Recap: Treebanks and PCFGs

- Ambiguity is a huge problem for parsers; adding probabilities might help to disambiguate.
- A simple way to create a probabilistic context free grammar (PCFG): just use MLE based on treebank counts.
- The PCFG is another type of generative model, and also gives us a way to compute tree probabilities.

Today’s lecture

- What are some problems that probabilistic CFGs help solve and how do we use probabilities in parsing?
- What are some remaining weaknesses of simple PCFGs, and what are some ways to address them?
- What is dependency syntax and why use it?
- How can we transform constituency $\rightarrow$ dependency parse?

The generative model

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

- Start with root category $S$.
- Choose an expansion $\alpha$ for $S$ with probability $P(\alpha|S)$.
- Then recurse on each symbol in $\alpha$.
- 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:

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

Statistical disambiguation example

How can parse probabilities help disambiguate PP attachment?

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

  - $S \rightarrow NP \ VP \ 1.0$
  - $NP \rightarrow NP \ PP \ 0.4$
  - $PP \rightarrow P \ NP \ 1.0$
  - $NP \rightarrow kids \ 0.1$
  - $VP \rightarrow V \ NP \ 0.7$
  - $NP \rightarrow birds \ 0.18$
  - $VP \rightarrow VP \ PP \ 0.3$
  - $NP \rightarrow saw \ 0.04$
  - $P \rightarrow with \ 1.0$
  - $NP \rightarrow fish \ 0.18$
  - $V \rightarrow saw \ 1.0$
  - $NP \rightarrow binoculars \ 0.1$

- We want to parse: $\text{kids saw birds with fish}$.

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!
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.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pro, NP</td>
<td></td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Vt,Vp,N</td>
<td>VP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Pro, PosPro, D</td>
<td>NP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>N,Vi</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^0$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.

But what about comparing different sized constituents?

---

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.

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

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...)

- Exactly the same probabilities as the "fish" trees, except divide out \( P(\text{fish}|\text{NP}) \) and multiply in \( P(\text{binoculars}|\text{NP}) \) in each case.

- So, the same (left) tree is preferred, but now incorrectly!
Vanilla PCFGs: no global structural preferences

- Ex. in Switchboard corpus, the probability of $NP \rightarrow $ 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:
  - $S^ROOT \rightarrow NP^S \ VP^S$
  - $NP^S \rightarrow Pro^NP$
  - $NP^S \rightarrow NP^NP \ PP^NP$

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:
  - $VP^Saw \rightarrow VP^Saw \ PP^Fish$ vs. $VP^Saw \rightarrow VP^Saw \ PP^Binoculars$
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.
- Intrinsic evaluation uses PARSEVAL measures: Precision, Recall, F-score on constituents (more later in course).

Parsing: where are we now?

- We discussed the basics of probabilistic parsing and you should now have 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%
  - 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.

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

Lexicalization, again

We saw that adding lexical head of the phrase can help choose the right parse:

In fact, due to increasing focus on multilingual NLP, constituency syntax/parsing (English-centric) is losing ground to dependency parsing...

Dependency syntax focuses on the head-dependent relationships.
Dependency syntax

An alternative approach to sentence structure.

- A fully lexicalized formalism: no phrasal categories.
- Assumes binary, asymmetric grammatical relations between words: head-dependent relations, shown as directed edges:

  kids → saw → birds → with → fish

- Here, edges point from heads to their dependents.

Sharon Goldwater  ANLP Lecture 15  28

It really is a tree!

- The usual way to show dependency trees is with edges over ordered sentences.
- But the edge structure (without word order) can also be shown as a more obvious tree:

  kids → saw → birds → with → fish

  saw → kids → birds → with → fish

Sharon Goldwater  ANLP Lecture 15  29

Labelled dependencies

It is often useful to distinguish different kinds of head → modifier relations, by labelling edges:

- Historically, different treebanks/languages used different labels.
- Now, the Universal Dependencies project aims to standardize labels and annotation conventions, bringing together annotated corpora from over 50 languages.
- Labels in this example (and in textbook) are from UD.

Sharon Goldwater  ANLP Lecture 15  30

Dependency trees

A valid dependency tree for a sentence requires:

- A single distinguished root word.
- All other words have exactly one incoming edge.
- A unique path from the root to each other word.

Sharon Goldwater  ANLP Lecture 15  29
Why dependencies??

Consider these sentences. Two ways to say the same thing:

\[
S \\
\text{NP} \rightarrow \text{Sasha} \\
\text{VP} \rightarrow \text{V} \text{ NP} \text{ NP} \\
\text{VP} \rightarrow \text{V} \text{ NP} \text{ PP} \\
\]

• We only need a few phrase structure rules:
  \[S \rightarrow \text{NP} \text{ VP}\]
  \[\text{VP} \rightarrow \text{V} \text{ NP} \text{ NP}\]
  plus rules for \text{NP} and \text{PP}.

Equivalent sentences in Russian

• Russian uses morphology to mark relations between words:
  – knigu means book (kniga) as a direct object.
  – devochke means girl (devochka) as indirect object (to the girl).

• So we can have the same word orders as English:
  – Sasha dal devochke knigu
  – Sasha dal knigu devochke

Equivalent sentences in Russian

• Russian uses morphology to mark relations between words:
  – knigu means book (kniga) as a direct object.
  – devochke means girl (devochka) as indirect object (to the girl).

• So we can have the same word orders as English:
  – Sasha dal devochke knigu
  – Sasha dal knigu devochke

• But also many others!
  – Sasha devochke dal knigu
  – Devochke dal Sasha knigu
  – Knigu dal Sasha devochke
Phrase structure vs dependencies

- In languages with free word order, phrase structure (constituency) grammars don’t make as much sense.
  - E.g., we would need both $S \rightarrow NP \ VP$ and $S \rightarrow VP \ NP$, etc. Not very informative about what’s really going on.

- Even more obvious if we just look at the trees without word order:

Pros and cons

- Sensible framework for free word order languages.
- Identifies syntactic relations directly. (using CFG, how would you identify the subject of a sentence?)
- Dependency pairs/chains can make good features in classifiers, for information extraction, etc.
- Parsers can be very fast (coming up...)

But

- The assumption of asymmetric binary relations isn’t always right...
  e.g., how to parse dogs and cats?
How do we annotate dependencies?

Two options:

1. Annotate dependencies directly.

2. Convert phrase structure annotations to dependencies. (Convenient if we already have a phrase structure treebank.)

Next slides show how to convert, assuming we have head-finding rules for our phrase structure trees.

... remove the phrasal categories. ...

... remove the (duplicated) terminals. ...

Lexicalized Constituency Parse
... and collapse chains of duplicates...

- kids saw
  - saw birds
    - birds fish
      - with fish

- kids saw
  - saw birds
    - with fish

... and collapse chains of duplicates...  

\[
\text{saw} \\
\text{kids} \quad \text{saw} \\
\text{saw} \quad \text{birds} \\
\text{fish} \\
\text{with} 
\]

... done!

\[
\text{saw} \\
\text{kids} \quad \text{birds} \\
\text{fish} \\
\text{with} 
\]

**Constituency Tree → Dependency Tree**

We saw how the *lexical head* of the phrase can be used to collapse down to a dependency tree:

\[
\text{S} \rightarrow \text{NP} \quad \text{VP} \\
\text{NP} \rightarrow \text{kids} \\
\text{VP} \rightarrow \text{saw} \\
\text{V} \rightarrow \text{saw} \\
\text{NP} \rightarrow \text{birds} \\
\text{PP} \rightarrow \text{with} \\
\text{P} \rightarrow \text{with} \\
\text{NP} \rightarrow \text{binoculars} \\
\]

• But how can we find each phrase’s head in the first place?

**Head Rules**

The standard solution is to use head rules: for every non-unary (P)CFG production, designate one RHS nonterminal as containing the head. \( S \rightarrow \text{NP} \quad \text{VP}, \quad \text{VP} \rightarrow \text{VP}, \quad \text{PP} \rightarrow \text{P} \quad \text{NP} \) (content head), etc.

\[
\text{S} \\
\text{NP} \rightarrow \text{kids} \\
\text{VP} \\
\text{V} \rightarrow \text{saw} \\
\text{NP} \rightarrow \text{birds} \\
\text{P} \rightarrow \text{with} \\
\text{NP} \rightarrow \text{binoculars} \\
\]

• Heuristics to scale this to large grammars: e.g., within an NP, last immediate N child is the head.
Then, propagate heads up the tree:

```
S
  NP-kids
    kids
  VP
    VP
      V-saw
        saw
      NP-birds
        birds
      P-with
        with
      NP-binoculars
        binoculars
```
Head Rules

Then, propagate heads up the tree:

- **S**: saw
- **NP**: kids
- **VP**: saw
- **NP**: birds
- **PP**: binoculars
- **VP**: saw
  - **V**: saw
  - **NP**: birds
  - **P**: with
  - **NP**: binoculars

Questions and exercises

1. How can probabilistic models of syntax be used:
   - (a) to compute the probability of a parse?
   - (b) to compute the probability of a sentence? (Hint: use the law of total probability)
   - (c) to help choose the right parse (disambiguation)?
   - (d) 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 22, draw the parent-annotated parse for the other analysis of *kids saw birds with fish*. Do the same for the lexicalized parse on slide 23. 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?

For review

- What are some problems that probabilistic CFGs help solve and how do we use probabilities in parsing?
- What are some remaining weaknesses of simple PCFGs, and what are some ways to address them?
- What is dependency syntax and why use it?
- How can we transform constituency → dependency parse?