ANLP Lecture 14
Treebanks and Statistical Parsing

Shay Cohen
(based on slides by Goldwater)

15 October 2019

Last class

- Recursive Descent Parsing
- Shift-Reduce Parsing
- CYK:

For $j > i + 1$:

$$\text{Chart}[A, i, j] = \bigvee_{k=i+1}^{j-1} \bigvee_{A \to B\ C} \mathit{Chart}[B, i, k] \wedge \mathit{Chart}[C, k, j]$$

Seed the chart, for $i + 1 = j$:

$\text{Chart}[A, i, i + 1] = \text{True}$ if there exists a rule $A \to w_{i+1}$ where w_{i+1} is the $(i + 1)$th word in the string

Towards probabilistic parsing

- We've seen various parsing algorithms, including one that parses exhaustively in polynomial time (CKY).
- But we haven't discussed how to choose which of many possible parses is the right one.
- The obvious solution: probabilities.

How big a problem is disambiguation?

- Early work in computational linguistics tried to develop broad-coverage hand-written grammars.
  - That is, grammars that *include* all sentences humans would judge as grammatical in their language;
  - while *excluding* all other sentences.
- As coverage grows, sentences can have hundreds or thousands of parses. Very difficult to write heuristic rules for disambiguation.
- Plus, grammar is hard to keep track of! Trying to fix one problem can introduce others.
- Enter the **treebank grammar**.

Today's lecture

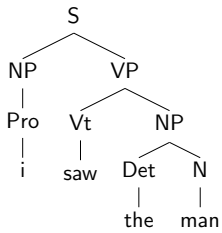
- What is a treebank and how do we use it to get a probabilistic grammar?
- How do we use probabilities in parsing?
- What are some problems that probabilistic CFGs help solve?
- What are some remaining weaknesses of simple PCFGs, and what are some ways to address them?

Treebank grammars

- The big idea: instead of paying linguists to write a grammar, pay them to annotate real sentences with parse trees.
- This way, we implicitly get a grammar (for CFG: read the rules off the trees).
- **And** we get probabilities for those rules (using any of our favorite estimation techniques).
- We can use these probabilities to improve disambiguation and even speed up parsing.

Treebank grammars

For example, if we have this tree in our corpus:



Then we add rules

S → NP VP
NP → Pro
Pro → i
VP → Vt NP
Vt → saw
NP → Det N
Art → the
N → man

With more trees, we can start to count rules and estimate their probabilities.

Example: The Penn Treebank

- The first large-scale parse annotation project, begun in 1989.
- Original corpus of syntactic parses: Wall Street Journal text
  - About 40,000 annotated sentences (1m words)
  - Standard phrasal categories like S, NP, VP, PP.
- Now many other data sets (e.g. transcribed speech), and different kinds of annotation; also inspired treebanks in many other languages.

Other language treebanks

- Many annotated with **dependency grammar** rather than CFG (see next lecture).
- Some require paid licenses, others are free.
- Just a few examples:
  - Danish Dependency Treebank
  - Alpino Treebank (Dutch)
  - Bosque Treebank (Portuguese)
  - Talbanken (Swedish)
  - Prague Dependency Treebank (Czech)
  - TIGER corpus, Tuebingen Treebank, NEGRA corpus (German)
  - Penn Chinese Treebank
  - Penn Arabic Treebank
  - Tuebingen Treebank of Spoken Japanese, Kyoto Text Corpus

Creating a treebank PCFG

A **probabilistic context-free grammar** (PCFG) is a CFG where each rule $\texttt{A} \rightarrow \alpha$ (where α is a symbol sequence) is assigned a probability $P(\alpha|A)$.

- The sum over all expansions of $\texttt{A}$ must equal 1: $\sum_{\alpha'} P(\alpha'|A) = 1$.
- Easiest way to create a PCFG from a treebank: MLE
  - Count all occurrences of $\texttt{A} \rightarrow \alpha$ in treebank.
  - Divide by the count of all rules whose LHS is $\texttt{A}$ to get $P(\alpha|A)$
- But as usual many rules have very low frequencies, so MLE isn't good enough and we need to smooth.

The generative model

Like *n*-gram models and HMMs, PCFGs are a **generative model**. Assumes sentences are generated as follows:

- Start with root category $\texttt{S}$.
- Choose an expansion α for $\texttt{S}$ with probability $P(\alpha|S)$.
- Then recurse on each symbol in α.
- Continue until all symbols are terminals (nothing left to expand).

The probability of a parse

- Under this model, the probability of a parse t is simply the product of all rules in the parse:

$$P(t) = \prod_{A \rightarrow \alpha \in t} p(A \rightarrow \alpha \mid A)$$

Statistical disambiguation example

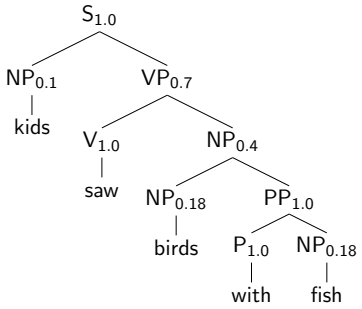
How can parse probabilities help disambiguate PP attachment?

- Let's use the following PCFG, inspired by Manning & Schuetze (1999):

S → NP VP	1.0	NP → NP PP	0.4
PP → P NP	1.0	NP → kids	0.1
VP → V NP	0.7	NP → birds	0.18
VP → VP PP	0.3	NP → saw	0.04
P → with	1.0	NP → fish	0.18
V → saw	1.0	NP → binoculars	0.1

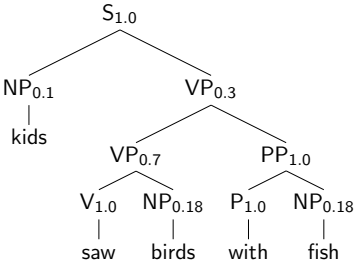
- We want to parse kids saw birds with fish.

Probability of parse 1



- $P(t_1) = 1.0 \cdot 0.1 \cdot 0.7 \cdot 1.0 \cdot 0.4 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0009072$

Probability of parse 2



- $P(t_2) = 1.0 \cdot 0.1 \cdot 0.3 \cdot 0.7 \cdot 1.0 \cdot 0.18 \cdot 1.0 \cdot 1.0 \cdot 0.18 = 0.0006804$
- which is less than $P(t_1) = 0.0009072$, so t_1 is preferred. Yay!

The probability of a sentence

- Since t implicitly includes the words $\vec{w}$, we have $P(t) = P(t, \vec{w})$.
- So, we also have a **language model**. Sentence probability is obtained by summing over $T(\vec{w})$, the set of valid parses of $\vec{w}$:

$$P(\vec{w}) = \sum_{t \in T(\vec{w})} P(t, \vec{w}) = \sum_{t \in T(\vec{w})} P(t)$$

- In our example, $P(\text{kids saw birds with fish}) = 0.0006804 + 0.0009072$.

How to find the best parse?

First, remember standard CKY algorithm.

- Fills in cells in well-formed substring table (chart) by combining previously computed child cells.

	1	2	3	4
0	Pro, NP			S
1		Vt,Vp,N		VP
2			Pro, PosPro, D	NP
3				N,Vi
	$_0$he$_1$	$_1$saw$_2$	$_2$her$_3$	$_3$duck$_4$

Probabilistic CKY

It is straightforward to extend CKY parsing to the probabilistic case.

- Goal: return the highest probability parse of the sentence.
  - When we find an A spanning (i, j), store its probability along with its label and backpointers in cell (i, j)
  - If we later find an A with the same span but higher probability, replace the probability for A in cell (i, j), and update the backpointers to the new children.
- Analogous to Viterbi: we iterate over all possible child pairs (rather than previous states) and store the probability and backpointers for the best one.

Probabilistic CKY

We also have analogues to the other HMM algorithms.

- The **inside algorithm** computes the probability of the sentence (analogous to forward algorithm)
  - Same as above, but instead of storing the *best* parse for A, store the *sum* of all parses.
- The **inside-outside algorithm** algorithm is a form of EM that learns grammar rule probs from unannotated sentences (analogous to forward-backward).

Best-first probabilistic parsing

- So far, we've been assuming **exhaustive** parsing: return all possible parses.
- But treebank grammars are huge!!
  - Exhaustive parsing of WSJ sentences up to 40 words long adds on average over 1m items to chart per sentence.[1]
  - Can be hundreds of possible parses, but most have extremely low probability: do we really care about finding these?
- **Best-first** parsing can help.

[1] Charniak, Goldwater, and Johnson, WVLC 1998.

Best-first probabilistic parsing

Use probabilities of subtrees to decide which ones to build up further.

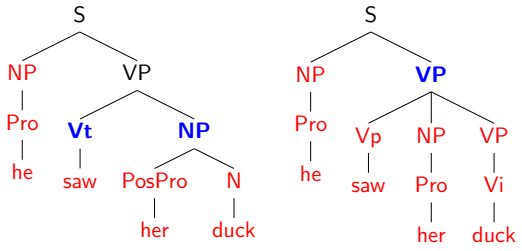
- Each time we find a new constituent, we give it a **score** ("figure of merit") and add it to an **agenda**[2], which is ordered by score.
- Then we pop the next item off the agenda, add it to the chart, and see which new constituents we can make using it.
- We add those to the agenda, and iterate.

Notice we are no longer filling the chart in any fixed order. Many variations on this idea, often limiting the size of the agenda by **pruning** out low-scoring edges (**beam search**).

[2] aka a priority queue

Best-first intuition

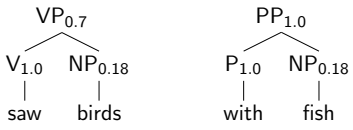
Suppose red constituents are in chart already; **blue** are on agenda.



If the VP in right-hand tree scores high enough, we'll pop that next, add it to chart, then find the S. So, we could complete the whole parse before even finding the alternative VP.

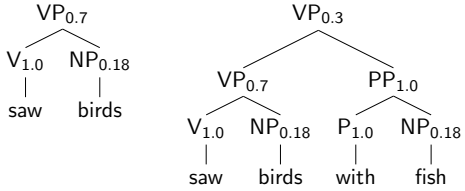
How do we score constituents?

Perhaps according to the probability of the subtree they span? So, P(left example)=(0.7)(0.18) and P(right example)=0.18.



How do we score constituents?

But what about comparing different sized constituents?



A better figure of merit

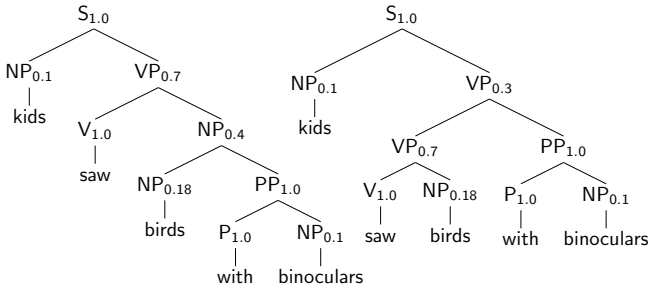
- If we use raw probabilities for the score, **smaller** constituents will almost always have higher scores.
  - Meaning we pop all the small constituents off the agenda before the larger ones.
  - Which would be very much like exhaustive bottom-up parsing!
- Instead, we can divide by the **number of words** in the constituent.
  - Very much like we did when comparing language models (recall **per-word** cross-entropy)!
- This works much better, though still not guaranteed to find the best parse first. Other improvements are possible.

But wait a minute...

Best-first parsing shows how simple ("vanilla") treebank PCFGs can improve **efficiency**. But do they really solve the problem of disambiguation?

- Our example grammar gave the right parse for this sentence:
  kids saw birds with fish
- What happens if we parse this sentence?
  kids saw birds with binoculars

Vanilla PCFGs: no lexical dependencies



- Exactly the same probabilities as the "fish" trees, except divide out $P(\text{fish}|\texttt{NP})$ and multiply in $P(\text{binoculars}|\texttt{NP})$ in each case.
- So, the same (left) tree is preferred, but now incorrectly!

Vanilla PCFGs: no lexical dependencies

Replacing one word with another with the same POS will never result in a different parsing decision, even though it should!

- More examples:
  - She stood by the door covered in tears vs.
    She stood by the door covered in ivy
  - She called on the student vs.
    She called on the phone.
    (assuming "on" has the same POS...)

Vanilla PCFGs: no global structural preferences

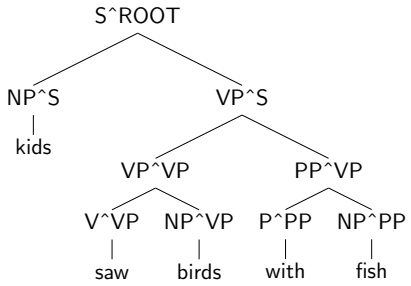
- Ex. in Switchboard corpus, the probability of `NP` → `Pronoun`
  - in **subject position** is 0.91 — he saw the dog
  - in **object position** is 0.34 — the dog bit him
- Lots of other rules also have different probabilities depending on where they occur in the sentence.
- But PCFGs are context-free, so an `NP` is an `NP` is an `NP`, and will have the same expansion probs regardless of where it appears.

Ways to fix PCFGs (1): parent annotation

Automatically create new categories that include the old category and its parent.

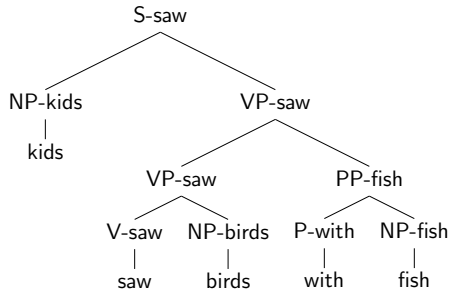
- So, an `NP` in subject position becomes `NP^S`, with other NPs becoming `NP^VP`, `NP^PP`, etc.
- Ex. rules:
  - `S^ROOT` → `NP^S VP^S`
  - `NP^S` → `Pro^NP`
  - `NP^S` → `NP^NP PP^NP`

Example of parent annotation



Ways to fix PCFGs (2): lexicalization

Again, create new categories, this time by adding the **lexical head** of the phrase:



- Now consider:

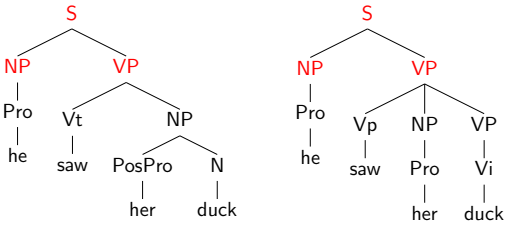
`VP-saw` → `VP-saw PP-fish` vs. `VP-saw` → `VP-saw PP-binoculars`

Practical issues, again

- All this category-splitting makes the grammar much more **specific** (good!)
- But leads to huge grammar blowup and very sparse data (bad!)
- Lots of effort over the years on how to balance these two issues.
  - Complex smoothing schemes (similar to N-gram interpolation/backoff).
  - More recently, emphasis on automatically learned subcategories and now neural parsers with implicit subcategories.
- But how do we know which method works best?

Evaluating parse accuracy

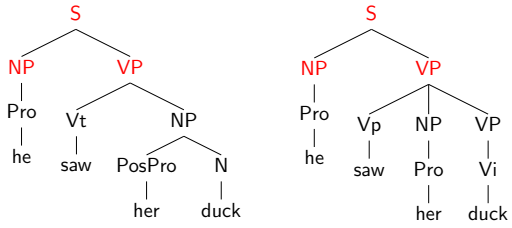
Compare **gold standard** tree (left) to **parser output** (right):



- Output constituent is counted correct if there is a gold constituent that spans the same sentence positions.
- Harsher measure: also require the constituent labels to match.
- Pre-terminals don't count as constituents.

Evaluating parse accuracy

Compare **gold standard** tree (left) to **parser output** (right):



- **Precision**: (# correct constituents)/(# in parser output) = 3/5
- **Recall**: (# correct constituents)/(# in gold standard) = 3/4
- **F-score**: balances precision/recall: 2pr/(p+r)

Parsing: where are we now?

- This lecture discussed the basics of probabilistic parsing and should give you a good idea of the issues involved.
- State-of-the-art parsers address these issues in other ways. For comparison, parsing F-scores on WSJ corpus are:
  - vanilla PCFG: $< 80\%$[3]
  - lexicalizing + cat-splitting: 89.5% (Charniak, 2000)
  - Best current parsers get about 94%
- We'll say a little bit about recent methods later, but most details in sem 2.

[3] Charniak (1996) reports 81% but using gold POS tags as input.

Parsing: where are we now?

Parsing is not just WSJ. Lots of situations are much harder!

- Other languages, esp with free word order (will discuss next time) or little annotated data.
- Other domains, esp with jargon (e.g., biomedical) or non-standard language (e.g., social media text).

And parsing to syntactic constituents isn't the only option, as we'll see next time...

Questions and exercises

1. How can probabilistic models of syntax be used
   - 1.1 to compute the probability of a parse?
   - 1.2 to compute the probability of a sentence?
   - 1.3 to help choose the right parse (disambiguation)?
   - 1.4 to help speed up parsing?
2. What's the problem with using constituent probability as the figure of merit to order the agenda in best-first parsing? What's one better alternative?
3. Give an example (besides the one in lecture) of two sentences with different correct parses where a vanilla PCFG would always assign the same parse to each sentence.
4. For the example on slide 28, draw the parent-annotated parse for the other analysis of kids saw birds with fish. Do the same for the lexicalized parse on slide 29. Can either (or both) of these methods hope to correctly disambiguate both kids saw birds with fish and kids saw birds with binoculars? If so, which are the rules that allow it?