Lecture 15: The CKY parsing algorithm

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Today’s class

Parsing with CFGs:
The CKY (Cocke Kasami Younger) algorithm

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. \( N = \{ S, NP, VP, PP, Noun, Verb, \ldots \} \))
- A set of terminals \( \Sigma \)
  (e.g. \( \Sigma = \{ I, you, he, eat, drink, sushi, ball, \ldots \} \))
- A set of rules \( R \)
  \( R \subseteq \{ A \rightarrow \beta \; \text{with left-hand-side (LHS)} \quad A \in N \quad \text{and right-hand-side (RHS)} \quad \beta \in (N \cup \Sigma)^* \} \)
- A start symbol \( S \in N \)

A note about \( \varepsilon \)-productions

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

\[
\begin{align*}
\text{VP} & \rightarrow \text{V NP} \quad \text{NP} \rightarrow \text{DT Noun} \quad \text{NP} \rightarrow \varepsilon \\
\end{align*}
\]

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

\[
\begin{align*}
\text{VP} & \rightarrow \text{V NP} \quad \text{NP} \rightarrow \text{DT Noun} \quad \text{NP} \rightarrow \varepsilon \\
\text{VP} & \rightarrow \text{V} \quad \text{NP} \rightarrow \varepsilon \\
\text{NP} & \rightarrow \text{DT Noun} \\
\end{align*}
\]

which in turn becomes

\[
\begin{align*}
\text{VP} & \rightarrow \text{V NP} \quad \text{VP} \rightarrow \text{V} \quad \text{NP} \rightarrow \text{DT Noun} \\
\end{align*}
\]

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

Chomsky Normal Form

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

\[
\begin{align*}
\text{VP} & \rightarrow \text{ADV eat NP} \\
\end{align*}
\]

A CFG in Chomsky Normal Form (CNF) allows only two kinds of right-hand sides:

- Two nonterminals: \( \text{VP} \rightarrow \text{ADV VP} \)
- One terminal: \( \text{VP} \rightarrow \text{eat} \)

Any CFG can be transformed into an equivalent CNF:

\[
\begin{align*}
\text{VP} & \rightarrow \text{ADVP VP} \\
\text{VP}_1 & \rightarrow \text{VP} \quad \text{NP} \\
\text{VP}_2 & \rightarrow \text{eat} \\
\end{align*}
\]

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

To recover the parse tree, each entry needs pairs of backpointers.

CS447 Natural Language Processing

1. Create the chart
   (an \(nxn\) 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].

CS447 Natural Language Processing
The CKY parsing algorithm

<table>
<thead>
<tr>
<th>V</th>
<th>VP</th>
<th>buy drinks with milk</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>NP</td>
<td>VP</td>
<td>VP</td>
</tr>
<tr>
<td>VP</td>
<td>V</td>
<td>NP</td>
<td>NP</td>
</tr>
<tr>
<td>VP</td>
<td>VP</td>
<td>PP</td>
<td>NP</td>
</tr>
<tr>
<td>V</td>
<td>drinks</td>
<td>drinks with milk</td>
<td>drugs</td>
</tr>
<tr>
<td>NP</td>
<td>NP</td>
<td>PP</td>
<td>NP</td>
</tr>
<tr>
<td>NP</td>
<td>we</td>
<td></td>
<td>NP</td>
</tr>
<tr>
<td>NP</td>
<td>drinks</td>
<td></td>
<td>NP</td>
</tr>
<tr>
<td>PP</td>
<td>P</td>
<td>NP</td>
<td>P</td>
</tr>
<tr>
<td>P</td>
<td>with</td>
<td></td>
<td>with</td>
</tr>
</tbody>
</table>

Each cell may have one entry for each nonterminal.

We buy drinks with milk

Additional unary rules

In practice, we may allow other unary rules, e.g.
NP → Noun
(where Noun is also a nonterminal)

In that case, we apply all unary rules to the entries in chart[i][j] after we’ve 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.:
X → Y and Y → X
or: X → Y and Y → Z and Z → X

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”)
CKY so far…

Each entry in a cell chart[i][j] is associated with a nonterminal X.

If there is a rule X → YZ in the grammar, and there is a pair of cells chart[i][k], chart[k+1][j] with a Y in chart[i][k] and a Z in chart[k+1][j], we can add an entry X to cell chart[i][j], and associate one pair of backpointers with the X in cell 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 pair of backpointers (e.g. VP → V NP): multiply #trees of children trees(VPVP → V NP) = #trees(V) × #trees(NP)

2. For each list of pairs of backpointers (e.g. VP → V NP and VP → V PP): sum #trees trees(VP) = #trees(VPVP → V NP) + #trees(VPVP → V 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

Cocke Kasami Younger (1)

ckyParse(n):
  initChart(n)
  fillChart(n)

fillChart(n):
  for span = 1...n-1:
    for i = 1...n-span:
      fillCell(i, i+span)

fillCell(i, j):
  for k = i, i+1:
    combineCells(i, k, j)

combineCells(i, k, j):
  for Y in chart[i][k]:
    for Z in chart[k+1][j]:
      for X in Nonterminals:
        if X → YZ in Rules:
          addToCell(chart[i][j], X, Y, Z)

initChart(n):
  for i = 1...n:
    initCell(i, i)

initCell(i, j):
  for c in lex(word[i]):
    addToCell(chart[i][i], c, null, null)

addToCell(Parent, chart, Left, Right)
  if chart.hasEntry(Parent):
    P = chart.getEntry(Parent)
    P.addBackpointers(Left, Right)
  else chart.addEntry(Parent, Left, Right)
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 \( P(\tau | S) \)

Using Bayes’ Rule:

\[
\arg\max_{\tau} P(\tau | S) = \arg\max_{\tau} \frac{P(\tau, S)}{P(S)}
\]

\[
= \arg\max_{\tau} P(\tau, S)
\]

\[
= \arg\max_{\tau} P(\tau) \text{ if } S = \text{yield}(\tau)
\]

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

\[
\text{yield}(\quad V \quad NP \quad NP \quad PP \quad NP)
\]

\[
= \text{eat sushi with tuna}
\]

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

\[
S \rightarrow NP \quad VP
\]

\[
VP \rightarrow \text{Verb} \quad NP
\]

\[
NP \rightarrow \text{Det} \quad \text{Noun}
\]

\[
S \rightarrow S \quad \text{conj} \quad S
\]

\[
VP \rightarrow VP \quad PP
\]

\[
NP \rightarrow NP \quad PP
\]

\[
S \rightarrow \ldots..\quad VP \rightarrow \ldots..\quad NP \rightarrow \ldots..
\]
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>Rule</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:

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3 = 0.00384$$

PCFG parsing (decoding): Probabilistic CKY

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
John_N eats_V pie_N with_P cream_N

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
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