

CS 4650/7650, Lecture 14: Dependency Parsing

Jacob Eisenstein

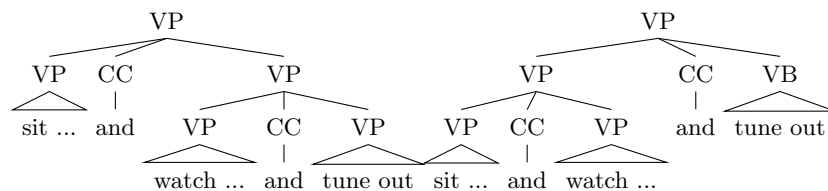
October 3, 2013

1 Homework 5

Here's my example:

I want to sit at a counter in a diner and watch the news and tune out...

- *in a diner* attaches to *counter*
- The two *ands* are coordinating the three verbs
- But there are two trees that seem equally reasonable...



- I couldn't find the answer in the PTB Bracketing Manual. FWIW, the Stanford Parser prefers the first one.
- What part-of-speech is *out*?

2 Review of PCFG Parsing

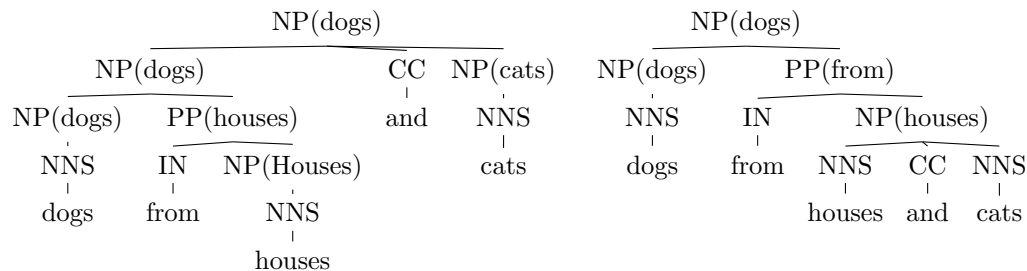
- PCFGs score the probability of a derivation $P(\tau, \mathbf{s})$ as the product of all productions in τ
- CKY...
 - CKY is a bottom-up algorithm for finding $\hat{\tau} = \arg \max_{\tau} P(\tau, \mathbf{s})$.
 - The **inside algorithm** finds $P(\mathbf{s}) = \sum_{\tau} P(\tau, \mathbf{s})$.
 - These algorithms are related through semiring notation.

- Let’s look at an example again.
- CKY is efficient, but it’s pretty implausible as a model of human parsing. **Shift-reduce** is a left-to-right parsing algorithm, which you may find more cognitively plausible.
 - It is related to the pushdown automata representation of context-free grammars: we move through the sentence while keeping a stack with infinite depth.
 - At each step, we have two choices
 - * **shift** the next word on to the stack
 - * **reduce** the stack by applying some production
 - If we can clear all the input and end up with just S on the stack, we have parsed the sentence correctly.
 - Each reduce move is a production in the derivation.
 - We might train a classifier to decide between shift and reduce.
- Regardless of the parsing algorithm, pure PCFG parsing on Penn Treebank nonterminals (NP, VP) doesn’t work well.
 - No flexibility on PP attachment, depends only on $P(\text{NP} \rightarrow \text{NP PP}) > P(\text{VP} \rightarrow \text{VP PP})$
 - No way to express semantic preferences or preference for right-branching or left-branching structures.
 - Rare productions get subsumed by common ones.
 - See slides for examples.
- Modern generative parsing algorithms automatically refine these nonterminals in various ways.
 - **Parent annotation**: label each non-terminal with its parent
 - **Lexicalization**: label each non-terminal with its head
 - We’ll talk about the details more after the midterm.

3 Lexicalization

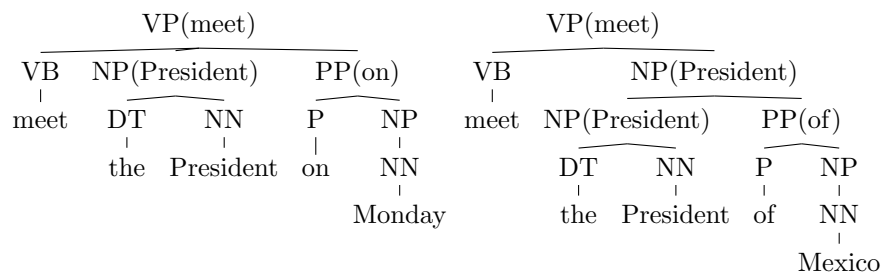
A simple way to capture semantics is through the words themselves. We can annotate each non-terminal with **head** word of the phrase.

3.1 Example: coordination scope



If $P(NP \rightarrow NP(\text{dogs}) \text{ CC } NP(\text{cats})) > (NP \rightarrow NP(\text{houses}) \text{ CC } NP(\text{cats}))$, we should get the right parse.

3.2 Example: PP attachment



- $P(VP(\text{meet}) \rightarrow \alpha PP(\text{on})) \gg P(NP(\text{President}) \rightarrow \beta PP(\text{on}))$
- $P(VP(\text{meet}) \rightarrow \alpha PP(\text{of})) \ll P(NP(\text{President}) \rightarrow \beta PP(\text{of}))$
- In plain English:
 - *Meeting* happens *on* things.
 - *Presidents* are *of* things.

3.3 Lexicalization was a major breakthrough

Vanilla PCFG	72%
Head-annotated PCFG (Johnson 1998)	80%
Lexicalized PCFG (Collins 1997, Charniak 1997)	87-89%

Eugene Charniak (2000): "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter."

One more example, subcategorization frames:

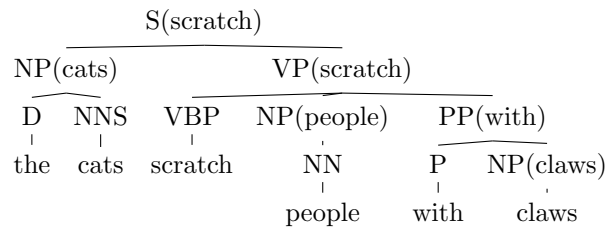
$$\begin{aligned} P(VP \rightarrow V \ NP \ NP) &= 0.00151 \\ P(VP(\text{said}) \rightarrow V(\text{said}) \ NP \ NP) &= 0.00001 \\ P(VP(\text{gave}) \rightarrow V(\text{gave}) \ NP \ NP) &= 0.01980 \end{aligned}$$

3.4 How to do it

- Naively: just augment the non-terminals to include the cross-product of all PTB non-terminals and all words.
- This will never work
 - Number of possible productions: $\mathcal{O}(N^3V^3)$, $V \approx 10^5$
 - Too slow, too sparse (total amount of PTB data = 10^6)
- Two practical algorithms: Charniak (1997) and Collins (1999). They each make independence assumptions to reduce the complexity.

4 Dependency parsing

Lexicalized parsing augments the non-terminals with **head words**.



- A set of deterministic **head percolation** rules determine how heads move up the tree through each production.
- Some rules are pretty obvious:
 - the head of a determiner-noun constituent is the noun:

$$\begin{array}{c} \text{NP(cats)} \\ \hline \text{D(the) \quad NP(cats)} \end{array}$$
 - prepositional adjuncts are not heads:

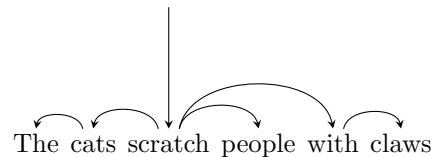
$$\begin{array}{c} \text{NP(people)} \\ \hline \text{NP(people) \quad PP(with claws)} \end{array}$$
 - verbal predicates are the heads of sentences:

$$\begin{array}{c} \text{S(scratch)} \\ \hline \text{NP(cats) \quad VP(scratch)} \end{array}$$
- Others are less clear-cut:

- the head of a conjunction is the left element:
$$\frac{\text{VP}(\text{cats})}{\text{VP}(\text{cats}) \quad \text{CC}(\text{and}) \quad \text{VP}(\text{dogs})}$$
- the head of a prepositional phrase is the preposition:
$$\frac{\text{PP}(\text{with})}{\text{P}(\text{with}) \quad \text{NP}(\text{claws})}$$
 (Alternatively, we can “collapse” out the preposition itself. This is the approach taken by the Stanford dependency parser.)

- We could argue about this stuff, but once we agree on a standard it can be applied deterministically to any parse tree.

A head-annotated parse tree defines a graph over the words in the sentence:



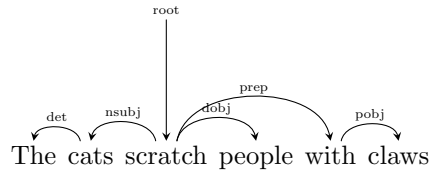
What are the properties of this graph?

- directed
- weakly connected
- every node has one incoming edge
- (therefore) no cycles
- (therefore) a tree

4.1 What is the meaning of the edges?

- A dependency edge means there is an asymmetric syntactic relationship between the head and the modifier.
- (sometimes called nucleus/satellite, or governor/dependent, or parent/child)
- Criteria for figuring out who is the head:
 - The modifier may be optional while the head is mandatory: (*red HAT*, *EAT quickly*)
 - The head sets the syntactic category of the construction: (prepositions are the heads of prepositional phrases)
 - The head determines the morphological form of the modifier: (*los LIBROS*, *una CASA*, *these HOUSES*)
- As always, these guidelines sometimes conflict.

- Edges may be **labeled** to indicate their function:

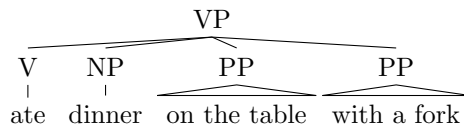


Dependency trees tell us who did what to whom.

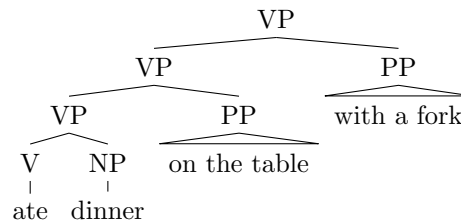
4.2 Expressiveness

(Unlabeled) dependency trees are less expressive than CFG derivations. That means they hide some of the ambiguity in CFGs that we may not care about. Remember the different representations for PP modification?

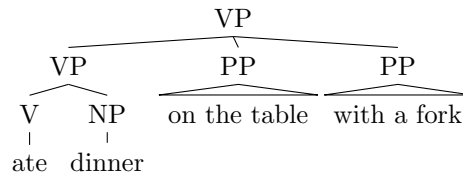
- Flat



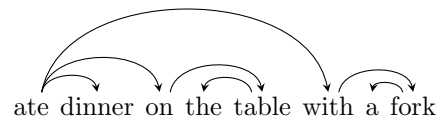
- (Chomsky) adjunction



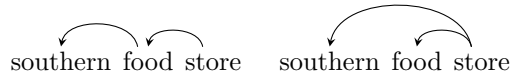
- Two-level (PTB)



These all look the same in a dependency parse:



- So if you didn't think there was any meaningful difference between these representations, you should be happy.
- Many kinds of CFG ambiguity remain in the dependency parse:

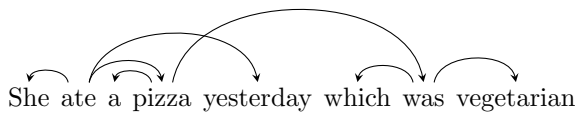


4.3 Applications

There are many; see slides for an example.

4.4 Projectivity

In projective dependency parsing, there can be no crossing edges.

- 
- Crossing edges are rare in English
 - They are quite common in other languages, like Czech.
 - We can build projective dependency banks from constituent treebanks, like PTB.
 - In this case, crossing edges are prohibited by construction.
 - What is the CFG analysis for the Pizza example?
 - In languages where non-projectivity is common, we can annotated it directly. Prague dependency treebank: 1.5M words of Czech.

4.5 Algorithms

Let's assume that the score for a dependency parse can be **factored** across the edges:

$$\Psi(G) = \bigotimes_i \psi(g(i) \rightarrow i) \quad (1)$$

Then we have efficient dynamic programs for dependency parsing.

- Non-projective dependency parsing reduces to the **maximum spanning tree** problem (in directed graphs).
 - We need a tree that touches all the nodes.
 - We want to get the maximum score $\bigotimes_i \psi(g(i) \rightarrow i)$, where $g(i)$ is the head of node i .
 - Chu-Liu-Edmonds algorithm computes this in $\mathcal{O}(N^3)$. [See slides.]
 - Tarjan algorithm is $\mathcal{O}(N^2)$
- Projective dependency parsing: Naive version

- $c[i, j, h]$ is the score of the best tree spanning $i \rightarrow j$ with head h

$$c[i, j, h] = \left(\bigoplus_{k, h'} c[i, k, h] \otimes c[k, j, h'] \otimes \psi(h \rightarrow h') \right) \oplus \left(\bigoplus_{k, h'} c[i, k, h'] \otimes c[k, j, h] \otimes \psi(h \rightarrow h') \right)$$

- The first line represents left-branching trees; the second line represents right-branching trees.
- **What is the complexity?** The size of the table is $\mathcal{O}(N^3)$. To fill in each node, we must consider $\mathcal{O}(N)$ possible split points and $\mathcal{O}(N)$ possible heads h' for the satellite subtree. So the time complexity is $\mathcal{O}(N^5)$.
- The Eisner algorithm reduces this to $\mathcal{O}(N^3)$, by adapting the CKY algorithm. We keep four tables instead of 1!
 - * scores of **incomplete** subtrees from i to j , headed to the left
 - * scores of **incomplete** subtrees from i to j , headed to the right
 - * scores of **complete** subtrees from i to j , headed to the left
 - * scores of **complete** subtrees from i to j , headed to the right
- Why?
 - * Incomplete subtrees can subsume complete subtrees heading in the same direction.
 - * Complete subtrees can combine if they are heading in opposite directions.
 - * Our goal is to produce a complete tree from 0 to N .

4.6 Learning dependency parsers

- Generative: $\psi(i \rightarrow j) = \log P(h = i|j)$, estimated from Treebank
- Log-linear: $\psi(i \rightarrow j) = \mathbf{w}^\top \mathbf{f}(x_i, x_j)$. Features:
 - * POS tag of x_i and x_j
 - * Word identity of x_i and x_j
 - * Word shape (e.g., suffix, prefix)
 - * Distance ($i - j$)
 - * ...
- Structured perceptron

$$\hat{\mathbf{y}} = \arg \max_{\mathbf{y}' \in \mathcal{T}(\mathbf{x})} \mathbf{w}^\top \mathbf{f}(\mathbf{x}, \mathbf{y}')$$

$$\mathbf{w} = \mathbf{w} + \mathbf{f}(\mathbf{x}, \mathbf{y}) - \mathbf{f}(\mathbf{x}, \hat{\mathbf{y}})$$

Same as before, but now $\arg \max_{\mathbf{y}' \in \mathcal{T}(\mathbf{x})}$ searches over dependency trees, using either MST or the Eisner algorithm.

- Conditional Random Field (CRF) a.k.a. globally-normalized conditional model

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_i \mathbf{f}(x_{h(i)} \rightarrow x_i) - \sum_j P(j \rightarrow i | \mathbf{x}) \mathbf{f}(x_j \rightarrow x_i) \quad (2)$$

We require **marginal** probabilities $P(j \rightarrow i | \mathbf{x})$. These can be obtained efficiently using a variant of inside-outside (for projective trees), and the matrix-tree theorem for non-projective trees [KGCC07].

4.7 Transition-based parsing

An alternative to exact global inference is transition-based parsing: making a series of local decisions. For projective dependency parsing, the algorithm is very similar to shift-reduce.

- Read the sentences left-to-right,
 - **shift**: push a word onto the stack
 - **arc-right**: make a right-facing edge
 - **arc-left**: make a left-facing edge

- [see [nivre slides](#)]

Some details:

- Inference
 - In practice, keep a “beam” of possible hypotheses.
 - This helps us recover from early mistakes
- Learning:
 - Identify the series of decisions required to produce the correct dependency parse.
 - Each decision in the derivation of the correct parse is a positive instance. Each other possible decision is a negative instance.
- No restriction to arc-factored features, can include any feature of the current partial parse, history of decisions, etc.
- Fast: linear time in the length of the sentence

References

- [KGCC07] Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. Structured prediction models via the matrix-tree theorem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 141–150, 2007.