A survey of computational linguistics

Mark Johnson

April, 2004
Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars
- Parsing and neural nets
What is computational linguistics?

Computational linguistics studies the *computational processes* involved in *language production, comprehension and acquisition*.

- assumption that language is *inherently computational*

- scientific side:
  - modeling human performance (computational psycholinguistics)
  - understanding how it can be done at all

- technological applications:
  - information extraction (who did what to whom)
  - machine translation (translation by computer)
  - speech recognition
(Some of the) problems in modeling language

+ Language is a product of the human mind
  ⇒ any structure we observe is a product of the mind

– Language involves a *transduction between form and meaning*, but we don’t know much about the way meanings are represented

+/− We have (reasonable?) guesses about some of the computational processes involved in language

– We don’t know very much about the cognitive processes that language interacts with

– We know little about the anatomical layout of language in the brain

– We know little about neural networks that might support linguistic computations
Aspects of linguistic structure

- Phonetics: the (production and perception) of speech sounds
- Phonology: the organization and regularities of speech sounds
- Morphology: the structure and organization of words
- Syntax: the way words combine to form phrases and sentences
- Semantics: the way meaning is associated with sentences
- Pragmatics: how language can be used to do things

In general the further we get from speech, the less well we understand what’s going on!
Why care about syntactic structure?

- Anaphora: *he* refers to *the president*
- Predicate-argument structure: *the president* is *agent* of *eat*
- Discourse structure: second clause is *contrasted* with first

These all refer to *phrase structure* entities!

It’s also the simplest aspect of language that incorporates unboundedly recursive combinatorics!
Talk outline

- What is computational linguistics?
- **Probabilistic grammars**
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars
- Parsing and neural nets
A context-free grammar $G = (\mathcal{V}, S, s, \mathcal{R})$ consists of:

- $\mathcal{V}$, a finite set of **terminals** ($\mathcal{V}_0 = \{\text{Sam, Sasha, thinks, snores}\}$)
- $S$, a finite set of **non-terminals** disjoint from $\mathcal{V}$ ($\mathcal{S}_0 = \{S, \text{NP, VP, V}\}$)
- $\mathcal{R}$, a finite set of **productions** of the form $A \rightarrow X_1 \ldots X_n$, where $A \in S$ and each $X_i \in S \cup \mathcal{V}$
- $s \in S$ is called the **start symbol** ($s_0 = S$)

$G$ generates a tree $\psi$ iff

- The label of $\psi$’s root node is $s$
- For all **local trees** with parent $A$ and children $X_1 \ldots X_n$ in $\psi$ 
  $A \rightarrow X_1 \ldots X_n \in \mathcal{R}$

$G$ generates a string $w \in \mathcal{V}^*$ iff $w$ is the **terminal yield** of a tree generated by $G$

**Productions**

- $S \rightarrow \text{NP VP}$
- $\text{NP} \rightarrow \text{Sam}$
- $\text{NP} \rightarrow \text{Sasha}$
- $\text{VP} \rightarrow \text{V}$
- $\text{VP} \rightarrow \text{V S}$
- $\text{V} \rightarrow \text{thinks}$
- $\text{V} \rightarrow \text{snores}$
CFGs as “plugging” systems

“Pluggings”

- Goal: no unconnected “sockets” or “plugs”
- The *productions* specify available types of components
- In a *probabilistic* CFG each type of component has a “price”
Structural Ambiguity

\[ R_1 = \{ \text{VP} \to \text{V NP}, \text{VP} \to \text{VP PP}, \text{NP} \to \text{D N}, \text{N} \to \text{N PP}, \ldots \} \]

- CFGs can capture *structural ambiguity* in language.
- Ambiguity generally grows *exponentially* in the length of the string.
  - The number of ways of parenthesizing a string of length \( n \) is \( \text{Catalan}(n) \)
- Broad-coverage statistical grammars are astronomically ambiguous.
Probabilistic Context Free Grammars

A Probabilistic Context Free Grammar (PCFG) $G$ consists of

- a CFG $(\mathcal{V}, S, S, \mathcal{R})$ with no useless productions, and
- production probabilities $p(A \to \beta) = P(\beta | A)$ for each $A \to \beta \in \mathcal{R}$,
  the conditional probability of an $A$ expanding to $\beta$

A production $A \to \beta$ is useless iff it is not used in any terminating derivation, i.e., there are no derivations of the form

$S \Rightarrow^* \gamma A \delta \Rightarrow \gamma \beta \delta \Rightarrow^* w$ for any $\gamma, \delta \in (\mathcal{V} \cup \mathcal{S})^*$ and $w \in \mathcal{V}^*$.

If $r_1 \ldots r_n$ is a sequence of productions used to generate a tree $\psi$, then

$$P_G(\psi) = p(r_1) \ldots p(r_n)$$

$$= \prod_{r \in \mathcal{R}} p(r)^{f_r(\psi)}$$

where $f_r(\psi)$ is the number of times $r$ is used in deriving $\psi$

$\sum_\psi P_G(\psi) = 1$ if $p$ satisfies suitable constraints
Example PCFG

\[
\begin{align*}
1.0 & \quad S \rightarrow NP \ VP \\
0.75 & \quad NP \rightarrow George \\
0.6 & \quad V \rightarrow barks \\
1.0 & \quad VP \rightarrow V \\
0.25 & \quad NP \rightarrow Al \\
0.4 & \quad V \rightarrow snores
\end{align*}
\]

\[
P\left(S \rightarrow NP\ VP \leftarrow\right) = 0.45 \\
P\left(S \rightarrow NP\ VP \leftarrow\right) = 0.1
\]
Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars
- Parsing and neural nets
String positions

String positions are a systematic way of representing substrings in a string.

A string position of a string \( w = x_1 \ldots x_n \) is an integer \( 0 \leq i \leq n \).

A substring of \( w \) is represented by a pair \( (i, j) \) of string positions, where \( 0 \leq i \leq j \leq n \).

\( w_{i,j} \) represents the substring \( w_{i+1} \ldots w_j \)

Example: \( w_{0,1} = \) Howard, \( w_{1,3} = \) likes mangoes, \( w_{1,1} = \epsilon \)

- Nothing depends on string positions being numbers, so
- this all generalizes to speech recognizer lattices, which are graphs where vertices correspond to word boundaries
Dynamic programming computation

Assume $G = (\mathcal{V}, \mathcal{S}, s, \mathcal{R}, p)$ is in \textit{Chomsky Normal Form}, i.e., all productions are of the form $A \rightarrow B C$ or $A \rightarrow x$, where $A, B, C \in \mathcal{S}$, $x \in \mathcal{V}$.

Goal: To compute $P(w) = \sum_{\psi \in \Psi_G(w)} P(\psi) = P(s \Rightarrow^* w)$

Data structure: A table $P(A \Rightarrow^* w_{i,j})$ for $A \in \mathcal{S}$ and $0 \leq i < j \leq n$

Base case: $P(A \Rightarrow^* w_{i-1,i}) = p(A \rightarrow w_{i-1,i})$ for $i = 1, \ldots, n$

Recursion: $P(A \Rightarrow^* w_{i,k})$

$$= \sum_{j=i+1}^{k-1} \sum_{A \rightarrow BC \in \mathcal{R}(A)} p(A \rightarrow BC)P(B \Rightarrow^* w_{i,j})P(C \Rightarrow^* w_{j,k})$$

Return: $P(s \Rightarrow^* w_{0,n})$
Dynamic programming recursion

\[ P_G(A \Rightarrow^* w_{i,k}) = \sum_{j=i+1}^{k-1} \sum_{A \rightarrow B C \in \mathcal{R}(A)} p(A \rightarrow B C) P_G(B \Rightarrow^* w_{i,j}) P_G(C \Rightarrow^* w_{j,k}) \]

\[ P_G(A \Rightarrow^* w_{i,k}) \] is called an “inside probability”.
Example PCFG parse

1.0 \( S \rightarrow NP \ VP \)

0.7 \( NP \rightarrow George \)

0.5 \( V \rightarrow likes \)

1.0 \( VP \rightarrow V \ NP \)

0.3 \( NP \rightarrow John \)

0.5 \( V \rightarrow hates \)
Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars
- Parsing and neural nets
Two approaches to computational linguistics

“Rationalist”: Linguist formulates generalizations and expresses them in a grammar

“Empiricist”: Collect a corpus of examples, linguists annotate them with relevant information, a machine learning algorithm extracts generalizations

- I don’t think there’s a deep philosophical difference here, but many people do

- Continuous models do much better than categorical models (statistical inference uses more information than categorical inference)

- Humans are lousy at estimating numerical probabilities, but luckily parameter estimation is the one kind of machine learning that (sort of) works
Treebanks, prop-banks and discourse banks

- A treebank is a corpus of phrase structure trees
  - The Penn treebank consists of about a million words from the Wall Street Journal, or about 40,000 trees.
  - The Switchboard corpus consists of about a million words of treebanked spontaneous conversations, linked up with the acoustic signal.
  - Treebanks are being constructed for other languages also

- The Penn treebank is being annotated with predicate argument structure (PropBank) and discourse relations.
Estimating PCFGs from visible data

<table>
<thead>
<tr>
<th>Rule</th>
<th>Count</th>
<th>Rel Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NP → rice</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>NP → corn</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>VP → grows</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
P \left( \begin{array}{c}
S \\
\downarrow \\
NP \\
\downarrow \\
rice \\
\downarrow \\
VP \\
\downarrow \\
grows
\end{array} \right) = \frac{2}{3}
\]

\[
P \left( \begin{array}{c}
S \\
\downarrow \\
NP \\
\downarrow \\
corn \\
\downarrow \\
VP \\
\downarrow \\
grows
\end{array} \right) = \frac{1}{3}
\]
Properties of MLE

- **Consistency**: As the sample size grows, the estimates of the parameters converge on the true parameters.

- **Asymptotic optimality**: For large samples, there is no other consistent estimator whose estimates have lower variance.

- The MLEs for statistical grammars work well in practice.
  - The Penn Treebank has $\approx 1.2$ million words of Wall Street Journal text annotated with syntactic trees.
  - The PCFG estimated from the Penn Treebank has $\approx 15,000$ rules.
Unsupervised training and EM

Expectation Maximization (EM) is a general technique for approximating the MLE when estimating parameters \( p \) from the visible data \( x \) is difficult, but estimating \( p \) from augmented data \( z = (x, y) \) is easier (\( y \) is the hidden data).

The EM algorithm given visible data \( x \):

1. guess initial value \( p_0 \) of parameters
2. repeat for \( i = 0, 1, \ldots \) until convergence:
   
   **Expectation step:** For all \( y_1, \ldots, y_n \in \mathcal{Y} \), generate pseudo-data \((x, y_1), \ldots, (x, y_n)\), where \((x, y_j)\) has frequency \( P_{p_i}(y_j|x)\)

   **Maximization step:** Set \( p_{i+1} \) to the MLE from the pseudo-data

The EM algorithm finds the MLE \( \hat{p}(x) = L_x(p) \) of the visible data \( x \).

Sometimes it is not necessary to explicitly generate the pseudo-data \((x, y)\); often it is possible to perform the maximization step directly from *sufficient statistics* (for PCFGs, the expected production frequencies)
Dynamic programming for $E_G[n_{A\rightarrow BC}|w]$ 

$$E_G[n_{A\rightarrow BC}|w] = \sum_{0 \leq i < j < k \leq n} E_G[A_{i,k} \rightarrow B_{i,j}C_{j,k}|w]$$

The expected fraction of parses of $w$ in which $A_{i,k}$ rewrites as $B_{i,j}C_{j,k}$ is:

$$E_G[A_{i,k} \rightarrow B_{i,j}C_{j,k}|w] = P(S \Rightarrow^* w_{1,i} Aw_{k,n}) P(A \rightarrow BC) P(B \Rightarrow^* w_{i,j}) P(C \Rightarrow^* w_{j,k}) / P_G(w)$$

[S
[A
[B
[C
[w_{0,i}
[w_{i,j}
[w_{j,k}
[w_{k,n}]]]]]]
Calculating $P_G(S \Rightarrow^* w_{0,i} A w_{k,n})$

Known as “outside probabilities” (but if $G$ contains unary productions, they can be greater than 1).

Recursion from *larger to smaller* substrings in $w$.

Base case: $P(S \Rightarrow^* w_{0,0} S w_{n,n}) = 1$

Recursion: $P(S \Rightarrow^* w_{0,j} C w_{k,n}) = $

$$
\sum_{i=0}^{j-1} \sum_{\substack{A,B \in S \\ A \rightarrow B \ C \in \mathcal{R}}} P(S \Rightarrow^* w_{0,i} A w_{k,n}) p(A \rightarrow B C) P(B \Rightarrow^* w_{i,j})
$$

$$
+ \sum_{l=k+1}^{n} \sum_{\substack{A,D \in S \\ A \rightarrow C \ D \in \mathcal{R}}} P(S \Rightarrow^* w_{0,j} A w_{l,n}) p(A \rightarrow C D) P(D \Rightarrow^* w_{k,l})
$$
Recursion in $P_G(S \Rightarrow^* w_{0,i} A w_{k,n})$

$$P(S \Rightarrow^* w_{0,j} C w_{k,n}) =$$

$$\sum_{i=0}^{j-1} \sum_{A,B \in S, A \rightarrow B, C \in \mathcal{R}} P(S \Rightarrow^* w_{0,i} A w_{k,n}) p(A \rightarrow B C) P(B \Rightarrow^* w_{i,j})$$

$$+ \sum_{l=k+1}^{n} \sum_{A,D \in S, A \rightarrow C, D \in \mathcal{R}} P(S \Rightarrow^* w_{0,j} A w_{l,n}) p(A \rightarrow C D) P(D \Rightarrow^* w_{k,l})$$
The EM algorithm for PCFGs

Infer *hidden structure* by maximizing likelihood of *visible data*:

1. guess initial rule probabilities
2. repeat until convergence
   (a) parse a sample of sentences
   (b) weight each parse by its conditional probability
   (c) count rules used in each weighted parse, and estimate rule frequencies from these counts as before

EM optimizes the *marginal likelihood of the strings* \( D = (w_1, \ldots, w_n) \)

Each iteration is *guaranteed* not to decrease the likelihood of \( D \), but EM can get trapped in local minima.

The *Inside-Outside algorithm* can produce the expected counts without enumerating all parses of \( D \).

When used with PFSA, the Inside-Outside algorithm is called the *Forward-Backward algorithm* (Inside=Backward, Outside=Forward)
Example: The EM algorithm with a toy PCFG

<table>
<thead>
<tr>
<th>Initial rule probs</th>
<th>&quot;English&quot; input</th>
<th>&quot;pseudo-Japanese&quot; input</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rule</strong></td>
<td><strong>prob</strong></td>
<td></td>
</tr>
<tr>
<td>VP → V</td>
<td>0.2</td>
<td>the dog bites</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.2</td>
<td>the dog bites a man</td>
</tr>
<tr>
<td>VP → NP V</td>
<td>0.2</td>
<td>a man gives the dog a bone</td>
</tr>
<tr>
<td>VP → V NP NP</td>
<td>0.2</td>
<td>...</td>
</tr>
<tr>
<td>VP → NP NP V</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Det → the</td>
<td>0.1</td>
<td>the dog bites</td>
</tr>
<tr>
<td>N → the</td>
<td>0.1</td>
<td>the dog a man bites</td>
</tr>
<tr>
<td>V → the</td>
<td>0.1</td>
<td>a man the dog a bone gives</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Rule probabilities from “English”

![Graph showing rule probabilities over iterations](image)
Probability of “Japanese”

Average sentence probability

Iteration

5 4 3 2 1 0

0.1

0.01

0.001

0.0001

1e-05

1e-06

0 1 2 3 4 5
Rule probabilities from "Japanese"
Learning in statistical paradigm

- The likelihood is a differentiable function of rule probabilities
  \[ \Rightarrow \text{learning can involve small, incremental updates} \]

- Learning new structure (rules) is hard, but . . .

- Parameter estimation can approximate rule learning
  - start with “superset” grammar
  - estimate rule probabilities
  - discard low probability rules
Applying EM to real data

- ATIS treebank consists of 1,300 hand-constructed parse trees
- ignore the words (in this experiment)
- about 1,000 PCFG rules are needed to build these trees
Experiments with EM

1. Extract productions from trees and estimate probabilities from trees to produce PCFG.

2. Initialize EM with the treebank grammar and MLE probabilities

3. Apply EM (to strings alone) to re-estimate production probabilities.

4. At each iteration:
   - Measure the likelihood of the training data and the quality of the parses produced by each grammar.
   - Test on training data (so poor performance is not due to overlearning).
Likelihood of training strings

\[- \log P_G(\bar{w})\]

![Graph showing the likelihood of training strings with iterations from 0 to 20, and the y-axis ranging from 14000 to 16000.](image)
Quality of ML parses

Parse Accuracy

Iteration

Precision
Recall

0.95
0.9
0.85
0.8
0.75
0.7

20 15 10 5 0
Why does EM do so poorly?

- EM assigns trees to strings to maximize the marginal probability of the strings, but the trees weren’t designed with that in mind.

- We have an “intended interpretation” of categories like NP, VP, etc., which EM has no way of knowing.

- Our grammar models are defective; real languages aren’t context-free.

- How can information about $P(w)$ provide information about $P(\psi|w)$?

- …but no one really knows.
Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars
- Parsing and neural nets
Subcategorization

Grammars that merely relate categories miss a lot of important linguistic relationships.

\[ R_3 = \{ \text{VP} \rightarrow \text{V}, \text{VP} \rightarrow \text{V NP}, \text{V} \rightarrow \text{sleeps}, \text{V} \rightarrow \text{likes}, \ldots \} \]

Verbs and other *heads of phrases* subcategorize for the number and kind of *complement phrases* they can appear with.
CFG account of subcategorization

General idea: *Split the preterminal states* to encode subcategorization.

\[ R_4 = \{ VP \rightarrow V, \ VP \rightarrow V_{\_NP} \ NP, \ V_{\_} \rightarrow \text{sleeps}, \ V_{\_NP} \rightarrow \text{likes} \} \]

The “split preterminal states” restrict which contexts verbs can appear in.

*Sparse data* becomes a big problem; addressed with *regularization* and *smoothing.*
Head-to-head dependencies are an approximation to real-world knowledge.

But note that selectional preferences involve more than head-to-head dependencies

Al drives a (#toy model) car
The number of possible rules grows rapidly; with a $\approx 10^4$ word vocabulary one might expect $\approx 10^8$ possible bilexical rules.
Subcategorization and selectional preferences are *preserved* under movement.

Movement can be encoded using *recursive* nonterminals (unification grammars).
Nonterminal labels as communication channels

The nonterminal label is the *communication channel* between two parts of the tree.

Learn or design the structure of non-terminals so they pass the appropriate information around the tree.

Modern statistical parsers pass around 7 different features through the tree, and condition productions on them.

*Sparse data* and therefore *smoothing* are major issues!
Talk outline

- What is computational linguistics?
- Probabilistic grammars
- Identifying phrase structure (parsing)
- Learning phrase structure
- More realistic grammars
- Parsing and neural nets
Applications of neural networks in CL

- The big problem: *how can NNs represent compositional structure?*
- NNs are sometimes used as *components* in larger models (e.g., predicting a node’s label given the labels of its children)
  - other machine learning techniques often work better, and are computationally more efficient
- Applying *recurrent NN models of structure in time* has generally not been successful (“a dog walking on its hind legs”)
  - Recurrent NNs seem to be equivalent to (factored) finite state machines
    ⇒ incapable of capturing *two-dimensional nature* of phrase structure
“Brute force” NN simulation of CKY parsing

\[ P_G(A \Rightarrow^* w_{i,k}) = \sum_{j=i+1}^{k-1} \sum_{A\rightarrow BC \in \mathcal{R}(A)} p(A \rightarrow BC) P_G(B \Rightarrow^* w_{i,j}) P_G(C \Rightarrow^* w_{j,k}) \]

- Introduce a node for each \( A_{i,k} \) and for each \((B_{i,j}, C_{j,k})\)
- \((B_{i,j}, C_{j,k})\) computes \( p(A \rightarrow BC) P_G(B \Rightarrow^* w_{i,j}) P_G(C \Rightarrow^* w_{j,k})\)
- \( A_{i,k} \) sums over all \( A \rightarrow BC \) and \((B_{i,j}, C_{j,k})\)
- Asynchronous (non-real-time) model with unnatural parameter tying
Incremental parsing algorithms

- Parsing involves incrementally constructing the parse tree
- We require that words are incorporated in left-to-right order
- Because of the two-dimensional nature of phrase structure trees, there are many ways of parsing incrementally
  - **Top-down**: Posit parent *before* any children
  - **Bottom-up**: Posit parent *after* all children
  - **Left-corner**: Posit parent *after first* child
- Every incremental parsing algorithm corresponds to different sets of *cuts through edges in the phrase structure tree*
- Only *the edges on such cuts are active*; the rest of the tree can be ignored (as far as parsing goes)
Top-down parsing is the only parsing algorithm that always maintains a connected partial parse tree.
Bottom-up parsing

```
the students know the professors snore
the students know the professors snore
... the students know the professors snore
```
Left-corner parsing
Incremental parsing and NNs

+ Only the small fraction of nodes in the parse tree lying on the cut are active at any point in the string
  – Can these be represented and updated by a recurrent NN? (Charniak)

  – These nodes are manipulated with a LIFO stack discipline
    – Unnatural in a NN architecture