Dependency Grammars and Parsing

CMSC 473/673

UMBC

November 20th & 22nd, 2017
Course Announcement 1: Assignment 3

Due Wednesday (< 2 days)

Any questions?
Course Announcement 2: Graduate Paper 2

Due Monday 11/27 (~7 days)

Any questions?
Recap from last time...
PCFG Likelihood

\[ p(w_1 w_2 w_3 \ldots w_N) \]

likelihood of word sequence \( w_1 w_2 \ldots w_N \)

\[ p(S \rightarrow^+ w_1 w_2 w_3 \ldots w_N) \]

likelihood of word sequence \( w_1 w_2 \ldots w_N \) based on starting at \( S \)

\[ p(\ ) + p(\ ) + p(\ ) + \ldots \]
PCFG Likelihood

\[ p(S \rightarrow^+ w_1 \, w_2 \, w_3 \, \ldots \, w_N) \]
PCFGs: Inside Algorithm

\[ \beta(X, s, t) = \sum_{Y, Z} \sum_{k : s < k < t} \beta(Y, s, k) \times \beta(Z, k, t) \times p(X \rightarrow YZ) \]

1. that start from non-terminal X, with left index s
2. that terminate after the t\(^{th}\) word
3. that emit the observations from s (inclusive) to t (exclusive)

All valid rules
\[ X \rightarrow YZ \]

All possible splits of the s \(\rightarrow\) t span
\[ \beta = \text{WeightedCell}[K][N][N+1] \]

\[
\text{for}(j = 1; j \leq N; ++j) \{
\quad \beta[X][j-1][j] += \left[ p(X \rightarrow \text{word}_j) \text{ for non-terminal } X \text{ in } G \text{ if } X \rightarrow \text{word}_j \right]
\}
\]

\[
\text{for}(\text{width} = 2; \text{width} \leq N; ++\text{width}) \{
\quad \text{for}(\text{start} = 0; \text{start} < N - \text{width}; ++\text{start}) \{
\quad \quad \text{end} = \text{start} + \text{width}
\quad \quad \text{for}(\text{mid} = \text{start}+1; \text{mid} < \text{end}; ++\text{mid}) \{
\quad \quad \quad \text{for}(\text{rule } X \rightarrow Y Z : G) \{
\quad \quad \quad \quad \beta[X][\text{start}][\text{end}] += \left[ p(X \rightarrow Y Z) \times \beta[Y][\text{start}][\text{mid}] \times \beta[Z][\text{mid}][\text{end}] \right] \text{ for rule } X \rightarrow Y Z : G
\quad \quad \quad \}
\quad \quad \}
\quad \}
\}
\]
## CKY Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Weights</th>
<th>(\oplus)</th>
<th>(\otimes)</th>
<th>(0)</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recognizer</td>
<td>Boolean (True/False)</td>
<td>or</td>
<td>and</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>Viterbi</td>
<td>[0,1]</td>
<td>max</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Inside</td>
<td>[0,1]</td>
<td>+</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Adapted from Jason Eisner
PCFG Outside Score

\[ p(w_1 w_2 \ldots w_{l-1} A w_{m+1} \ldots w_N) \]

likelihood of non-terminal A from l to m, and surrounding words

\[ p(S \rightarrow^+ w_1 w_2 \ldots w_{l-1} A w_{m+1} \ldots w_N) \]
PCFG Outside Score

\[ p(S \rightarrow^+ w_1 w_2 \ldots w_{l-1} A w_{m+1} \ldots w_N) \]

Option 1: \( B \rightarrow C A \)

*must have already considered this word*
PCFG Outside Score

\[ p(S \rightarrow^+ w_1 w_2 \ldots w_{l-1} A w_{m+1} \ldots w_N) \]

Option 1: \( B \rightarrow C A \)

Option 2: \( B \rightarrow A C \)

must have already considered these words
PCFGs: Outside Algorithm

\[
\alpha(X, s, t) =
\]

Option 1: \( B \rightarrow CA \)

\[
\sum_{Y, Z} \sum_{k: 0 \leq k < s} \beta(Z, k, s) \cdot \alpha(Y, k, t) \cdot p(Y \rightarrow ZX) +
\]

Option 2: \( B \rightarrow AC \)

\[
\sum_{Y, Z} \sum_{k: t \leq k \leq N} \beta(Z, t, k) \cdot \alpha(Y, s, k) \cdot p(Y \rightarrow XZ)
\]

\( \alpha(X, s, t) \) is the total probability of all derivations:

1. that include non-terminal \( X \) (but not words) from \( s \) to \( t \)
2. and all (& only) the observed words before \( s \) and after \( t \)
\[ \alpha = \text{WeightedCell}[K][N][N+1] \]

```
for (width = N; width \geq 1; --width) {
    for (start = 0; start \leq N - width; ++start) {
        end = start + width
        for (mid = start+1; mid < end; ++mid) {
            \[ \alpha[Y][start][mid] += \]
            \[ \left[ p(X \rightarrow Y Z) \times \alpha[Y][start][end] \times \beta[Z][mid][end] \right] \]
            for rule \( X \rightarrow Y Z : G \)

            \[ \alpha[Z][mid][end] += \]
            \[ \left[ p(X \rightarrow Y Z) \times \alpha[Z][start][end] \times \beta[Y][mid][end] \right] \]
            for rule \( X \rightarrow Y Z : G \)
        }
    }
}
```

Outside Algorithm
## CKY Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Weights</th>
<th>(\oplus)</th>
<th>(\otimes)</th>
<th>(0)</th>
<th>(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Recognizer</strong></td>
<td>Boolean (True/False)</td>
<td>or</td>
<td>and</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td><strong>Viterbi</strong></td>
<td>[0,1]</td>
<td>max</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Inside</strong></td>
<td>[0,1]</td>
<td>+</td>
<td>*</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Adapted from Jason Eisner
Inside-Outside Algorithm

1. Run inside algorithm (bottom-up, small-to-large)

2. Run outside algorithm (top-down, large-to-small)

Q: Why?
A: Compute Posterior Probabilities (Expectations)
Getting Expected Counts

\[ \alpha_{[VP, 1, 4]} = p(\text{Papa VP with a spoon}) \]

\[ \beta_{[VP, 1, 4]} = p(\text{ate the caviar | VP}) \]

\[ \frac{\alpha_{[VP, 1, 4]} \ast \beta_{[VP, 1, 4]}}{\beta_{[S, 0, 7]}} = p(\text{VP}_{1 \rightarrow 4} | \text{Papa ... spoon}) \]
Getting Expected Counts

\[
\alpha[VP, 1, 4] = p(\text{Papa VP with a spoon})
\]

\[
\alpha[VP, 1, 4] \cdot \beta[V, 1, 2] \cdot \beta[NP, 2, 4] \cdot P(VP \rightarrow V NP) \]

\[
\beta[S, 0, 7] = p(VP_1 \rightarrow V_1 \ NP_2 \rightarrow V_1 | \text{Papa ... spoon})
\]
Expected Counts

\[
\mathbb{E}[X \rightarrow a \mid w_1w_2 \cdots w_N] = \frac{p(X \rightarrow a)}{L(w_1w_2 \cdots w_N)} \sum_{0 \leq i < N : w_i = a} \alpha(X, i, i + 1)
\]

\[
\mathbb{E}[X \rightarrow Y Z \mid w_1w_2 \cdots w_N] = \frac{p(X \rightarrow Y Z)}{L(w_1w_2 \cdots w_N)} \sum_{0 \leq i < k < j \leq N} \alpha(X, i, j)\beta(Y, i, k)\beta(Z, k, j)
\]
Expectation Maximization (EM)

0. Assume some value for your parameters $p(X \rightarrow Y Z)$

Two step, iterative algorithm

1. E-step: count under uncertainty, assuming these parameters

$$
\mathbb{E}[X \rightarrow a \mid w_1 w_2 \cdots w_N] = \frac{p(X \rightarrow a)}{L(w_1 w_2 \cdots w_N)_{0 \leq i < N : w_i = a}} \sum_{0 \leq i < j \leq N} \alpha(X, i, i + 1)
$$

$p(z_i)$ $\longrightarrow$ count($z_i, w_i$)

2. M-step: maximize log-likelihood, assuming these uncertain counts

$$
\mathbb{E}[X \rightarrow Y Z \mid w_1 w_2 \cdots w_N] = \frac{p(X \rightarrow Y Z)}{L(w_1 w_2 \cdots w_N)} \sum_{0 \leq i < k < j \leq N} \alpha(X, i, j)\beta(Y, i, k)\beta(Z, k, j)
$$

$p(t)(z)$ $\underset{\text{estimated counts}}{\longleftrightarrow}$ $p(t+1)(z)$

“Inside-outside”
Probabilistic Context Free Grammar (PCFG) Tasks

Find the most likely parse (for an observed sequence)

Calculate the (log) likelihood of an observed sequence $w_1, \ldots, w_N$

Learn the grammar parameters
Structure vs. Word Relations

Constituency trees/analyses: based on *structure*

Dependency analyses: based on word relations
Remember: (P)CFGs Help Clearly Show Ambiguity

I ate the meal with friends.
I ate the meal with friends.
I ate the meal with friends.
I ate the meal with friends.
I ate the meal with friends

CFGs to Labeled Dependencies
Labeled Dependencies

Word-to-word labeled relations

governor (head)

dependent

Chris ate

nsubj

Chris

ate
Labeled Dependencies

Word-to-word labeled relations

governor (head)
dependent

Core dependents of clausal predicates

<table>
<thead>
<tr>
<th>Nominal dep</th>
<th>Predicate dep</th>
<th>Nominal dep</th>
<th>Predicate dep</th>
<th>Modifier word</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsubj</td>
<td>csubj</td>
<td>nsubjpass</td>
<td>csubjpass</td>
<td></td>
</tr>
<tr>
<td>dobj</td>
<td>ecomp</td>
<td>xcomp</td>
<td>xcomp</td>
<td></td>
</tr>
<tr>
<td>iobj</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Non-core dependents of clausal predicates

<table>
<thead>
<tr>
<th>Nominal dep</th>
<th>Predicate dep</th>
<th>Modifier word</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmod</td>
<td>advcl</td>
<td>advmod</td>
</tr>
<tr>
<td></td>
<td>nfincl</td>
<td>neg</td>
</tr>
<tr>
<td></td>
<td>ncmod</td>
<td></td>
</tr>
</tbody>
</table>

Special clausal dependents

<table>
<thead>
<tr>
<th>Nominal dep</th>
<th>Auxiliary</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>vocative</td>
<td>aux</td>
<td>mark</td>
</tr>
<tr>
<td>discourse</td>
<td>auxpass</td>
<td>punct</td>
</tr>
<tr>
<td>expl</td>
<td>cop</td>
<td></td>
</tr>
</tbody>
</table>

Coordination

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>conj</td>
<td>cc</td>
</tr>
</tbody>
</table>

Noun dependents

<table>
<thead>
<tr>
<th>Nominal dep</th>
<th>Predicate dep</th>
<th>Modifier word</th>
</tr>
</thead>
<tbody>
<tr>
<td>nummod</td>
<td>relcl</td>
<td>amod</td>
</tr>
<tr>
<td>appos</td>
<td>nfincl</td>
<td>det</td>
</tr>
<tr>
<td>nmod</td>
<td>ncmod</td>
<td>neg</td>
</tr>
</tbody>
</table>

Compounding and unanalyzed

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>compound</td>
<td>mwe</td>
<td>goeswith</td>
</tr>
<tr>
<td>name</td>
<td>foreign</td>
<td></td>
</tr>
</tbody>
</table>

Case-marking, prepositions, possessive case

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>parataxis</td>
<td>remnant</td>
</tr>
<tr>
<td>dislocated</td>
<td></td>
<td>reparandum</td>
</tr>
</tbody>
</table>

Loose joining relations

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sentence head</td>
<td>Unspecified dependency</td>
<td></td>
</tr>
<tr>
<td>root</td>
<td>dep</td>
<td></td>
</tr>
</tbody>
</table>

Other

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

de Marneffe et al., 2014
<table>
<thead>
<tr>
<th>Language</th>
<th>UD size</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>African Languages</td>
<td>48K</td>
<td></td>
</tr>
<tr>
<td>Ancient Greek</td>
<td>202K</td>
<td></td>
</tr>
<tr>
<td>Arabic</td>
<td>242K</td>
<td></td>
</tr>
<tr>
<td>Arabic-NYUAD</td>
<td>62K</td>
<td></td>
</tr>
<tr>
<td>Arabic-PUD</td>
<td>20K</td>
<td></td>
</tr>
<tr>
<td>Basque</td>
<td>121K</td>
<td></td>
</tr>
<tr>
<td>Belarusian</td>
<td>8K</td>
<td></td>
</tr>
<tr>
<td>Bulgarian</td>
<td>158K</td>
<td></td>
</tr>
<tr>
<td>Buryat</td>
<td>16K</td>
<td></td>
</tr>
<tr>
<td>Catalan</td>
<td>532K</td>
<td></td>
</tr>
<tr>
<td>Chinese</td>
<td>12K</td>
<td></td>
</tr>
<tr>
<td>Chinese-CFL</td>
<td>7K</td>
<td></td>
</tr>
<tr>
<td>Chinese-PUD</td>
<td>21K</td>
<td></td>
</tr>
<tr>
<td>Cornic</td>
<td>4K</td>
<td></td>
</tr>
<tr>
<td>Croatian</td>
<td>107K</td>
<td></td>
</tr>
<tr>
<td>Czech</td>
<td>1,503K</td>
<td></td>
</tr>
<tr>
<td>Czech-CAC</td>
<td>483K</td>
<td></td>
</tr>
<tr>
<td>Czech-CLTT</td>
<td>37K</td>
<td></td>
</tr>
<tr>
<td>Czech-PUD</td>
<td>18K</td>
<td></td>
</tr>
<tr>
<td>Danish</td>
<td>109K</td>
<td></td>
</tr>
<tr>
<td>Dutch</td>
<td>209K</td>
<td></td>
</tr>
<tr>
<td>Dutch-LassySmall</td>
<td>101K</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>254K</td>
<td></td>
</tr>
<tr>
<td>English-ESL</td>
<td>88K</td>
<td></td>
</tr>
<tr>
<td>English-LindS</td>
<td>82K</td>
<td></td>
</tr>
<tr>
<td>English-PUD</td>
<td>21K</td>
<td></td>
</tr>
<tr>
<td>English-ParTUT</td>
<td>49K</td>
<td></td>
</tr>
<tr>
<td>Estonian</td>
<td>47K</td>
<td></td>
</tr>
<tr>
<td>Estonian-PUD</td>
<td>15K</td>
<td></td>
</tr>
<tr>
<td>Finnish</td>
<td>202K</td>
<td></td>
</tr>
<tr>
<td>Finnish-PUD</td>
<td>15K</td>
<td></td>
</tr>
<tr>
<td>French</td>
<td>351K</td>
<td></td>
</tr>
<tr>
<td>French-FTB</td>
<td>556K</td>
<td></td>
</tr>
<tr>
<td>French-PUD</td>
<td>24K</td>
<td></td>
</tr>
<tr>
<td>French-P周岁</td>
<td>27K</td>
<td></td>
</tr>
<tr>
<td>French-Seq</td>
<td>68K</td>
<td></td>
</tr>
<tr>
<td>Galician</td>
<td>126K</td>
<td></td>
</tr>
<tr>
<td>Galician-TreeGal</td>
<td>23K</td>
<td></td>
</tr>
<tr>
<td>German</td>
<td>287K</td>
<td></td>
</tr>
<tr>
<td>German-PUD</td>
<td>29K</td>
<td></td>
</tr>
<tr>
<td>Gothic</td>
<td>55K</td>
<td></td>
</tr>
<tr>
<td>Greek</td>
<td>61K</td>
<td></td>
</tr>
<tr>
<td>Hebrew</td>
<td>115K</td>
<td></td>
</tr>
<tr>
<td>Hindi</td>
<td>351K</td>
<td></td>
</tr>
<tr>
<td>Hindi-PUD</td>
<td>23K</td>
<td></td>
</tr>
<tr>
<td>Hungarian</td>
<td>42K</td>
<td></td>
</tr>
<tr>
<td>Indonesian</td>
<td>121K</td>
<td></td>
</tr>
<tr>
<td>Indonesian-PUD</td>
<td>25K</td>
<td></td>
</tr>
<tr>
<td>Irish</td>
<td>23K</td>
<td></td>
</tr>
<tr>
<td>Italian</td>
<td>270K</td>
<td></td>
</tr>
<tr>
<td>Italian-PUD</td>
<td>22K</td>
<td></td>
</tr>
<tr>
<td>Italian-ParTUT</td>
<td>39K</td>
<td></td>
</tr>
<tr>
<td>Japanese</td>
<td>186K</td>
<td></td>
</tr>
<tr>
<td>Japanese-KTC</td>
<td>18K</td>
<td></td>
</tr>
<tr>
<td>Kazakh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Korean</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Korean-PUD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kargjani</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latin-ITTB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latin-PROIEL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latvian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lithuanian</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maltese</td>
<td></td>
<td></td>
</tr>
<tr>
<td>North Sami</td>
<td>55K</td>
<td></td>
</tr>
<tr>
<td>Norwegian-Bokmaal</td>
<td>310K</td>
<td></td>
</tr>
<tr>
<td>Norwegian-Nynorsk</td>
<td>301K</td>
<td></td>
</tr>
<tr>
<td>Old Church Slavonic</td>
<td>57K</td>
<td></td>
</tr>
<tr>
<td>Persian</td>
<td>191K</td>
<td></td>
</tr>
<tr>
<td>Polish</td>
<td>82K</td>
<td></td>
</tr>
<tr>
<td>Portuguese</td>
<td>210K</td>
<td></td>
</tr>
<tr>
<td>Portuguese-BR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portuguese-PUD</td>
<td>21K</td>
<td></td>
</tr>
<tr>
<td>Romanian</td>
<td>218K</td>
<td></td>
</tr>
<tr>
<td>Russian</td>
<td>93K</td>
<td></td>
</tr>
<tr>
<td>Russian-PUD</td>
<td>19K</td>
<td></td>
</tr>
<tr>
<td>Russian-SynTagRus</td>
<td>1,107K</td>
<td></td>
</tr>
<tr>
<td>Sanskrit</td>
<td>1K</td>
<td></td>
</tr>
<tr>
<td>Serbian</td>
<td>86K</td>
<td></td>
</tr>
<tr>
<td>Slovak</td>
<td>106K</td>
<td></td>
</tr>
<tr>
<td>Slovenian</td>
<td>140K</td>
<td></td>
</tr>
<tr>
<td>Slovenian-SST</td>
<td>29K</td>
<td></td>
</tr>
<tr>
<td>Spanish</td>
<td>423K</td>
<td></td>
</tr>
<tr>
<td>Spanish-AnCoral</td>
<td>847K</td>
<td></td>
</tr>
<tr>
<td>Spanish-PUD</td>
<td>22K</td>
<td></td>
</tr>
<tr>
<td>Swedish</td>
<td>96K</td>
<td></td>
</tr>
<tr>
<td>Swedish-LInES</td>
<td>79K</td>
<td></td>
</tr>
<tr>
<td>Swedish-PUD</td>
<td>19K</td>
<td></td>
</tr>
<tr>
<td>Swedish Sign Language</td>
<td>&lt;1K</td>
<td></td>
</tr>
<tr>
<td>Tamil</td>
<td>8K</td>
<td></td>
</tr>
<tr>
<td>Thai</td>
<td>23K</td>
<td></td>
</tr>
<tr>
<td>Turkish</td>
<td>56K</td>
<td></td>
</tr>
<tr>
<td>Turkish-PUD</td>
<td>19K</td>
<td></td>
</tr>
<tr>
<td>Ukrainian</td>
<td>25K</td>
<td></td>
</tr>
<tr>
<td>Upper Sorbian</td>
<td>11K</td>
<td></td>
</tr>
<tr>
<td>Urdu</td>
<td>13K</td>
<td></td>
</tr>
<tr>
<td>Uyghur</td>
<td>11K</td>
<td></td>
</tr>
<tr>
<td>Vietnamese</td>
<td>43K</td>
<td></td>
</tr>
</tbody>
</table>

**Upcoming UD Treebanks**

- Amharic
- Armenian
- Bangla
- Bengali-ODS
- Cantonese
- Chinese-HK
- Czech-FicTree
- Dargwa
- Faroese
- French-Spoken
- Italian-PoSTWITA
- Lithuanian-KTC
(Labeled) Dependency Parse

Directed graphs

Vertices: linguistic blobs in a sentence
Edges: (labeled) arcs
(Labeled) Dependency Parse

Directed graphs

Vertices: linguistic blobs in a sentence
Edges: (labeled) arcs

Often directed trees

1. A single root node with no incoming arcs
2. Each vertex except root has exactly one incoming arc
3. Unique path from the root node to each vertex
Dependency Parsing as a (Renewed) Core NLP Problem

- Parser
- Grammar
- Evaluation
- Gold (correct) reference trees
- Other NLP task (entity coref., MT, Q&A, ...)

independent operations

sentence 1
sentence 2
sentence 3
sentence 4
Projective Dependency Trees

No crossing arcs

United canceled the morning flights to Houston
Projective Dependency Trees

No crossing arcs

✔ Projective

✖ Not projective
Projective Dependency Trees

No crossing arcs

✔ Projective

项目经理

✖ Not projective

非项目化解析捕捉
• 某些长距离依赖关系
• 自由的词序

United canceled the morning flights to Houston

JetBlue canceled our flight this morning which was already late

非项目化解析捕捉
• 某些长距离依赖关系
• 自由的词序
Are CFGs for Naught?
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

SLP3: 11.4.3
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.

2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.

2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using "appropriate" head rules.
2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

Papa ate the caviar with a spoon
Are CFGs for Naught?

Nope! Simple algorithm from Xia and Palmer (2011)

1. Mark the head child of each node in a phrase structure, using “appropriate” head rules.

2. In the dependency structure, make the head of each non-head child depend on the head of the head-child.

\[
\text{Papa ate the caviar with a spoon}
\]
Dependency Post-Processing (Keep Tree Structure)
Amaranthus, collectively known as amaranth, is a cosmopolitan genus of annual or short-lived perennial plants.
(Some) Dependency Parsing Algorithms

Dynamic Programming
  Eisner Algorithm (Eisner 1996)

Transition-based
  Shift-reduce, arc standard

Graph-based
  Maximum spanning tree
(Some) Dependency Parsing
Algorithms

Dynamic Programming
Eisner Algorithm (Eisner 1996)

Transition-based
Shift-reduce, arc standard

Graph-based
Maximum spanning tree
Shift-Reduce Parsing

Recall from CMSC 331
Shift-Reduce Parsing

Recall from CMSC 331

Bottom-up

Tools: input words, some special root symbol ($), and a stack to hold configurations
Shift-Reduce Parsing

Recall from CMSC 331
Bottom-up

Tools: input words, some special root symbol ($), and a stack to hold configurations

Shift:
  – move tokens onto the stack
  – match top two elements of the stack against the grammar (RHS)

Reduce:
  – If match occurred, place LHS symbol on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold *configurations*

**Shift:**
- move tokens onto the stack
- *decide* if top two elements of the stack form a valid (good) grammatical dependency

**Reduce:**
- If there’s a valid relation, place head on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

**Shift:**
- move tokens onto the stack
- decide *how?*
  - decide if top two elements of the stack form a valid (good) grammatical dependency

**Reduce:**
- If there’s a valid relation, place head on the stack
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

**Shift:**
- move tokens onto the stack
- decide *how?*
  - Search problem!
- decide if top two elements of the stack form a valid (good) grammatical dependency

**Reduce:**
- If there’s a valid relation, place head on the stack

**decide how?**
Search problem!

**What is valid?**
Learn it!
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations.

Shift:
- move tokens onto the stack
- decide if top two elements of the stack form a valid (good) grammatical dependency

Reduce:
- If there’s a valid relation, place head on the stack

What is valid? Learn it!

What are the possible actions?

Decide how? Search problem!
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold configurations

**Shift:**
- move tokens onto the stack
- **decide** if top two elements of the stack form a valid (good) grammatical dependency

**Reduce:**
- If there’s a valid relation, place head on the stack
- **what is valid?** Learn it!
- **what are the possible actions?**

**decide how?** Search problem!
Shift-Reduce Actions

<table>
<thead>
<tr>
<th>Possibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
</tr>
</tbody>
</table>
Shift-Reduce Actions

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
# Shift-Reduce Actions

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>

“Arc standard”
Arc Standard Parsing

state $\leftarrow \{\text{[root]}, \text{[words]}, [] \}$
Arc Standard Parsing

\[
\text{state} \leftarrow \{ \text{[root]}, \text{[words]}, \text{[]} \} \\
\text{while state not final} \{ \\
\}
\]
Arc Standard Parsing

\[
\text{state} \leftarrow \{[\text{root}], [\text{words}], [] \}
\]

while state not final {
    \[
    t \leftarrow \text{ORACLE}(\text{state})
    \]
    \[
    \text{state} \leftarrow \text{APPLY}(t, \text{state})
    \]
}
Arc Standard Parsing

state ← {[root], [words], [] }

while state not final {
    t ← ORACLE(state)
    state ← APPLY(t, state)
}

return state
Arc Standard Parsing

\[
\text{state} \leftarrow \{ \text{[root], [words], []} \}
\]

while \( \text{state} \neq \{ \text{[root], [], [(deps)]} \} \) {
    \[
    \text{t} \leftarrow \text{ORACLE(} \text{state} \text{)}
    \]
    \[
    \text{state} \leftarrow \text{APPLY(} \text{t, state} \text{)}
    \]
}

return \text{state}
Arc Standard Parsing

\[
\text{state} \leftarrow \{\text{[root]}, \text{[words]}, \text{[]} \}
\]

while \( \text{state} \neq \{\text{[root]}, \text{[]}, \text{[(deps)]}\} \) {
  \[
t \leftarrow \text{ORACLE(state)}
  \]
  \[
  \text{state} \leftarrow \text{APPLY}(t, \text{state})
  \]
}

\[
\text{return state}
\]

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
Arc Standard Parsing

\[
\text{state} \leftarrow \{[\text{root}], [\text{words}], [] \}
\]

while \text{state} \neq \{[\text{root}], [], [(deps)]\} {
    \begin{align*}
    t & \leftarrow \text{ORACLE(} \text{state} \text{)} \\
    \text{state} & \leftarrow \text{APPLY(} t, \text{state} \text{)}
    \end{align*}
}

\text{return state}

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
</tbody>
</table>
Papa ate the caviar

work through on board

state $\leftarrow \{[\text{root}], [\text{words}], [] \} \}$

while state $\neq \{[\text{root}], [], [(\text{ deps})]\}$ {
  t $\leftarrow \text{ORACLE}(\text{state})$
  state $\leftarrow \text{APPLY}(t, \text{state})$
}

return state

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some previously seen word</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>Assign some previously seen word as the head of the current word</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>Wait processing the current word; add it for later</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
<tr>
<td>Deps</td>
<td>Stack</td>
<td>Word Buffer</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>---</td>
<td>$</td>
<td>Papa ate the caviar</td>
</tr>
<tr>
<td>---</td>
<td>Papa $</td>
<td>ate the caviar</td>
</tr>
<tr>
<td>---</td>
<td>ate Papa $</td>
<td>the caviar</td>
</tr>
<tr>
<td>ate-&gt;Papa</td>
<td>ate $</td>
<td>caviar</td>
</tr>
<tr>
<td>ate-&gt;Papa</td>
<td>the ate $</td>
<td>---</td>
</tr>
<tr>
<td>ate-&gt;Papa</td>
<td>caviar the ate $</td>
<td>---</td>
</tr>
<tr>
<td>ate-&gt;Papa, caviar-&gt; the</td>
<td>caviar ate $</td>
<td>---</td>
</tr>
<tr>
<td>ate-&gt;Papa, caviar-&gt; the, ate-&gt;caviar</td>
<td>ate $</td>
<td>---</td>
</tr>
<tr>
<td>ate-&gt;Papa, caviar-&gt; the, ate-&gt;caviar, $-&gt;ate</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Arc Standard Parsing

\[
\text{state} \leftarrow \{\text{[root]}, \text{[words]}, \text{[]}\}
\]

\[
\text{while state} \neq \{\text{[root]}, \text{[]}, \text{[(deps)]}\} \{
\]

\[
t \leftarrow \text{PREDICT(state)}
\]

\[
\text{state} \leftarrow \text{APPLY}(t, \text{state})
\]

\[
\}
\]

\[
\text{return state}
\]

<table>
<thead>
<tr>
<th>Possibility</th>
<th>Action Name</th>
<th>Action Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign the current word as the head of some</td>
<td>LEFTARC</td>
<td>Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack</td>
</tr>
<tr>
<td>previously seen word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assign some previously seen word as the head of</td>
<td>RIGHTARC</td>
<td>Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack</td>
</tr>
<tr>
<td>the current word</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wait processing the current word; add it for</td>
<td>SHIFT</td>
<td>Remove the word from the front of the input buffer and push it onto the stack</td>
</tr>
<tr>
<td>later</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arc Standard Parsing

Q: What is the time complexity?

while state ≠ { [root], [], [(deps)] } {
  t ← ORACLE(state)
  state ← APPLY(t, state)
}

return state
Arc Standard Parsing

Q: What is the time complexity?

A: Linear

state ← APPLY(t, state)

return state
Arc Standard Parsing

Q: What is the time complexity?

A: Linear

\[
\text{while state} \neq \{\text{[root]}, [], [(\text{deps})]]\} \{
\quad \text{t} \leftarrow \text{ORACLE(state)}
\quad \text{state} \leftarrow \text{APPLY}(\text{t}, \text{state})
\}
\]

return \text{state}

Q: What’s potentially problematic?
Arc Standard Parsing

Q: What is the time complexity?

A: Linear

Q: What’s potentially problematic?

A: This is a greedy algorithm
Shift-Reduce Dependency Parsing

Tools: input words, some special root symbol ($), and a stack to hold *configurations*

**Shift:**
- move tokens onto the stack
- *decide* if top two elements of the stack form a valid (good) grammatical dependency

**Reduce:**
- If there's a valid relation, place head on the stack
  - what is valid?
    - Learn it!
  - what are the possible actions?
Learning An Oracle (Predictor)

Training data: dependency treebank

Input: configuration

Output: \{LEFTARC, RIGHTARC, SHIFT\}

t ← ORACLE(state)
Learning An Oracle (Predictor)

Training data: dependency treebank
Input: configuration
Output: \{LEFTARC, RIGHTARC, SHIFT\}

\[ t \leftarrow \text{ORACLE}(\text{state}) \]

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration
Learning An Oracle (Predictor)

Training data: dependency treebank

Input: \textit{configuration}

Output: \{\texttt{LEFTARC}, \texttt{RIGHTARC}, \texttt{SHIFT}\}

\[ t \leftarrow \text{ORACLE(state)} \]

- Choose \texttt{LEFTARC} if it produces a correct head-dependent relation given the reference parse and the current configuration
- Choose \texttt{RIGHTARC} if
  - it produces a correct head-dependent relation given the reference parse and
Learning An Oracle (Predictor)

Training data: dependency treebank

Input: configuration

Output: {LEFTARC, RIGHTARC, SHIFT}

t ← ORACLE(state)

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration
- Choose RIGHTARC if
  - it produces a correct head-dependent relation given the reference parse and
  - all of the dependents of the word at the top of the stack have already been assigned
Learning An Oracle (Predictor)

- Choose **LEFTARC** if it produces a correct head-dependent relation given the reference parse and the current configuration.
- Choose **RIGHTARC** if
  - it produces a correct head-dependent relation given the reference parse and
  - all of the dependents of the word at the top of the stack have already been assigned.
Learning An Oracle (Predictor)

Training data: dependency treebank

Input: configuration

Output: \{\textsc{leftarc}, \textsc{rightarc}, \textsc{shift}\}

\[ t \leftarrow \text{ORACLE}(\text{state}) \]

- Choose \textsc{leftarc} if it produces a correct head-dependent relation given the reference parse and the current configuration
- Choose \textsc{rightarc} if
  - it produces a correct head-dependent relation given the reference parse and
  - all of the dependents of the word at the top of the stack have already been assigned
- Otherwise, choose \textsc{shift}
Training the Predictor

Predict action $t$ give configuration $s$

$$t = \phi(s)$$
Training the Predictor

Predict action $t$ given configuration $s$

$t = \phi(s)$

Extract *features* of the configuration

Examples: word forms, lemmas, POS, morphological features
Training the Predictor

Predict action $t$ given configuration $s$

$t = \phi(s)$

Extract *features* of the configuration

Examples: word forms, lemmas, POS, morphological features

How? Perceptron, Maxent, Support Vector Machines, Multilayer Perceptrons, Neural Networks

Take CMSC 478 (678) to learn more about these
Becoming Less Greedy

Beam search

Breadth-first search strategy (CMSC 471/671)

At each stage, keep K options open
Evaluation

Exact Match (per-sentence accuracy)

Unlabeled Attachment Score (UAS)

Labeled Attachment Score (LS, LAS)

Recall/Precision/$F_1$ for particular relation types
From Dependencies to Shallow Semantics
From Syntax to Shallow Semantics

"Open Information Extraction"

Angeli et al. (2015)
From Syntax to Shallow Semantics

“Open Information Extraction”

Angeli et al. (2015)

http://corenlp.run/ (constituency & dependency)

https://github.com/hltcoe/predpatt

http://openie.allenai.org/

http://www.cs.rochester.edu/research/knext/browse/ (constituency trees)

http://rtw.ml.cmu.edu/rtw/