Lecture 9: The CKY parsing algorithm

Defining grammars for natural language

Last lecture’s key concepts

Natural language syntax
  Constituents
  Dependencies
  Context-free grammar
  Arguments and modifiers
  Recursion in natural language

An example CFG

DT → {the, a}
N → {ball, garden, house, sushi}
P → {in, behind, with}
NP → DT N
NP → NP PP
PP → P NP

N: noun
P: preposition
NP: “noun phrase”
PP: “prepositional phrase”
Reminder: Context-free grammars

A CFG is a 4-tuple \( \langle N, \Sigma, R, S \rangle \) consisting of:
- A set of nonterminals \( N \)
  (e.g. \( N = \{S, NP, VP, PP, Noun, Verb, ....\} \))
- A set of terminals \( \Sigma \)
  (e.g. \( \Sigma = \{I, you, he, eat, drink, sushi, ball, \} \))
- A set of rules \( R \)
  \( R \subseteq \{ A \rightarrow \beta \text{ with left-hand-side (LHS) } A \in N \text{ and right-hand-side (RHS) } \beta \in (N \cup \Sigma)^* \} \)
- A start symbol \( S \in N \)

Constituents:
Heads and dependents

There are different kinds of constituents:
- **Noun phrases**: the man, a girl with glasses, Illinois
- **Prepositional phrases**: with glasses, in the garden
- **Verb phrases**: eat sushi, sleep, sleep soundly

Every phrase has a **head**:
- **Noun phrases**: the man, a girl with glasses, Illinois
- **Prepositional phrases**: with glasses, in the garden
- **Verb phrases**: eat sushi, sleep, sleep soundly

The other parts are its **dependents**.
Dependents are either **arguments** or **adjuncts**

Is string \( \alpha \) a constituent?
He talks [in class].

Substitution test:
Can \( \alpha \) be replaced by a single word?
He talks [there].

Movement test:
Can \( \alpha \) be moved around in the sentence?
[In class], he talks.

Answer test:
Can \( \alpha \) be the answer to a question?
Where does he talk? - [In class].

Arguments are obligatory

Words subcategorize for specific sets of arguments:
- **Transitive verbs (sbj + obj)**: [John] likes [Mary]
- **All arguments have to be present**: *[John] likes. *likes [Mary].
- **No argument can be occupied multiple times**: *[John] [Peter] likes [Ann] [Mary].
- **Words can have multiple subcat frames**:
  - **Transitive eat (sbj + obj)**: [John] eats [sushi].
  - **Intransitive eat (sbj)**: [John] eats.
Adjuncts are optional

Adverbs, PPs and adjectives can be adjuncts:
- Adverbs: John runs [fast].
  a [very] heavy book.
- PPs: John runs [in the gym].
  the book [on the table]
- Adjectives: a [heavy] book

There can be an arbitrary number of adjuncts:
- John saw Mary.
- John saw Mary [yesterday].
- John saw Mary [yesterday] [in town]
- [Perhaps] John saw Mary [yesterday] [in town] [during lunch]

Heads, Arguments and Adjuncts in CFGs

Heads:
We assume that each RHS has one head, e.g.
- VP → Verb NP (Verbs are heads of VPs)
- NP → Det Noun (Nouns are heads of NPs)
- S → NP VP (VPs are heads of sentences)

Exception: Coordination, lists: VP → VP conj VP

Arguments:
The head has a different category from the parent:
- VP → Verb NP (the NP is an argument of the verb)

Adjuncts:
The head has the same category as the parent:
- VP → VP PP (the PP is an adjunct)

Chomsky Normal Form

The right-hand side of a standard CFG can have an arbitrary number of symbols (terminals and nonterminals):

\[
VP \rightarrow ADV \text{ eat } NP
\]

A CFG in Chomsky Normal Form (CNF) allows only two kinds of right-hand sides:
- Two nonterminals: VP → ADV VP
- One terminal: VP → eat

Any CFG can be transformed into an equivalent CNF:

\[
VP \rightarrow \text{ADVP } VP_1
\]

\[
VP_1 \rightarrow VP_2 \text{ NP}
\]

\[
VP_2 \rightarrow \text{eat}
\]

A note about \(\varepsilon\)-productions

Formally, context-free grammars are allowed to have empty productions (\(\varepsilon = \) the empty string):

\[
VP \rightarrow V NP \quad NP \rightarrow DT Noun \quad NP \rightarrow \varepsilon
\]

These can always be eliminated without changing the language generated by the grammar:

\[
VP \rightarrow V NP \quad VP \rightarrow V \varepsilon \quad NP \rightarrow DT Noun
\]

which in turn becomes

\[
VP \rightarrow V NP \quad VP \rightarrow V \quad NP \rightarrow DT Noun
\]

We will assume that our grammars don’t have \(\varepsilon\)-productions
CKY chart parsing algorithm

Bottom-up parsing:
start with the words
Dynamic programming:
save the results in a table/chart
re-use these results in finding larger constituents

Complexity: $O(n^3|G|)$
$n$: length of string, $|G|$: size of grammar

Presumes a CFG in Chomsky Normal Form:
Rules are all either $A \rightarrow B C$ or $A \rightarrow a$
(with $A,B,C$ nonterminals and $a$ a terminal)

The CKY parsing 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 $X \rightarrow w^{(0)}$, 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], ..., untl you reach chart[1][n] (the top right corner of the chart)
   - To fill chart[i][j], consider all binary splits $w^{(0)} \ldots w^{(k)}w^{(k+1)} \ldots w^{(j)}$
   - 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].
The CKY parsing algorithm

Each cell contains only a **single entry** for each nonterminal. Each entry may have a **list** of pairs of backpointers.

We eat sushi with tuna

What are the terminals in NLP?

Are the “terminals”: words or POS tags?

For toy examples (e.g. on slides), it’s typically the words

With POS-tagged input, we may either treat the POS tags as the terminals, or we assume that the unary rules in our grammar are of the form

POS-tag → word

(so POS tags are the only nonterminals that can be rewritten as words; some people call POS tags “preterminals”)
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 \( \text{chart}[i][j] \) after we’ve checked all binary splits (\( \text{chart}[i][k], \text{chart}[k+1][j] \))

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

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

or:

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

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!

Exercise: CKY parser

I eat sushi with chopsticks with you

I eat sushi with chopsticks with you

\[
\begin{align*}
\text{S} & \rightarrow \text{NP} \quad \text{VP} \\
\text{NP} & \rightarrow \text{NP} \quad \text{PP} \\
\text{NP} & \rightarrow \text{sushi} \\
\text{NP} & \rightarrow \text{I} \\
\text{NP} & \rightarrow \text{chopsticks} \\
\text{NP} & \rightarrow \text{you} \\
\text{VP} & \rightarrow \text{VP} \quad \text{PP} \\
\text{VP} & \rightarrow \text{Verb} \quad \text{NP} \\
\text{Verb} & \rightarrow \text{eat} \\
\text{PP} & \rightarrow \text{Prep} \quad \text{NP} \\
\text{Prep} & \rightarrow \text{with}
\end{align*}
\]
Dealing with ambiguity: Probabilistic Context-Free Grammars (PCFGs)

A grammar might generate multiple trees for a sentence:

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

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

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.

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

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

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

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

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

$\sum_{\tau \in T} P(\tau) = 1$

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

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>0.8</td>
</tr>
<tr>
<td>S → S conj S</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → Noun</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → Det Noun</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → NP conj NP</td>
<td>0.2</td>
</tr>
<tr>
<td>VP → Verb</td>
<td>0.4</td>
</tr>
<tr>
<td>VP → Verb NP</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → Verb NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>PP → 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:

$P(\tau) = \prod_{\text{rules}} P_{\text{rule}}$

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

PCFG parsing (decoding): Probabilistic CKY
Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities.
Finding the most likely tree $\arg\max_\tau P(\tau, s)$ is similar to Viterbi for HMMs:

Initialization: every chart entry that corresponds to a terminal (entries $X$ in cell $[i][i]$) has a Viterbi probability $P_{VIT}(X_{[i][i]}) = 1$.

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 $[k+1][j]$): $P_{VIT}(X_{[i][j]}) = \arg\max_{Y,Z,k} P_{VIT}(Y_{[i][k]} \times P_{VIT}(Z_{[k+1][j]}) \times P(X \rightarrow Y Z | X)$.

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

---

Probabilistic CKY

Input: POS-tagged sentence

<table>
<thead>
<tr>
<th></th>
<th>John</th>
<th>eats</th>
<th>pie</th>
<th>with</th>
<th>cream</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1.0</td>
<td>0.2</td>
<td>S</td>
<td>0.8</td>
<td>0.08</td>
</tr>
<tr>
<td>eats</td>
<td>1.0</td>
<td>0.3</td>
<td>S</td>
<td>0.8</td>
<td>0.08</td>
</tr>
<tr>
<td>pie</td>
<td></td>
<td></td>
<td>S</td>
<td>0.8</td>
<td>0.08</td>
</tr>
<tr>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>0.008</td>
</tr>
<tr>
<td>cream</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
</tr>
</tbody>
</table>

John_N eats_V pie_N with_P cream_N

<table>
<thead>
<tr>
<th></th>
<th>1.0</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>PP</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

Input: POS-tagged sentence

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>NP</th>
<th>VP</th>
<th>S</th>
<th>NP</th>
<th>VP</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1.0</td>
<td>0.2</td>
<td>0.3</td>
<td>John</td>
<td>1.0</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>eats</td>
<td></td>
<td></td>
<td></td>
<td>S</td>
<td>0.8</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>pie</td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
<td>0.2</td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td>VP</td>
<td>0.3</td>
<td>0.008</td>
<td></td>
</tr>
<tr>
<td>cream</td>
<td></td>
<td></td>
<td></td>
<td>VP</td>
<td>0.3</td>
<td>0.008</td>
<td></td>
</tr>
</tbody>
</table>

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 → P NP 1.0