Recitation 7: Trees

Ekin Akyürek & Wei Fang

MIT 6.806-6.864 Spring 2021

Context Free Grammars

CFG

A Context-Free Grammar consists of a tuple $(\mathsf{N}, \Sigma, \mathsf{S}, \mathsf{R})$ such that:

- N is a finite set of non-terminal symbols;
- Σ is a finite set of terminal symbols;
- S is the start symbol;
- R is a finite set of rules of the form $X \to \alpha$ where $X \in N$ and α is a sequence of symbols drawn from $\Sigma \cup N$;

An example **CFG** that can *generate* arithmetic expressions :

$$
\begin{aligned}
E &\longrightarrow E + E, \\
E &\longrightarrow E * E, \\
E &\longrightarrow (E), \\
E &\longrightarrow a \mid b
\end{aligned}
$$

Context Free Grammars

What can you do with a **CFG**?

- **Generate** strings (sentences) by recursively applying the rules.

Language
Set of strings that can be derived from the grammar

Context Free Grammars

$$E \longrightarrow E + E \mid E * E \mid (E) \mid a \mid b$$

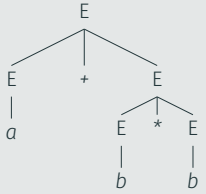
Derivation

Recursive application of the rules

$$E \to E + E \to a + E \to a + E * E \to a + b * E \to a + b * b$$

Derivation Tree

Graphical representation of the derivation.



Example: Palindrome Language

Find a CFG on alphabet $\Sigma = \{a, b\}$ that generates palindromes? (aaa,bb, abbabba, bbabb)

$$
\begin{aligned}
P &\longrightarrow a, \\
P &\longrightarrow b, \\
P &\longrightarrow aPa, \\
P &\longrightarrow bPb, \\
p &\longrightarrow \epsilon
\end{aligned}
$$

Context Free Grammars

What can you do with a **CFG**?

- **Generate** strings (sentences) by recursively applying the rules.
- **Parse** a string to find a possible derivation (existance)

CKY Parsing

CKY

The **Cocke-Kasami-Younger** (CKY, or CYK) **Parsing Algorithm** is a *bottom-up* parser with dynamic programming for a CFG in Chomsky Normal Form

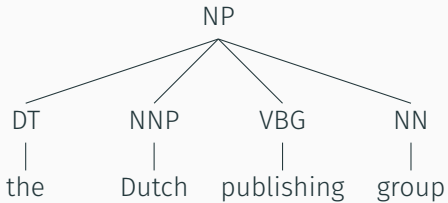
Chomsky Normal Form

A grammar in which each rule takes one of the three following forms:

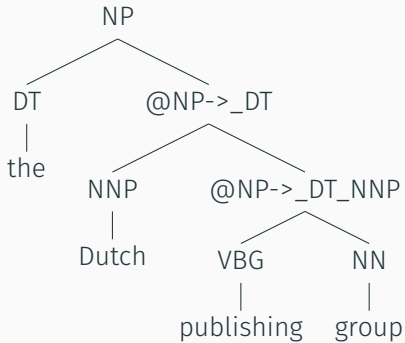
- $S \rightarrow \epsilon$ where S is the start symbol;
- $X \rightarrow Y\ Z$ where Y and Z are non-terminals; or
- $X \rightarrow t$ where t is a terminal.

Markovization

a.

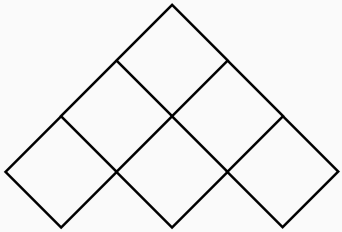


b.



CKY Parsing

```
1: function CKY-PARSE(words, grammar)
2:     Initialize table to the upper half of an n × n matrix
3:     for l in 1 to LEN(words) do
4:         for i in 0 to LEN(words) − l do
5:             if l == 1 then
6:                 for rule X → w such that words[i]==w do
7:                     Put X in table[i, i + 1]
8:                 end for
9:             else
10:                j = i + l
11:                for k in i + 1 to j − 1 do
12:                    for rule X → Y Z such that Y ∈ table[i, k] and Z ∈ table[k, j] do
13:                        Put X in table[i, j]
14:                        Put backpointers from table[i,j][X] to (table[i, k][Y], table[k, j][Z])
15:                    end for
16:                end for
17:            end if
18:        end for
19:    end for
20:    return table
21: end function
```



$$S \rightarrow NP\ VP$$
$$VP \rightarrow M\ V \mid V$$
$$NP \rightarrow N\ NP \mid N$$

$$N \rightarrow can \mid lead \mid poison$$
$$M \rightarrow can \mid must$$
$$V \rightarrow poison \mid lead$$

Probabilistic Context Free Grammars

PCFG

A **probabilistic context-free grammar** consists of a tuple (N, V, S, R, P) such that:

- **N** is a finite set of non-terminal symbols;
- **V** is a finite set of terminal symbols;
- **S** is the start symbol;
- **R** is a finite set of rules of the form $X \to \alpha$ where $X \in N$ and α is a sequence of symbols drawn from $V \cup N$;
- **P** is a mapping from R into probabilities, such that for each $X \in N$

$$\sum_{[X \to \alpha] \in R} P(X \to \alpha) = 1$$

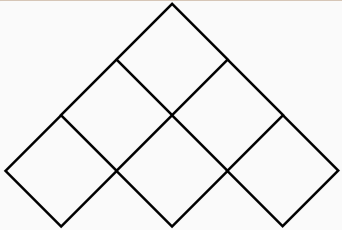
Probabilistic Context Free Grammars

What can you do with a **PCFG**?

- **Sample** a sentence with a tree
  - $\rightarrow$ S,T $\sim$ P(S,T)
- **Calculate** joint probability of a derivation
  - $\rightarrow$ $P(S, T)$ by multiplying rule probabilities on the derivation
- **Assign** probability to the strings (sentences) in the grammar
  - $\rightarrow$ $P(S) = \Sigma_T P(S, T)$ by inside-outside algorithm
- **Parse** a string to find maximum probable derivation tree
  - $\rightarrow$ $\arg\max_T P(T|S)$ by **CKY Algorithm**

Viterbi CKY Parsing

```
1: function VITERBI-CKY-PARSE(words,grammar)
2:     Initialize table to the upper half of an n × n matrix
3:     for l in 1 to LEN(words) do
4:         for i in 0 to LEN(words) − l do
5:             if l == 1 then
6:                 for rule X → w such that words[i]==w do
7:                     Put X in table[i, i + 1] with score P(X → w)
8:                 end for
9:             else
10:                j = i + l
11:                for k in i + 1 to j − 1 do
12:                    for rule X → Y Z such that Y ∈ table[i, k] with score c1 and Z ∈ table[k, j] with score c2 do
13:                        Let c be equal to P(X → Y Z) × c1 × c2
14:                        if c > current score of table[i, j][X] then
15:                            Put X in table[i, j] with score c
16:                            Put backpointers from table[i,j][X] to (table[i, k][Y], table[k, j][Z])
17:                        end if
18:                    end for
19:                end for
20:            end if
21:        end for
22:    end for
23:    return table
24: end function
```



$$S \xrightarrow{1.0} NP\ VP$$

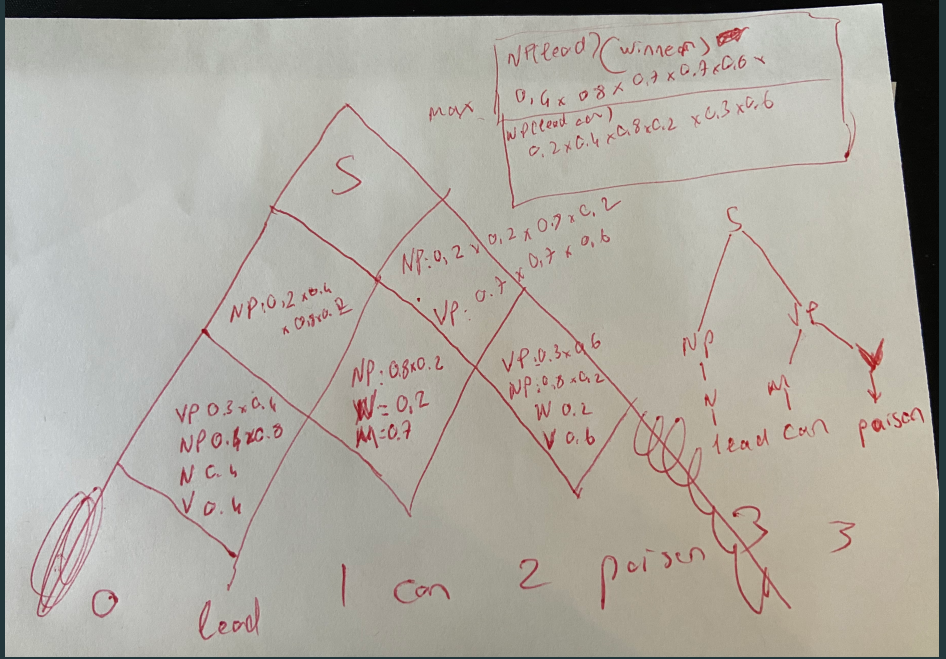
$$VP \xrightarrow{0.7} M\ V \mid \xrightarrow{0.3} V$$

$$NP \xrightarrow{0.2} N\ NP \mid \xrightarrow{0.8} N$$

$$N \xrightarrow{0.2} can \mid \xrightarrow{0.4} lead \mid \xrightarrow{0.2} poison$$

$$M \xrightarrow{0.7} can \mid \xrightarrow{0.3} must$$

$$V \xrightarrow{0.6} poison \mid \xrightarrow{0.4} lead$$


max
NP lead) (will can)
0.4 x 0.8 x 0.7 x 0.7 x 0.6 x
NP lead can)
0.2 x 0.4 x 0.8 x 0.2 x 0.3 x 0.6
S
NP: 0.2 x 0.4 x 0.8 x 0.2
VP 0.3 x 0.4
NP 0.8 x 0.3
N 0.4
V 0.4
NP: 0.2 x 0.7 x 0.2
VP: 0.7 x 0.7 x 0.6
NP: 0.8 x 0.2
W = 0.2
M = 0.7
VP: 0.3 x 0.6
NP: 0.8 x 0.2
W 0.2
V 0.6
S
NP
VP
N
M
lead can poison
0 lead 1 can 2 poison 3 3

Viterbi CKY

Viterbi CKY

`table[i,j][X]` stores the highest-scoring tree with root X covering the span i:j

- **Base Case:** $table[i,i+1][X] = P(X \rightarrow w_i)$
- **Inductive Case:** $table[i,j][X] = \max_{k \in [i+1, j-1]} \max_{Y,Z} P(X \rightarrow YZ) P(Y \rightarrow i, k) P(Z \rightarrow k, j)$

PCFG vs HMM

Viterbi CKY

table[i,j][X] stores the highest-scoring tree with root X covering the span i:j

- **Base Case:** $table[i,i+1][X] = P(X \rightarrow w_i)$
- **Inductive Case:** $table[i,j][X] = \max_{k \in [i+1, j-1]} \max_{Y,Z} P(X \rightarrow YZ) P(Y \rightarrow i, k) P(Z \rightarrow k, j)$

Inside-Outside Algorithm: replace max with sum and apply the same algorithm

Viterbi HMM

table[t][s] stores the highest-scoring sequence with state s at time step t.

- **Base Case:** $table[1][s] = b_s(o_1)\pi_s$
- **Inductive Case:** $table[t][s] = b_s(o_t) \max_{s'} table[t\text{-}1][s'] a_{s's}$

Forward Algorithm: replace max with sum and apply the same algorithm

Comparison with HMM

HMM

1. Compute $P(S|\lambda)$:
   Forward probability (before):

$$\alpha_i(t) = P(w_{1t}, q_t = i|\lambda)$$

   Backward probability (after):

$$\beta_i(t) = P(w_{(t+1)T}|q_t = i|\lambda)$$

2. Best state seq. Q given W:
   Viterbi algorithm
3. Learn λ:
   EM (Baum-Welch)

PCFG

1. Compute $P(S|G)$:
   Outside probability (above):

$$\alpha_x(i,j) = P(w_{1(i-1)}, N^x_{ij}, w_{(j+1)L}|G)$$

   Inside probability (below):

$$\beta_x(i,j) = P(w_{ij}|N^x_{ij}, G)$$

2. Best tree T given W:
   Viterbi Parsing
3. Learn G:
   EM

Features of PCFGs

- Good for *grammar induction*
- Does not give a very good idea of the plausibility of different parses, since its estimates are based purely on structural factors and do not factor in lexical co-occurrence
- Real text tends to have grammatical mistakes and errors
- PCFG gives a probabilistic language model (CFG does not). In practice, PCFG is a worse language model than *n*-gram since an *n*-gram model takes local lexical context into account whereas a PCFG uses none
- PCFG have certain biases, such as assigning larger probability to smaller trees (all else being equal).

Limitations

- Slow compared to HMM (linear).
  - → HMM forward/backward: $O(TN^2)$, T is length, N is number of states
  - → PCFG inside/outside: $O(T^3N^3)$, T is length, N is number of nonterminals
- Found to have poor local minima
- In practice requires many more nonterminals than are theoretically needed
- No guarantee that algorithm would learn nonterminals that are linguistically motivated

Improvements

- Lexicalization: adding word information to nonterminal
  Eg. Prob of VP expanding as V NP should depend on the choice of verb
- Markovization: adding local tree structure to nonterminal
- Use CRF-like features or NN for probabilities
- Other grammars

References

- Foundations of Statistical Natural Language Processing by Manning and Schütze (1999), Ch. 11-12
- Computational Linguistics course by Levy (2015), Lec. 20-22
- Natural Language Models and Interfaces course, Part B by Titov (2012)