NLP Programming Tutorial 8 - Phrase Structure Parsing

Graham Neubig
Nara Institute of Science and Technology (NAIST)
Interpreting Language is Hard!

I saw a girl with a telescope

- "Parsing" resolves structural ambiguity in a formal way
Two Types of Parsing

- **Dependency**: focuses on relations between words

  I saw a girl with a telescope

- **Phrase structure**: focuses on identifying phrases and their recursive structure

  I saw a girl with a telescope
Recursive Structure?

I saw a girl with a telescope
Recursive Structure?

I saw a girl with a telescope.
Recursive Structure?

I saw a girl with a telescope.
Recursive Structure?

I saw a girl with a telescope.
Recursive Structure?

I saw a girl with a telescope.
Different Structure, Different Interpretation

I saw a girl with a telescope
Different Structure, Different Interpretation

I saw a girl with a telescope

- **S**
  - **VP**
  - **NP**
    - **PP**
      - **NP**
        - **IN**
          - **DT**
            - **NN**
          - **NP**
            - **IN**
              - **DT**
                - **NN**
Different Structure, Different Interpretation

I saw a girl with a telescope

NP

S

NP

VP

NP

PP

NP

NP

IN

DT

NN

IN

DT

NN

with a telescope
Different Structure, Different Interpretation

I saw a girl with a telescope

PRP VBD DT NN IN DT NN NP VP S NP NP

with a telescope
Non-Terminals, Pre-Terminals, Terminals

I saw a girl with a telescope

I (PRP) saw (VBD) a (DT) girl (NN) with (IN) a (DT) telescope (NN)
Parsing as a Prediction Problem

- Given a sentence \( X \), predict its parse tree \( Y \)

```
I saw a girl with a telescope
```

- A type of “structured” prediction (similar to POS tagging, word segmentation, etc.)
Probabilistic Model for Parsing

- Given a sentence $X$, predict the most probable parse tree $Y$

$$\text{argmax}_Y P(Y|X)$$
Probabilistic Generative Model

- We assume some probabilistic model generated the parse tree $Y$ and sentence $X$ jointly

$$P(Y, X)$$

- The parse tree with highest joint probability given $X$ also has the highest conditional probability

$$\arg\max_Y P(Y|X) = \arg\max_Y P(Y, X)$$
Probabilistic Context Free Grammar (PCFG)

- How do we define a joint probability for a parse tree?

\[
P(\quad)
\]

I saw a girl with a telescope
Probabilistic Context Free Grammar (PCFG)

- PCFG: Define probability for each node

I saw a girl with a telescope

P(S → NP VP) → S

P(PRP → “I”) → NP

P(NP → DT NN) → NP

P(NP → “telescope”) → NN

P(PP → IN NP) → PP

P(VP → VBD NP PP) → VP

P(S → NP VP) → S

I saw a girl with a telescope
Probabilistic Context Free Grammar (PCFG)

- PCFG: Define probability for each node

\[
P(S \rightarrow NP \ VP) \quad S
\]

\[
P(PRPR \rightarrow "I") \quad NP
\]

\[
P(VP \rightarrow VBD \ NP \ PP) \quad VP
\]

\[
P(NP \rightarrow DT \ NN) \quad NP
\]

\[
P(NP \rightarrow PRP) \quad P(PRPR \rightarrow "I")
\]

\[
P(VP \rightarrow VBD \ NP \ PP) \quad P(VP \rightarrow VBD \ NP \ PP)
\]

\[
P(VBD \rightarrow "saw") \quad P(VBD \rightarrow "saw")
\]

\[
P(DT \rightarrow "a") \quad P(DT \rightarrow "a")
\]

\[
P(NN \rightarrow "telescope") \quad P(NN \rightarrow "telescope")
\]

I saw a girl with a telescope

- Parse tree probability is product of node probabilities

\[
P(S \rightarrow NP \ VP) \times P(NP \rightarrow PRP) \times P(PRPR \rightarrow "I")
\]

\[
\times P(VP \rightarrow VBD \ NP \ PP) \times P(VBD \rightarrow "saw") \times P(NP \rightarrow DT \ NN)
\]

\[
\times P(DT \rightarrow "a") \times P(NN \rightarrow "girl") \times P(PP \rightarrow IN \ NP) \times P(IN \rightarrow "with")
\]

\[
\times P(NP \rightarrow DT \ NN) \times P(DT \rightarrow "a") \times P(NN \rightarrow "telescope")
\]
Probabilistic Parsing

• Given this model, parsing is the algorithm to find

$$\arg\max_Y P(Y, X)$$

• Can we use the Viterbi algorithm as we did before?
Probabilistic Parsing

- Given this model, parsing is the algorithm to find

$$\underset{Y}{\text{argmax}} \ P(Y, X)$$

- Can we use the Viterbi algorithm as we did before?
  - Answer: No!
  - Reason: Parse candidates are not graphs, but hypergraphs.
What is a Hypergraph?

- Let's say we have two parse trees

I saw a girl with a telescope
What is a Hypergraph?

• Most parts are the same!
What is a Hypergraph?

- Graph with all same edges + all nodes
What is a Hypergraph?

- Create graph with all same edges + all nodes

```
I saw a girl with a telescope
```

```
S
  VP
    NP
      PRP VBD
        DT NN IN DT NN
          0,1 1,2 2,3 3,4 4,5 5,6 6,7
```

```
NP
  PP
    NP
      PRP VBD DT NN IN DT NN
        0,1 1,2 2,3 3,4 4,5 5,6 6,7
```

What is a Hypergraph?

- With the edges in the **first** trees:

```
S 0,7
   VP 1,7
   NP 2,7
      NP 2,4
         PRP 0,1
         VBD 1,2
          DT 2,3
          NN 3,4
          IN 4,5
          DT 5,6
          NN 6,7

I saw a girl with a telescope
```
What is a Hypergraph?

- With the edges in the second tree:
What is a Hypergraph?

- With the edges in the **first** and **second** trees:

Two choices!
Choose **red**, get the **first** tree
Choose **blue**, get the **second** tree
Why a “Hyper” graph?

- The “degree” of an edge is the number of children.

<table>
<thead>
<tr>
<th>Degree 1</th>
<th>Degree 2</th>
<th>Degree 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRP 0,1</td>
<td>VBD 1,2</td>
<td>VP 1,7</td>
</tr>
<tr>
<td>I</td>
<td>saw</td>
<td>VBD 1,2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NP 2,7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The degree of a hypergraph is the maximum degree of all its edges.

- A graph is a hypergraph of degree 1!

Example →

![Diagram with nodes and edges labeled with numbers]
Weighted Hypergraphs

• Like graphs:
  • can add weights to hypergraph edges
  • use negative log probability of rule

-\log(P(S \rightarrow NP \ VP))

-\log(P(VP \rightarrow VBD \ NP \ PP))

-\log(P(VP \rightarrow VBD \ NP))

log(P(PR \rightarrow "I"))
Solving Hypergraphs

- Parsing = finding minimum path through a hypergraph
Solving Hypergraphs

• Parsing = finding minimum path through a hypergraph
• We can do this for graphs with the Viterbi algorithm
  • **Forward**: Calculate score of best path to each state
  • **Backward**: Recover the best path
Solving Hypergraphs

- Parsing = finding minimum path through a hypergraph
- We can do this for graphs with the Viterbi algorithm
  - Forward: Calculate score of best path to each state
  - Backward: Recover the best path
- For hypergraphs, almost identical algorithm!
  - Inside: Calculate score of best subtree for each node
  - Outside: Recover the best tree
**Review: Viterbi Algorithm (Forward Step)**

```
best_score[0] = 0
for each node in the graph (ascending order)
    best_score[node] = ∞
    for each incoming edge of node
        score = best_score[edge.prev_node] + edge.score
        if score < best_score[node]
            best_score[node] = score
            best_edge[node] = edge
```
Example:

Initialize:
best_score[0] = 0
Example:

Initialize:
best_score[0] = 0

Check $e_1$:
score = 0 + 2.5 = 2.5 (< ∞)
best_score[1] = 2.5
best_edge[1] = $e_1$
Example:

Initialize:

\[ \text{best_score}[0] = 0 \]

Check \( e_1 \):

\[ \text{score} = 0 + 2.5 = 2.5 \quad (< \infty) \]
\[ \text{best_score}[1] = 2.5 \]
\[ \text{best_edge}[1] = e_1 \]

Check \( e_2 \):

\[ \text{score} = 0 + 1.4 = 1.4 \quad (< \infty) \]
\[ \text{best_score}[2] = 1.4 \]
\[ \text{best_edge}[2] = e_2 \]
Example:

Initialize:
best_score[0] = 0

Check e₁:
score = 0 + 2.5 = 2.5 (< ∞)
best_score[1] = 2.5
best_edge[1] = e₁

Check e₂:
score = 0 + 1.4 = 1.4 (< ∞)
best_score[2] = 1.4
best_edge[2] = e₂

Check e₃:
score = 2.5 + 4.0 = 6.5 (> 1.4)
No change!
Example:

Initialize:
best_score[0] = 0

Check \( e_1 \):
\[
\text{score} = 0 + 2.5 = 2.5 (< \infty)
\]
best_score[1] = 2.5
best_edge[1] = \( e_1 \)

Check \( e_2 \):
\[
\text{score} = 0 + 1.4 = 1.4 (< \infty)
\]
best_score[2] = 1.4
best_edge[2] = \( e_2 \)

Check \( e_3 \):
\[
\text{score} = 2.5 + 4.0 = 6.5 (> 1.4)
\]
No change!

Check \( e_4 \):
\[
\text{score} = 2.5 + 2.1 = 4.6 (< \infty)
\]
best_score[3] = 4.6
best_edge[3] = \( e_4 \)
Example:

Initialize:
best_score[0] = 0

Check $e_1$:
\[
\text{score} = 0 + 2.5 = 2.5 (< \infty)
\]
best_score[1] = 2.5
best_edge[1] = $e_1$

Check $e_2$:
\[
\text{score} = 0 + 1.4 = 1.4 (< \infty)
\]
best_score[2] = 1.4
best_edge[2] = $e_2$

Check $e_3$:
\[
\text{score} = 2.5 + 4.0 = 6.5 (> 1.4)
\]
No change!

Check $e_4$:
\[
\text{score} = 2.5 + 2.1 = 4.6 (< \infty)
\]
best_score[3] = 4.6
best_edge[3] = $e_4$

Check $e_5$:
\[
\text{score} = 1.4 + 2.3 = 3.7 (< 4.6)
\]
best_score[3] = 3.7
best_edge[3] = $e_5$
Result of Forward Step

best_score = ( 0.0, 2.5, 1.4, 3.7 )

best_edge = ( NULL, e_1, e_2, e_5 )
Review: Viterbi Algorithm (Backward Step)

```
best_path = [ ]
next_edge = best_edge[best_edge.length – 1]
while next_edge != NULL
    add next_edge to best_path
    next_edge = best_edge[next_edge.prev_node]
reverse best_path
```
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e₅
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e_{5}

Process e_{5}:
best_path = [e_{5}]
next_edge = best_edge[2] = e_{2}
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e₅

Process e₂:
best_path = [e₅, e₂]
next_edge = best_edge[0] = NULL

Process e₅:
best_path = [e₅]
next_edge = best_edge[2] = e₂
Example of Backward Step

Initialize:
best_path = []
next_edge = best_edge[3] = e_5

Process \( e_5 \):
best_path = [e_5, e_2]
next_edge = best_edge[0] = NULL

Reverse:
best_path = [e_2, e_5]
Inside Step for Hypergraphs:

- Find the score of best subtree of VP1,7
Inside Step for Hypergraphs:

- Find the score of best subtree of VP1,7

\[
\text{score}(e_1) = -\log(P(VP \rightarrow VBD \ NP \ PP)) + \text{best_score}[VBD1,2] + \text{best_score}[NP2,4] + \text{best_score}[NP2,7]
\]

\[
\text{score}(e_2) = -\log(P(VP \rightarrow VBD \ NP)) + \text{best_score}[VBD1,2] + \text{best_score}[VBD2,7]
\]
Inside Step for Hypergraphs:

- Find the score of best subtree of VP1,7

\[
\text{score}(e_1) = -\log(P(VP \rightarrow VBD \ NP \ PP)) + \text{best_score}[VBD1,2] + \text{best_score}[NP2,4] + \text{best_score}[NP2,7]
\]

\[
\text{score}(e_2) = -\log(P(VP \rightarrow VBD \ NP)) + \text{best_score}[VBD1,2] + \text{best_score}[VBD2,7]
\]

best_edge[VB1,7] = argmin_{e_1,e_2} score
Inside Step for Hypergraphs:

- Find the score of best subtree of VP1,7

\[
\begin{align*}
\text{score}(e_1) &= -\log(P(\text{VP} \rightarrow \text{VBD NP PP})) + \\
&\quad \text{best}\_\text{score}[\text{VBD1,2}] + \\
&\quad \text{best}\_\text{score}[\text{NP2,4}] + \\
&\quad \text{best}\_\text{score}[\text{NP2,7}]
\end{align*}
\]

\[
\begin{align*}
\text{score}(e_2) &= -\log(P(\text{VP} \rightarrow \text{VBD NP})) + \\
&\quad \text{best}\_\text{score}[\text{VBD1,2}] + \\
&\quad \text{best}\_\text{score}[\text{VBD2,7}]
\end{align*}
\]

\[
\text{best}\_\text{edge}[\text{VB1,7}] = \arg\min_{e_1,e_2} \text{score}
\]

\[
\text{best}\_\text{score}[\text{VB1,7}] = \text{score}(\text{best}\_\text{edge}[\text{VB1,7}])
\]
Building Hypergraphs from Grammars

- Ok, we can solve hypergraphs, but what we have is:

<table>
<thead>
<tr>
<th>A Grammar</th>
<th>A Sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>P(S \rightarrow NP VP) = 0.8</td>
<td></td>
</tr>
<tr>
<td>P(S \rightarrow PRP VP) = 0.2</td>
<td></td>
</tr>
<tr>
<td>P(VP \rightarrow VBD NP PP) = 0.6</td>
<td>I saw a girl with a telescope</td>
</tr>
<tr>
<td>P(VP \rightarrow VBD NP) = 0.4</td>
<td></td>
</tr>
<tr>
<td>P(NP \rightarrow DT NN) = 0.5</td>
<td></td>
</tr>
<tr>
<td>P(NP \rightarrow NN) = 0.5</td>
<td></td>
</tr>
<tr>
<td>P(PRP \rightarrow “I”) = 0.4</td>
<td></td>
</tr>
<tr>
<td>P(VBD \rightarrow “saw”) = 0.05</td>
<td></td>
</tr>
<tr>
<td>P(DT \rightarrow “a”) = 0.6</td>
<td></td>
</tr>
</tbody>
</table>

- How do we build a hypergraph?
CKY Algorithm

• The CKY (Cocke-Kasami-Younger) algorithm creates and solves hypergraphs

• Grammar must be in Chomsky normal form (CNF)
  • All rules have two non-terminals or one terminal on right

<table>
<thead>
<tr>
<th>OK</th>
<th>OK</th>
<th>Not OK!</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>PRP → “I”</td>
<td>VP → VBD NP PP</td>
</tr>
<tr>
<td>S → PRP VP</td>
<td>VBD → “saw”</td>
<td>VP' → NP PP</td>
</tr>
<tr>
<td>VP → VBD NP</td>
<td>DT → “a”</td>
<td>NP → PRP</td>
</tr>
</tbody>
</table>

• Can convert rules into CNF

<table>
<thead>
<tr>
<th>VP → VBD NP PP</th>
<th>VP → VBD VP'</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP' → NP PP</td>
<td>NP_PRP → “I”</td>
</tr>
</tbody>
</table>
CKY Algorithm

- Start by expanding all rules for terminals with scores
CKY Algorithm

- Expand all possible nodes for 0,2

\[
0.5 + 3.2 + 1.0 = 4.7
\]

\[
S_{0,2}
\]

\[
SBAR_{0,2}
\]

\[
5.3
\]

\[
PRP_{0,1}
\]

\[
NP_{0,1}
\]

\[
0.5
\]

\[
VBD_{1,2}
\]

\[
VBP_{1,2}
\]

\[
1.4
\]

\[
PRP_{2,3}
\]

\[
NP_{2,3}
\]

\[
2.6
\]

\[
l
\]

\[
saw
\]

\[
him
\]
CKY Algorithm

- Expand all possible nodes for 1,3
CKY Algorithm

- Expand all possible nodes for 0,3
CKY Algorithm

- Find the S that covers the entire sentence and its best edge
CKY Algorithm

- Expand the left child, right child recursively until we have our tree
CKY Algorithm

- Expand the left child, right child recursively until we have our tree
CKY Algorithm

- Expand the left child, right child recursively until we have our tree
CKY Algorithm

- Expand the left child, right child recursively until we have our tree
Printing Parse Trees

- Standard text format for parse tree: “Penn Treebank”

```
(PP
  (IN with)
  (NP
    (DT a)
    (NN telescope)))
```

```
with a telescope
```

```
IN   DT   NN
  ↓   ↓   ↓
  PP  NP
```

IN

"with a telescope"
Printing Parse Trees

- Hypergraphs printed recursively, starting at top:

```
PRINT(S₀,₇) = “(S “ + PRINT(NP₀,₁) + “ “ + print(VP₁,₇)+”)
PRINT(NP₀,₁) = “(NP “ + PRINT(PRP₀,₁) + “)”
PRINT(PRP₀,₁) = “(PRP I)”
```

...
Pseudo-Code
CKY Pseudo-Code: Read Grammar

# Read a grammar in format “lhs \t rhs \t prob \n”
make list nonterm # Make list of (lhs, rhs1, rhs2, prob)
make map preterm # Make a map preterm[rhs] = [ (lhs, prob) ...]
for rule in grammar_file
    split rule into lhs, rhs, prob (with “\t”) # Rule P(lhs → rhs)=prob
    split rhs into rhs_symbols (with “ “)
    if length(rhs) == 1: # If this is a pre-terminal
        add (lhs, log(prob)) to preterm[rhs]
    else: # Otherwise, it is a non-terminal
        add (lhs, rhs[0], rhs[1], log(prob)) to nonterm
CKY Pseudo-Code: Add Pre-Terminals

split line into words
make map best_score # index: sym_{i,j} value = best log prob
make map best_edge # index: sym_{i,j} value = (lsym_{i,k}, rsym_{k,j})
# Add the pre-terminal sym
for i in 0 .. length(words)-1:
    for lhs, log_prob in preterm where P(lhs \rightarrow words[i]) > 0:
        best_score[lhs_{i,i+1}] = [log_prob]
CKY Pseudo-Code:

Combine Non-Terminals

for $j$ in 2 .. length(words):  # $j$ is right side of the span
  for $i$ in $j-2 .. 0$:  # $i$ is left side (Note: Reverse order!)
    for $k$ in $i+1 .. j-1$:  # $k$ is beginning of the second child
      # Try every grammar rule
      log(P(sym → lsym rsym)) = logprob
      for sym, lsym, rsym, logprob in nonterm:
        # Both children must have a probability
        if $best\_score[lsym_{i,k}] > -\infty$ and $best\_score[rsym_{k,j}] > -\infty$:
          # Find the log probability for this node/edge
          $my\_lp = best\_score[lsym_{i,k}] + best\_score[rsym_{k,j}] + logprob$
          # If this is the best edge, update
          if $my\_lp > best\_score[sym_{i,j}]$:
            $best\_score[sym_{i,j}] = my\_lp$
            $best\_edge[sym_{i,j}] = (lsym_{i,k}, rsym_{k,j})$
CKY Pseudo-Code: Print Tree

\[
\text{PRINT}(S_{0,\text{length(words)}}) \ # \ Print \ the \ "S" \ that \ spans \ all \ words
\]

\[
\text{subroutine PRINT(sym}_{i,j}):\n\]

\[
\text{if sym}_{i,j} \ \text{exists in best_edge}: \ # \ for \ non-terminals
\]

\[
\text{return } \"(+sym+" "
+ \text{PRINT(best_edge}[0]) + " " +
+ \text{PRINT(best_edge}[1]) + ")"
\]

\[
\text{else: } \ # \ for \ terminals
\]

\[
\text{return } \"(+sym+" "+words[i]+"")"
\]
Exercise
Exercise

- **Write** cky.py
- **Test** the program
  - Input: `test/08-input.txt`
  - Grammar: `test/08-grammar.txt`
  - Answer: `test/08-output.txt`
- **Run the program on actual data:**
- **Visualize** the trees
  - `script/print-trees.py < wiki-en-test.trees`
  - *(Requires NLTK: http://nltk.org/)*
- **Challenge**: think of a way to handle unknown words
Thank You!