Lecture 16: Statistical Parsing with PCFGs

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center
Where we’re at

Previous lecture:
Standard **CKY** (for non-probabilistic CFGs)
The standard CKY algorithm finds all possible parse trees $\tau$ for a sentence $S = w^{(1)} \ldots w^{(n)}$ under a CFG $G$ in Chomsky Normal Form.

Today’s lecture:
**Probabilistic Context-Free Grammars (PCFGs)**
- CFGs in which each rule is associated with a probability

**CKY for PCFGs (Viterbi):**
- CKY for PCFGs finds the most likely parse tree $\tau^* = \text{argmax } P(\tau \mid S)$ for the sentence $S$ under a PCFG.
Previous Lecture: CKY for CFGs
CKY for standard CFGs

CKY is a bottom-up chart parsing algorithm that finds all possible parse trees $\tau$ for a sentence $S = w^{(1)}\ldots w^{(n)}$ under a CFG $G$ in Chomsky Normal Form (CNF).

- **CNF**: $G$ has two types of rules: $X \rightarrow Y \ Z$ and $X \rightarrow w$  
  (X, Y, Z are nonterminals, w is a terminal)
- CKY is a **dynamic programming** algorithm
- The **parse chart** is an n×n upper triangular matrix:  
  Each cell $\text{chart}[i][j]$ ($i \leq j$) stores **all subtrees** for $w^{(i)}\ldots w^{(j)}$
- Each cell $\text{chart}[i][j]$ has at most **one entry for each nonterminal** $X$ (and **pairs of backpointers** to each pair of $(Y, Z)$ entry in cells $\text{chart}[i][k]$ $\text{chart}[k+1][j]$ from which an $X$ can be formed
- Time Complexity: $O(n^3 |G|)$
Recap: CKY algorithm

1. Create the chart
   (an $n \times n$ upper triangular matrix for an sentence with $n$ words)
   - Each cell chart[i][j] corresponds to the substring $w^{(i)} \ldots w^{(j)}$

2. Initialize the chart (fill the diagonal cells chart[i][i]):
   - For all rules $X \rightarrow w^{(i)}$, add an entry $X$ to chart[i][i]

3. Fill in the chart:
   - Fill in all cells chart[i][i+1], then chart[i][i+2], …, until you reach chart[1][n] (the top right corner of the chart)
   - To fill chart[i][j], consider all binary splits $w^{(i)} \ldots w^{(k)} | w^{(k+1)} \ldots w^{(j)}$
   - If the grammar has a rule $X \rightarrow YZ$, chart[i][k] contains a $Y$ and chart[k+1][j] contains a $Z$, add an $X$ to chart[i][j] with two backpointers to the $Y$ in chart[i][k] and the $Z$ in chart[k+1][j]

4. Extract the parse trees from the $S$ in chart[1][n].
Additional unary rules

In practice, we may allow other unary rules, e.g.

\[
\text{NP} \rightarrow \text{Noun}
\]
(where Noun is also a nonterminal)

In that case, we apply all unary rules to the entries in chart[i][j] after we have checked all binary splits (chart[i][k], chart[k+1][j])

Unary rules are fine as long as there are no “loops” that could lead to an infinite chain of unary productions, e.g.:

\[
\text{X} \rightarrow \text{Y} \quad \text{and} \quad \text{Y} \rightarrow \text{X}
\]

or:

\[
\text{X} \rightarrow \text{Y} \quad \text{and} \quad \text{Y} \rightarrow \text{Z} \quad \text{and} \quad \text{Z} \rightarrow \text{X}
\]
CKY so far…

Each entry in a cell $\text{chart}[i][j]$ is associated with a nonterminal $X$.

If there is a rule $X \rightarrow YZ$ in the grammar, and there is a pair of cells $\text{chart}[i][k], \text{chart}[k+1][j]$ with a $Y$ in $\text{chart}[i][k]$ and a $Z$ in $\text{chart}[k+1][j]$, we can add an entry $X$ to cell $\text{chart}[i][j]$, and associate one pair of backpointers with the $X$ in cell $\text{chart}[i][k]$.

Each entry might have multiple pairs of backpointers. When we extract the parse trees at the end, we can get all possible trees. We will need probabilities to find the single best tree!
How do you count the **number of parse trees** for a sentence?

1. For each **initial item** (e.g. $V \rightarrow \text{eat}$): $\#\text{trees} = 1$
   
   $$\text{trees}(V_v \rightarrow \text{eat}) = 1$$

2. For each **pair of backpointers** (e.g. $VP \rightarrow V \ NP$): **multiply** $\#\text{trees}$ of children
   
   $$\text{trees}(VP_{VP} \rightarrow V \ NP) = \text{trees}(V) \times \text{trees}(NP)$$

3. For each **list of pairs of backpointers** (e.g. $VP \rightarrow V \ NP$ and $VP \rightarrow VP \ PP$): **sum** $\#\text{trees}$
   
   $$\text{trees}(VP) = \text{trees}(VP_{VP \rightarrow V \ NP}) + \text{trees}(VP_{VP \rightarrow VP \ PP})$$
Exercise: CKY parser

I eat sushi with chopsticks with you

S → NP VP
NP → NP PP
NP → sushi
NP → I
NP → chopsticks
NP → you
VP → VP PP
VP → Verb NP
Verb → eat
PP → Prep NP
Prep → with
Dealing with ambiguity: Probabilistic Context-Free Grammars (PCFGs)
Grammars are ambiguous

A grammar might generate multiple trees for a sentence:

What’s the most likely parse $\tau$ for sentence $S$?

We need a model of $P(\tau | S)$
Computing $P(\tau \mid S)$

Using Bayes’ Rule:

$$\arg\max_{\tau} P(\tau \mid S) = \arg\max_{\tau} \frac{P(\tau, S)}{P(S)} = \arg\max_{\tau} P(\tau, S) = \arg\max_{\tau} P(\tau) \text{ if } S = \text{yield}(\tau)$$

The **yield of a tree** is the string of terminal symbols that can be read off the leaf nodes

$\text{yield}(\text{eat} \ NP \text{sushi} \ PP \text{with tuna}) = \text{eat sushi with tuna}$
Computing $P(\tau)$

$T$ is the (infinite) set of all trees in the language:

$$L = \{s \in \Sigma^* | \exists \tau \in T : \text{yield}(\tau) = s \}$$

The set $T$ is generated by a context-free grammar:

$$
\begin{align*}
S & \rightarrow \text{NP} \ \text{VP} \\
S & \rightarrow \text{S} \ \text{conj} \ \text{S} \\
S & \rightarrow \ldots \ldots \\
\text{NP} & \rightarrow \text{Det} \ \text{Noun} \\
\text{NP} & \rightarrow \text{NP} \ \text{PP} \\
\text{NP} & \rightarrow \ldots \ldots \\
\text{VP} & \rightarrow \text{Verb} \ \text{NP} \\
\text{VP} & \rightarrow \text{VP} \ \text{PP} \\
\text{VP} & \rightarrow \ldots \ldots \\
\end{align*}
$$

We need to define $P(\tau)$ such that:

$$\forall \tau \in T : \quad 0 \leq P(\tau) \leq 1$$

$$\sum_{\tau \in T} P(\tau) = 1$$
**Probabilistic Context-Free Grammars**

For every nonterminal $X$, define a probability distribution $P(X \rightarrow \alpha | X)$ over all rules with the same LHS symbol $X$:

<table>
<thead>
<tr>
<th>Production</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow NP\ VP$</td>
<td>0.8</td>
</tr>
<tr>
<td>$S \rightarrow S\ conj\ S$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow Noun$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow Det\ Noun$</td>
<td>0.4</td>
</tr>
<tr>
<td>$NP \rightarrow NP\ PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP \rightarrow NP\ conj\ NP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$VP \rightarrow Verb$</td>
<td>0.4</td>
</tr>
<tr>
<td>$VP \rightarrow Verb\ NP$</td>
<td>0.3</td>
</tr>
<tr>
<td>$VP \rightarrow Verb\ NP\ NP$</td>
<td>0.1</td>
</tr>
<tr>
<td>$VP \rightarrow VP\ PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$PP \rightarrow P\ NP$</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Computing $P(\tau)$ with a PCFG

The probability of a tree $\tau$ is the product of the probabilities of all its rules:

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3 = 0.00384$$
Learning the parameters of a PCFG

If we have a treebank (a corpus in which each sentence is associated with a parse tree), we can just count the number of times each rule appears, e.g.:

\[ S \rightarrow \text{NP VP} . \quad (1000) \quad S \rightarrow S \text{ conj } S . \quad (220) \]

etc.

and then we divide the observed frequency of each rule \( X \rightarrow Y Z \) by the sum of the frequencies of all rules with the same LHS \( X \) to turn these counts into probabilities:

\[ S \rightarrow \text{NP VP} . \quad (p = 1000/1220) \]
\[ S \rightarrow S \text{ conj } S . \quad (p = 220/1220) \]
More on probabilities:

**Computing** $P(s)$:
If $P(\tau)$ is the probability of a tree $\tau$, the probability of a sentence $s$ is the sum of the probabilities of all its parse trees:

$$P(s) = \sum_{\tau: \text{yield}(\tau) = s} P(\tau)$$

**How do we know that** $P(L) = \sum_{\tau} P(\tau) = 1$?
If we have learned the PCFG from a corpus via MLE, this is guaranteed to be the case. If we just set the probabilities by hand, we could run into trouble, as in the following example:

$$S \rightarrow S \ S \ (0.9) \ S \rightarrow w \ (0.1)$$
PCFG parsing (decoding): Probabilistic CKY
Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities. Finding the most likely tree is similar to Viterbi for HMMs:

**Initialization:**
- [optional] Every chart entry that corresponds to a **terminal** (entries $w$ in $cell[i][i]$) has a Viterbi probability $P_{\text{VIT}}(w[i][i]) = 1$ (*)
- Every entry for a **non-terminal** $X$ in $cell[i][i]$ has Viterbi probability $P_{\text{VIT}}(X[i][i]) = P(X \rightarrow w \mid X)$ [and a single backpointer to $w[i][i]$ (*)]

**Recurrence:** For every entry that corresponds to a **non-terminal** $X$ in $cell[i][j]$, keep only the highest-scoring pair of backpointers to any pair of children ($Y$ in $cell[i][k]$ and $Z$ in $cell[k+1][j]$):

$$P_{\text{VIT}}(X[i][j]) = \arg\max_{Y,Z,k} P_{\text{VIT}}(Y[i][k]) \times P_{\text{VIT}}(Z[k+1][j]) \times P(X \rightarrow Y Z \mid X)$$

**Final step:** Return the Viterbi parse for the start symbol $S$ in the top $cell[1][n]$.

*this is unnecessary for simple PCFGs, but can be helpful for more complex probability models
### Probabilistic CKY

#### Input: POS-tagged sentence

John\_N eats\_V pie\_N with\_P cream\_N

<table>
<thead>
<tr>
<th>John</th>
<th>eats</th>
<th>pie</th>
<th>with</th>
<th>cream</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>NP</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.8·0.2·0.3</td>
<td>0.8·0.2·0.06</td>
<td>0.2·0.0036·0.8</td>
</tr>
<tr>
<td>Verb</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>1.0</td>
<td>0.3</td>
<td>1.0·0.3·0.2</td>
<td>0.06</td>
<td>max( 1.0·0.008·0.3, 0.06·0.2·0.3 )</td>
</tr>
<tr>
<td>Prep</td>
<td>PP</td>
<td>PP</td>
<td>PP</td>
<td>PP</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0·1·0.2</td>
<td>1.0·0.2·0.2</td>
<td>0.008</td>
<td>1.0·0.2·0.2</td>
</tr>
<tr>
<td>Noun</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
<td>NP</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.2·0.2·0.2</td>
<td>0.008</td>
<td>0.2·0.2·0.2</td>
</tr>
<tr>
<td>Prep</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0·1·0.2</td>
<td>1.0·0.2·0.2</td>
<td>0.008</td>
<td>1.0·0.2·0.2</td>
</tr>
<tr>
<td>Noun</td>
<td>Det Noun</td>
<td>NP</td>
<td>NP conj NP</td>
<td>NP</td>
</tr>
<tr>
<td>1.0</td>
<td>0.4</td>
<td>0.2·0.2·0.2</td>
<td>0.008</td>
<td>0.2·0.2·0.2</td>
</tr>
<tr>
<td>NP</td>
<td>PP</td>
<td>PP</td>
<td>PP</td>
<td>PP</td>
</tr>
<tr>
<td>1.0</td>
<td>0.2</td>
<td>0.2·0.2·0.2</td>
<td>0.008</td>
<td>0.2·0.2·0.2</td>
</tr>
<tr>
<td>Prep</td>
<td>Prep</td>
<td>Prep</td>
<td>Prep</td>
<td>Prep</td>
</tr>
<tr>
<td>1.0</td>
<td>0.3</td>
<td>1.0·0.3·0.2</td>
<td>0.06</td>
<td>1.0·0.3·0.2</td>
</tr>
<tr>
<td>Noun</td>
<td>Noun</td>
<td>Noun</td>
<td>Noun</td>
<td>Noun</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0·1·0.2</td>
<td>1.0·0.2·0.2</td>
<td>1.0·0.2·0.2</td>
</tr>
<tr>
<td>Verb</td>
<td>Verb</td>
<td>Verb</td>
<td>Verb</td>
<td>Verb</td>
</tr>
<tr>
<td>1.0</td>
<td>20</td>
<td>1.0·20·1.0</td>
<td>20·1.0·1.0</td>
<td>20·1.0·1.0</td>
</tr>
<tr>
<td>S</td>
<td>NP VP</td>
<td>0.8</td>
<td>0.8·0.2·0.3</td>
<td>0.8·0.2·0.2</td>
</tr>
<tr>
<td>S</td>
<td>S conj S</td>
<td>0.2</td>
<td>0.2·0.2·0.2</td>
<td>0.2·0.2·0.1</td>
</tr>
<tr>
<td>NP</td>
<td>Noun</td>
<td>0.2</td>
<td>0.2·0.2·0.2</td>
<td>0.2·0.2·0.1</td>
</tr>
<tr>
<td>NP</td>
<td>Det Noun</td>
<td>0.4</td>
<td>0.4·0.2·0.2</td>
<td>0.4·0.2·0.1</td>
</tr>
<tr>
<td>NP</td>
<td>NP PP</td>
<td>0.2</td>
<td>0.2·0.2·0.2</td>
<td>0.2·0.2·0.1</td>
</tr>
<tr>
<td>NP</td>
<td>NP conj NP</td>
<td>0.2</td>
<td>0.2·0.2·0.2</td>
<td>0.2·0.2·0.1</td>
</tr>
<tr>
<td>VP</td>
<td>Verb</td>
<td>0.3</td>
<td>0.3·0.3·0.2</td>
<td>0.3·0.3·0.2</td>
</tr>
<tr>
<td>VP</td>
<td>Verb NP</td>
<td>0.3</td>
<td>0.3·0.3·0.2</td>
<td>0.3·0.3·0.2</td>
</tr>
<tr>
<td>VP</td>
<td>Verb NP NP</td>
<td>0.1</td>
<td>0.1·0.3·0.2</td>
<td>0.1·0.3·0.2</td>
</tr>
<tr>
<td>VP</td>
<td>VP PP</td>
<td>0.3</td>
<td>0.3·0.3·0.2</td>
<td>0.3·0.3·0.2</td>
</tr>
<tr>
<td>PP</td>
<td>Prep NP</td>
<td>1.0</td>
<td>1.0·1.0·1.0</td>
<td>1.0·1.0·1.0</td>
</tr>
<tr>
<td>Prep</td>
<td>P</td>
<td>1.0</td>
<td>1.0·1.0·1.0</td>
<td>1.0·1.0·1.0</td>
</tr>
<tr>
<td>Noun</td>
<td>N</td>
<td>1.0</td>
<td>1.0·1.0·1.0</td>
<td>1.0·1.0·1.0</td>
</tr>
<tr>
<td>Verb</td>
<td>V</td>
<td>20</td>
<td>20·1.0·20</td>
<td>20·1.0·20</td>
</tr>
</tbody>
</table>
How do we handle flat rules?

S → NP VP 0.8
S → S conj S 0.2
NP → Noun 0.2
NP → Det Noun 0.4
NP → NP PP 0.2
NP → NP conj NP 0.2
VP → Verb 0.3
VP → Verb NP 0.3
VP → Verb NP NP 0.1
VP → VP PP 0.3
PP → Prep NP 1.0

S → S ConjS 0.2
ConjS → conj S 1.0

Define a new nonterminal (ConjS) and add a new rule