{id} \quad \text{(right is head of left)}
  \]
  \[
  \rightarrow \text{corr} = \text{LEFT}
  \]
  
  else
  
  \[
  \rightarrow \text{corr} = \text{SHIFT}
  \]

- Problem: too greedy for right-branching dependencies
Keeping Track of the Correct Answer (Revised)

- Count the number of unprocessed children

- \( \text{stack[-1].head} == \text{stack[-2].id} \) (right is head of left)
  \( \text{stack[-1].unproc} == 0 \) (left no unprocessed children)
  \( \rightarrow \text{corr} = \text{RIGHT} \)

- \( \text{stack[-2].head} == \text{stack[-1].id} \) (left is head of right)
  \( \text{stack[-2].unproc} == 0 \) (right no unprocessed children)
  \( \rightarrow \text{corr} = \text{LEFT} \)

- else
  \( \rightarrow \text{corr} = \text{SHIFT} \)

- Increase \( \text{unproc} \) when reading in the tree
  When we reduce a head, decrement \( \text{unproc} \)
  \( \text{corr} == \text{RIGHT} \) \( \rightarrow \text{stack[-1].unproc} -= 1 \)
Shift Reduce Training Algorithm

\texttt{ShiftReduceTrain(queue)}

\begin{itemize}
  \item \texttt{make list heads}
  \item \texttt{stack = [(0, “ROOT”, “ROOT”)]}
  \item \texttt{while \mid \texttt{queue} \mid > 0 \textbf{or} \mid \texttt{stack} \mid > 1:}
    \begin{itemize}
      \item \texttt{feats = \texttt{MAKEFEATS}(stack, queue)}
      \item \texttt{calculate ans} \quad \# \texttt{Same as ShiftReduce}
      \item \texttt{calculate corr} \quad \# \texttt{Previous slides}
      \item \texttt{if ans \neq corr:}
        \begin{itemize}
          \item \texttt{w}_{\texttt{ans}} -= \texttt{feats}
          \item \texttt{w}_{\texttt{corr}} += \texttt{feats}
        \end{itemize}
      \item \texttt{perform action according to corr}
    \end{itemize}
\end{itemize}
CoNLL File Format:

- Standard format for dependencies
- Tab-separated columns, sentences separated by space

<table>
<thead>
<tr>
<th>ID</th>
<th>Word</th>
<th>Base</th>
<th>POS</th>
<th>POS2</th>
<th>?</th>
<th>Head</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ms.</td>
<td>ms.</td>
<td>NNP</td>
<td>NNP</td>
<td>_</td>
<td>2</td>
<td>DEP</td>
</tr>
<tr>
<td>2</td>
<td>haag</td>
<td>haag</td>
<td>NNP</td>
<td>NNP</td>
<td>_</td>
<td>3</td>
<td>NP-SBJ</td>
</tr>
<tr>
<td>3</td>
<td>plays</td>
<td>plays</td>
<td>VBZ</td>
<td>VBZ</td>
<td>_</td>
<td>0</td>
<td>ROOT</td>
</tr>
<tr>
<td>4</td>
<td>elianti</td>
<td>elianti</td>
<td>NNP</td>
<td>NNP</td>
<td>_</td>
<td>3</td>
<td>NP-OBJ</td>
</tr>
<tr>
<td>5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>_</td>
<td>3</td>
<td>DEP</td>
</tr>
</tbody>
</table>
Exercise
Exercise

- **Write** `train-sr.py test-sr.py`
- **Train the program**
  - Input: `data/mstparser-en-train.dep`
- **Run the program on actual data:**
  - `data/mstparser-en-test.dep`
- **Measure:** accuracy with `script/grade-dep.py`
- **Challenge:**
  - think of better features to use
  - use a better classification algorithm than perceptron
  - analyze the common mistakes
Thank You!