CS 533: Natural Language Processing

Constituency Parsing

Karl Stratos

Rutgers University
Project Logistics

1. Proposal (due 3/24)

2. Milestone (due 4/15)

3. Presentation (tentatively 4/29)

4. Final report (due 5/4)
Possible Project Types (More Details in Template)

- Extend and apply recent machine learning methods to previously unconsidered NLP tasks
  - Search last/this year’s AISTATS/ICLR/ICML/NeurIPS/UAI publications

- Extend a recent machine learning method in NLP
  - Search last/this year’s ACL/CoNLL/EMNLP/NAACL

- Reimplement and replicate an existing technically challenging NLP paper from scratch
  - No available public code!

- There may be a limited number of predefined projects
  - No promise
  - Priority given to those with higher assignment grades
Course Status

Covered (all in the context of neural networks)
- Language models and conditional language models
- Pretrained representations from language modeling, evaluation tasks (GLUE, SuperGLUE)
- Tagging, parsing (today)

Will cover
- Latent-variable models (EM, VAEs)
- Information extraction (entity linking)

before the proposal due date

You will have enough background to read state-of-the-art research papers for your proposal.
Recall HMM

\[
p(x_1 \ldots x_n, y_1 \ldots y_n) = \prod_{i=1}^{n+1} t(y_i|y_{i-1}) \times \prod_{i=1}^{n} o(x_i|y_i)
\]

The forward algorithm computes in \(O(|\mathcal{Y}|^2 n)\)

\[
\alpha(i, y) = \sum_{y_1 \ldots y_i \in \mathcal{Y}^i: y_i=y} p(x_1 \ldots x_i, y_1 \ldots y_i)
\]

The backward algorithm computes in \(O(|\mathcal{Y}|^2 n)\)

\[
\beta(i, y) = \sum_{y_i \ldots y_n \in \mathcal{Y}^{n-i+1}: y_i=y} p(x_{i+1} \ldots x_n, y_{i+1} \ldots y_n|y_i)
\]
Backward Algorithm

**Base case.** For $y \in \mathcal{Y}$,

$$\beta(n, y) = t(\text{STOP}|y)$$

**Main.** For $i = n - 1 \ldots 1$, for $y \in \mathcal{Y}$,

$$\beta(i, y) = \sum_{y' \in \mathcal{Y}} t(y'|y) \times o(x_{i+1}|y') \times \beta(i+1, y')$$
Once forward and backward probabilities are computed, useful marginal probabilities can be computed.

Debugging tip: \( p(x_1 \ldots x_n) = \sum_y \alpha(i, y) \beta(i, y) \) for any \( i \)

Marginal decoding: for \( i = 1 \ldots n \) predict \( y \in \mathcal{Y} \) with highest

\[
p(x_1 \ldots x_n, y_i = y) = \alpha(i, y) \times \beta(i, y)
\]

Tag pair conditional marginals (will be useful later)

\[
p(y_i = y, y_{i+1} = y' | x_1 \ldots x_n) = \frac{\alpha(i, y) \times t(y' | y) \times o(x_{i+1} | y') \times \beta(i + 1, y')}{p(x_1 \ldots x_n)}
\]
Agenda

1. Parsing

2. PCFG: A model for constituency parsing

3. Transition-based dependency parsing

(Some slides adapted from Chris Manning, Mike Collins)
Constituency Parsing vs Dependency Parsing

Two formalisms for linguistic structure
Constituency Structure

Nested constituents/phrases

- Start by labeling words with POS tags
  
  (D the) (N dog) (V saw) (D the) (N cat)

- Recursively combine constituents according to rules
  
  (NP (D the) (N dog)) (V saw) (D the) (N cat)
  
  (NP (D the) (N dog)) (V saw) (NP (D the) (N cat))
  
  (NP (D the) (N dog)) (VP (V saw) (NP (D the) (N cat)))
  
  (S (NP (D the) (N dog)) (VP (V saw) (NP (D the) (N cat))))

Used rules: NP → D N, VP → V NP, S → NP VP
Dependency Structure

Labeled pairwise relations between words

Satellites spot whales from space

nsubj nmod
dobj case
Case for Parsing

- Most compelling example of latent structure in language

- Hypothesis: parsing is intimately related to intelligence

- Also some useful applications, e.g., relation extraction
Context-Free Grammar (CFG)

A tuple $G = (N, \Sigma, R, S)$

- $N$: non-terminal symbols (constituents)
- $\Sigma$: terminal symbols (words)
- $R$: rules of form $X \rightarrow Y_1 \ldots Y_m$ where $X \in N, Y_i \in N \cup \Sigma$
- $S \in N$: start symbol
Example CFG

\[ N = \{S, \text{NP}, \text{VP}, \text{PP}, \text{DT}, \text{Vi}, \text{Vt}, \text{NN}, \text{IN}\} \]
\[ S = S \]
\[ \Sigma = \{\text{sleeps, saw, man, woman, telescope, the, with, in}\} \]

\[ R = \]

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Lexicon</th>
</tr>
</thead>
</table>
| \( S \rightarrow \text{NP} \text{ VP} \) | \( \text{Vi} \rightarrow \text{sleeps} \)
| \( \text{VP} \rightarrow \text{Vi} \)       | \( \text{Vt} \rightarrow \text{saw} \)
| \( \text{VP} \rightarrow \text{Vt} \text{ NP} \) | \( \text{NN} \rightarrow \text{man} \)
| \( \text{VP} \rightarrow \text{VP} \text{ PP} \) | \( \text{NN} \rightarrow \text{woman} \)
| \( \text{NP} \rightarrow \text{DT} \text{ NN} \) | \( \text{NN} \rightarrow \text{telescope} \)
| \( \text{NP} \rightarrow \text{NP} \text{ PP} \) | \( \text{NN} \rightarrow \text{dog} \)
| \( \text{PP} \rightarrow \text{IN} \text{ NP} \) | \( \text{DT} \rightarrow \text{the} \)
|                  | \( \text{IN} \rightarrow \text{with} \)
|                  | \( \text{IN} \rightarrow \text{in} \)  

Given a CFG $G = (N, \Sigma, R, S)$, a left-most derivation is a sequence of strings $s_1 \ldots s_n$ where

- $s_1 = S$
- For $i = 2 \ldots n$, $s_i = \text{ExpandLeftMostNonterminal}(s_{i-1})$
- $s_n \in \Sigma^*$ (aka “yield” of the derivation)
Example

- $s_1 = S$
- $s_2 = NP \ VP$
- $s_3 = DT \ NN \ VP$
- $s_4 = the \ NN \ VP$
- $s_5 = the \ man \ VP$
- $s_6 = the \ man \ Vi$
- $s_7 = the \ man \ sleeps$

A derivation can be represented as a parse tree!

- A string $s \in \Sigma^*$ is in the language defined by the CFG if there is at least one derivation whose yield is $s$
- The set of possible derivations may be finite or infinite
Ambiguity

Some string can have multiple valid derivations (i.e., parse trees).

Number of binary trees over $n + 1$ nodes ($n$-th Catalyn number)

$$C_n = \frac{1}{n+1} \binom{2n}{n} > 6.5 \text{ billion for } n = 20$$
Rule-Based to Statistical

- Rule-based: manually construct some CFG that recognizes as many English strings as possible
  - Never enough, no way to choose the right parse

- Statistical: annotate sentences with parse trees (aka. treebank) and learn a statistical model to disambiguate
Treebanks

- Standard setup: WSJ portion of Penn Treebank
  - 40,000 trees for training
  - 1,700 trees for validation
  - 2,400 trees for evaluation

- Building a treebank vs building a grammar?
  - Broad coverage, more natural annotation
  - Contains distributional information of English
  - Can be used to evaluate parsers
A tuple $G = (N, \Sigma, R, S, q)$

- $N$: non-terminal symbols (constituents)
- $\Sigma$: terminal symbols (words)
- $R$: rules of form $X \rightarrow Y_1 \ldots Y_m$ where $X \in N, Y_i \in N \cup \Sigma$
- $S \in N$: start symbol
- $q$: rule probability $q(\alpha \rightarrow \beta) \geq 0$ for every rule $\alpha \rightarrow \beta \in R$ such that $\sum_{\beta} q(X \rightarrow \beta) = 1$ for any $X \in N$

<table>
<thead>
<tr>
<th>$S$</th>
<th>$NP$</th>
<th>$VP$</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$VP$</td>
<td>$Vi$</td>
<td></td>
<td>0.4</td>
</tr>
<tr>
<td>$VP$</td>
<td>$Vt$</td>
<td>$NP$</td>
<td>0.4</td>
</tr>
<tr>
<td>$VP$</td>
<td>$VP$</td>
<td>$PP$</td>
<td>0.2</td>
</tr>
<tr>
<td>$NP$</td>
<td>$DT$</td>
<td>$NN$</td>
<td>0.3</td>
</tr>
<tr>
<td>$NP$</td>
<td>$NP$</td>
<td>$PP$</td>
<td>0.7</td>
</tr>
<tr>
<td>$PP$</td>
<td>$P$</td>
<td>$NP$</td>
<td>1.0</td>
</tr>
<tr>
<td>$Vi$</td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>$Vt$</td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>$NN$</td>
<td></td>
<td></td>
<td>0.7</td>
</tr>
<tr>
<td>$NN$</td>
<td></td>
<td></td>
<td>0.2</td>
</tr>
<tr>
<td>$NN$</td>
<td></td>
<td></td>
<td>0.1</td>
</tr>
<tr>
<td>$DT$</td>
<td></td>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>$IN$</td>
<td></td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>$IN$</td>
<td></td>
<td></td>
<td>0.5</td>
</tr>
</tbody>
</table>
For any derivation (parse tree) containing rules: 
\[ \alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_l \rightarrow \beta_l, \] the probability of the parse is:
\[
\prod_{i=1}^{l} q(\alpha_i \rightarrow \beta_i)
\]

\[
P(t) = q(S \rightarrow NP \ VP) \times q(NP \rightarrow DT \ NN) \times q(DT \rightarrow \text{the}) \times q(NN \rightarrow \text{man}) \times q(VP \rightarrow Vi) \times q(Vi \rightarrow \text{sleeps})
\]

\[= 1.0 \times 0.3 \times 1.0 \times 0.7 \times 0.4 \times 1.0 = 0.084\]
Estimating a PCFG from a Treebank

Given trees $t^{(1)} \ldots t^{(N)}$ in the training data

- $N$: all non-terminal symbols (constituents) seen in the data
- $\Sigma$: all terminal symbols (words) seen in the data
- $R$: all rules seen in the data
- $S \in N$: special start symbol (if the data does not already have it, add it to every tree)
- $q$: MLE estimate

$q(\alpha \rightarrow \beta) = \frac{\text{count}(\alpha \rightarrow \beta)}{\sum_{\beta} \text{count}(\alpha \rightarrow \beta)}$

If we see $VP \rightarrow Vt \times NP$ 10 times and $VP$ 1000 times, then $q(VP \rightarrow Vt \times NP) = 0.01$
Aside: Improper PCFG

\[ A \rightarrow A \ A \]  \quad \text{with probability} \ \gamma

\[ A \rightarrow a \]  \quad \text{with probability} \ \ 1 - \gamma

**Lemma.** Define \( S_h = \sum_{t: \text{height}(t) \leq h} p(t) \). Let \( S^* = \lim_{h \to \infty} S_h \). Then \( S^* < 1 \) if \( \gamma > 0.5 \).

- Total probability of parses is less than one!

Fortunately, an MLE from a finite treebank is never improper (aka. “tight”) (Chi and Geman, 2015)

Marginalization and Inference

\( \text{GEN}(x_1 \ldots x_n) \) denotes the set of all valid derivations for \( x_1 \ldots x_n \) under the considered PCFG.

1. What is the probability of \( x_1 \ldots x_n \) under a PCFG?

\[
\sum_{t \in \text{GEN}(x_1 \ldots x_n)} p(t)
\]

2. What is the most likely tree of \( x_1 \ldots x_n \) under a PCFG?

\[
\arg \max_{t \in \text{GEN}(x_1 \ldots x_n)} p(t)
\]
Chomsky Normal Form (CNF)

From here on, always assume that a PCFG $G = (N, \Sigma, R, S, q)$ is in CNF: meaning every $\alpha \rightarrow \beta \in R$ is either

1. Binary non-terminal production: $X \rightarrow Y Z$ where $X, Y, Z \in N$

2. Unary terminal production: $X \rightarrow x$ where $X \in N, x \in \Sigma$

Not a big deal: can convert PCFG to equivalent CNF (and back)
Inside Algorithm: Bottom-Up Marginalization

For $1 \leq i \leq j \leq n$, for all $X \in N$,

$$\alpha(i, j, X) = \sum_{t \in \text{GEN}(x_i \ldots x_j): \text{root}(t) = X} p(t)$$

We will see that computing each $\alpha(i, j, X)$ takes $O(n |R|)$ time using dynamic programming.

Return

$$\alpha(1, n, S) = \sum_{t \in \text{GEN}(x_1 \ldots x_n)} p(t)$$

Total runtime?
Base Case \((i = j)\)

For \(i = 1 \ldots n\), for \(X \in N\),

\[
\alpha(i, i, X) = q(X \rightarrow x_i)
\]
Main Body \((j = i + l)\)

For \(l = 1 \ldots n - 1\), for \(i = 1 \ldots n - l\) (set \(j = i + l\)), for \(X \in N\),

\[
\alpha(i, j, X) = \sum_{i \leq k < j} q(X \rightarrow Y \ Z) \times \alpha(i, k, Y) \times \alpha(k + 1, j, Z)
\]
CKY Parsing Algorithm: Bottom-Up Maximization

- For $1 \leq i \leq j \leq n$, for all $X \in N$,

\[
\pi(i, j, X) = \max_{t \in \text{GEN}(x_i \ldots x_j) \colon \text{root}(t) = X} p(t)
\]

- Base: $\pi(i, j, X) = q(X \rightarrow x_i)$

- Main: $\pi(i, j, X) = \max_{i \leq k < j, X \rightarrow Y Z \in R} q(X \rightarrow Y Z) \times \pi(i, k, Y) \times \pi(k + 1, j, Z)$

- We have

\[
\pi(1, n, S) = \max_{t \in \text{GEN}(x_1 \ldots x_n)} p(t)
\]

The optimal tree can be retrieved by backtracking
CKY Backtracking

\[ b(i, j, X) = \arg \max_{i \leq k < j, X \to Y \ Z \in R} q(X \to Y \ Z) \times \pi(i, k, Y) \times \pi(k + 1, j, Z) \]
Computing Marginals Under PCFG

- Can we calculate something like
  \[ \mu(i, j, X) = \sum_{t \in \text{GEN}(x_1...x_n): \text{root}(t,i,j)=X} p(t) \]

- Yes, by combining the inside algorithm with the **outside** algorithm

  \[ \beta(i, j, X) = \sum_{t \in \text{OUT}(x_i...x_j): \text{foot}(t)=X} p(t) \]
Outside Algorithm: Top-Down Marginalization

**Base.** $\beta(1, n, S) = 1$ and $\beta(1, n, X) = 0$ for all $X \neq S$

**Main.** For $l = n - 2 \ldots 1$, for $i = 1 \ldots n - l$ (set $j = i + l$), for $X \in N$,

$$
\beta(i, j, X) = \sum_{j < k \leq n}^{\beta(i, k, Z) \times \alpha(j + 1, k, Y) \times q(Z \rightarrow X Y) +} \\
\sum_{1 \leq k < i}^{\beta(k, j, Z) \times \alpha(k, i - 1, Y) \times q(Z \rightarrow Y X)}
$$
Max Marginal Parsing

- Inside + outside: for $1 \leq i \leq j \leq n$, for all $X \in N$,

$$
\mu(i, j, X) = \sum_{t \in \text{GEN}(x_1...x_n) : \text{root}(t,i,j) = X} p(t) = \alpha(i, j, X) \times \beta(i, j, X)
$$

- New parsing objective: find max marginal parse

$$
t^* = \arg \max_{t \in \text{GEN}(x_1...x_n)} \sum_{(i,j,X) \in t} \mu(i, j, X)
$$

- Labeled recall algorithm $O(n^3 |N|)$ (Goodman, 1996)

$$
\gamma(i, j) = \max_X \mu(i, j, X) + \max_{i \leq k < j} \gamma(i, k) + \gamma(k + 1, j)
$$

- How many algorithms do we need for max marginal parsing??
Evaluating Parser Predictions

- **Precision**
  \[ p = \frac{\text{number of correctly predicted } (i, j, X)}{\text{number of predicted } (i, j, X)} \]

- **Recall**
  \[ r = \frac{\text{number of correctly predicted } (i, j, X)}{\text{number of ground-truth } (i, j, X)} \]

- **Labeled \( F_1 \)**
  \[ F_1 = \frac{2 \times p \times r}{p + r} \]

Can also consider unlabeled \( F_1 \)
Gold standard brackets: $S(0:11), NP(0:2), VP(2:9), VP(3:9), NP(4:6), PP(6:9), NP(7,9), NP(9:10)$

Candidate brackets: $S(0:11), NP(0:2), VP(2:10), VP(3:10), NP(4:6), PP(6:10), NP(7,10)$

Precision 3/7 (42.9%), recall 3/8 (37.5%), labeled $F_1 40$

- What is the tagging accuracy?
Lexicalized PCFGs

- PCFG: extremely strong conditional independence assumption

- Can consider lexicalizing the grammar (Collins, 2003)
Constituency Parsing Performance

Labeled precision/recall on PTB-WSJ

▶ Vanilla PCFG: 70.6% recall, 74.8% precision
▶ Lexicalized PCFG: 88.1% recall, 88.3% precision
▶ Neuralized constituency parsing (Kitaev and Klein, 2018): 94.9% recall, 95.4% precision

Neural encoding followed by max marginal decoding: no independence assumption (read the paper)