Recall: Constituency
- Tree-structured syntactic analyses of sentences
- Nonterminals (NP, VP, etc.) as well as POS tags (bottom layer)
- Structured is defined by a CFG

Recall: PCFGs
- Context-free grammar: symbols which rewrite as one or more symbols
- Lexicon consists of “preterminals” (POS tags) rewriting as terminals (words)
- CFG is a tuple (N, T, S, R): N = nonterminals, T = terminals, S = start symbol (generally a special ROOT symbol), R = rules
- PCFG: probabilities associated with rewrites, normalize by source symbol

Adminstrivia
- Project 1 graded
- Submission on Gradescope
- Final project proposals due next Thursday
Recall: CKY

- Find argmax \( P(T|x) = \arg\max P(T, x) \)
- Dynamic programming: chart maintains the best way of building symbol \( X \) over span \((i, j)\)
- Loop over all split points \( k \), apply rules \( X \rightarrow Y \, Z \) to build \( X \) in every possible way

Outline

- Dependency representation, contrast with constituency
- Graph-based dependency parsers
- Transition-based (shift-reduce) dependency parsers
- State-of-the-art parsers

Dependency Representation

- Dependency syntax: syntactic structure is defined by these arcs
- Head (parent, governor) connected to dependent (child, modifier)
- Each word has exactly one parent except for the ROOT symbol, dependencies must form a directed acyclic graph
- POS tags same as before, usually run a tagger first as preprocessing
Still a notion of hierarchy! Subtrees often align with constituents

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)

Can label dependencies according to syntactic function

Major source of ambiguity is in the structure, so we focus on that more (labeling separately with a classifier works pretty well)
Dependency vs. Constituency: Coordination

- Constituency: ternary rule NP -> NP CC NP

- Dependency: first item is the head
- Coordination is decomposed across a few arcs as opposed to being a single rule production as in constituency
- Can also choose *and* to be the head
- In both cases, headword doesn’t really represent the phrase — constituency representation makes more sense

Stanford Dependencies

- Designed to be practically useful for relation extraction

Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas

- Dependency is often more useful in practice (models predicate argument structure)
- Slightly different representational choices:
  - PP attachment is better modeled under dependency
  - Coordination is better modeled under constituency
- Dependency parsers are easier to build: no “grammar engineering”, no unaries, easier to get structured discriminative models working well
- Dependency parsers are usually faster
- Dependencies are more universal cross-lingually
Universal Dependencies

- Annotate dependencies with the same representation in many languages

English

Bulgarian

Czech

Swiss

http://universaldependencies.org/

Graph-Based Parsing

Defining Dependency Graphs

- Words in sentence $x$, tree $T$ is a collection of directed edges (parent($i$), $i$) for each word $i$
  - Parsing = identify parent($i$) for each word
  - Each word has exactly one parent. Edges must form a projective tree
- Log-linear CRF (discriminative): $P(T|x) = \exp \left( \sum_i w^T f(i, \text{parent}(i), x) \right)$
- Example of a feature = $l[\text{head}=\text{to} \& \text{modifier}=$house$]$

ROOT the dog ran to the house

Biaffine Neural Parsing

- Neural CRFs for dependency parsing: let $c =$ LSTM embedding of $i$, $p =$ LSTM embedding of parent($i$), $\text{score}(i, \text{parent}(i), x) = p^T U c$

Dozat and Manning (2017)
Generalizing CKY

- DP chart with three dimensions: start, end, and head, start <= head < end
- new score = chart(2, 5, 4) + chart(5, 7, 5) + edge score(4 -> 5)
- score(2, 7, 4) = max(score(2, 7, 4), new score)
- Many spurious derivations: can build the same tree in many ways...need a better algorithm
- Eisner’s algorithm is cubic time

Evaluating Dependency Parsing

- UAS: unlabeled attachment score. Accuracy of choosing each word’s parent (n decisions per sentence)
- LAS: additionally consider label for each edge
- Log-linear CRF parser, decoding with Eisner algorithm: 91 UAS
- Higher-order features from Koo parser: 93 UAS
- Best English results with neural CRFs (Dozat and Manning): 95-96 UAS

Shift-Reduce Parsing

- Similar to deterministic parsers for compilers
  - Also called transition-based parsing
- A tree is built from a sequence of incremental decisions moving left to right through the sentence
- Stack containing partially-built tree, buffer containing rest of sentence
- Shifts consume the buffer, reduces build a tree on the stack
**Shift-Reduce Parsing**

ROOT  
I ate some spaghetti bolognese

- Initial state: Stack: [ROOT] Buffer: [I ate some spaghetti bolognese]
- Shift: top of buffer -> top of stack
  - Shift 1: Stack: [ROOT I] Buffer: [ate some spaghetti bolognese]
  - Shift 2: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]

**Shift-Reduce Parsing**

ROOT  
I ate some spaghetti bolognese

- State: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]
- Left-arc (reduce): Let \( \sigma \) denote the stack, \( \sigma \cdot w_{-1} \) = stack ending in \( w_{-1} \)
  - “Pop two elements, add an arc, put them back on the stack”
    \[
    \sigma \cdot [w_{-2}, w_{-1}] \rightarrow \sigma \cdot [w_{-1}]
    \]
    \( w_{-2} \) is now a child of \( w_{-1} \)
- State: Stack: [ROOT ate] Buffer: [some spaghetti bolognese]

**Arc-Standard Parsing**

ROOT  
I ate some spaghetti bolognese

- Start: stack contains [ROOT], buffer contains [I ate some spaghetti bolognese]
- Arc-standard system: three operations
  - Shift: top of buffer -> top of stack
    - Left-Arc: \( \sigma \cdot [w_{-2}, w_{-1}] \rightarrow \sigma \cdot [w_{-1}] \) \( w_{-2} \) is now a child of \( w_{-1} \)
    - Right-Arc: \( \sigma \cdot [w_{-2}, w_{-1}] \rightarrow \sigma \cdot [w_{-2}] \) \( w_{-1} \) is now a child of \( w_{-2} \)
- End: stack contains [ROOT], buffer is empty []
- How many transitions do we need if we have \( n \) words in a sentence?

**Arc-Standard Parsing**

ROOT  
I ate some spaghetti bolognese

- Stack contains [ROOT], buffer contains [I ate some spaghetti bolognese]
- LA pop two, left arc between them
- RA pop two, right arc between them

- [ROOT]  
- [ROOT I]  
- [ROOT ate]  

- S top of buffer -> top of stack
- LA pop two, left arc between them
- RA pop two, right arc between them

- [I ate some spaghetti bolognese]
- [ate some spaghetti bolognese]
- [some spaghetti bolognese]
- [some spaghetti bolognese]

- Could do the left arc later! But no reason to wait
- Can’t attach ROOT <- ate yet even though this is a correct dependency!
### Arc-Standard Parsing

**I ate some spaghetti bolognese**

\[ \text{Stack consists of all words that are still waiting for right children, end with a bunch of right-arc ops} \]

**Final state:**

1. **Stack:** top of buffer -> top of stack
2. **LA:** pop two, left arc between them
3. **RA:** pop two, right arc between them

### Building Shift-Reduce Parsers

**[ROOT]**

- How do we make the right decision in this case?
- Only one legal move (shift)

**[ROOT ate some spaghetti bolognese]**

- How do we make the right decision in this case? (all three actions legal)
- Multi-way classification problem: shift, left-arc, or right-arc?

\[
\arg\max_{a \in \{S, LA, RA\}} w^T f(\text{stack, buffer, } a)
\]

### Features for Shift-Reduce Parsing

**[ROOT ate some spaghetti bolognese] [bolognese]**

- Features to know this should left-arc?
- One of the harder feature design tasks!
- In this case: the stack tag sequence VBD - DT - NN is pretty informative — looks like a verb taking a direct object which has a determiner in it
- Things to look at: top words/POS of buffer, top words/POS of stack, leftmost and rightmost children of top items on the stack
Training a Greedy Model

\[
\text{argmax}_{\alpha \in \{S,L,A,R\}} w^T f(\text{stack, buffer, } \alpha)
\]

- Can turn a tree into a decision sequence \( \alpha \) by building an oracle
- Train a classifier to predict the right decision using these as training data
- Training data assumes you made correct decisions up to this point and teaches you to make the correct decision, but what if you screwed up…

Greedy training

- Greedy: 2n local training examples
- Non-gold states unobserved during training: consider making bad decisions but don’t condition on bad decisions

Speed Tradeoffs

<table>
<thead>
<tr>
<th>Parser</th>
<th>Dev UAS</th>
<th>Dev LAS</th>
<th>Test UAS</th>
<th>Test LAS</th>
<th>Speed (sent/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unoptimized S-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standard eager</td>
<td>89.9</td>
<td>88.7</td>
<td>89.7</td>
<td>88.3</td>
<td>51</td>
</tr>
<tr>
<td>Optimized S-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malt:sp</td>
<td>90.0</td>
<td>88.8</td>
<td>89.9</td>
<td>88.5</td>
<td>560</td>
</tr>
<tr>
<td>Malt:eager</td>
<td>90.1</td>
<td>88.9</td>
<td>90.1</td>
<td>88.7</td>
<td>535</td>
</tr>
<tr>
<td>Graph-based</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSTParser</td>
<td>92.1</td>
<td>90.8</td>
<td>92.0</td>
<td>90.5</td>
<td>12</td>
</tr>
<tr>
<td>Neural S-R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our parser</td>
<td>92.2</td>
<td>91.0</td>
<td>92.0</td>
<td>90.7</td>
<td>1013</td>
</tr>
</tbody>
</table>

- Many early-2000s constituency parsers were ~5 sentences/sec
- Using S-R used to mean taking a performance hit compared to graph-based, that’s no longer (quite as) true

Chen and Manning (2014)
### Dependency Parsers

- 2005: Eisner algorithm graph-based parser was SOTA (~91 UAS)
- 2010: Koo’s 3rd-order parser was SOTA for graph-based (~93 UAS)
- 2012: Maltparser was SOTA was for transition-based (~90 UAS)
- 2014: Chen and Manning got 92 UAS with transition-based neural model
- 2016: Improvements to Chen and Manning

### Shift-Reduce with FFNNs

**Softmax layer:**
\[ p = \text{softmax}(W_2h) \]

**Hidden layer:**
\[ h = (W_1^u x^u + W_1^d x^d + W_1^b b_1)^3 \]

**Input layer:** \([x^u, x^d, x^b]\)

#### Configuration
- \( \text{ROOT has VBZ good JJ} \)
- \( \text{control NN ...} \)
- \( \text{nsubj} \)
- \( \text{He PRP} \)

Chen and Manning (2014)

### Parsey McParseFace (a.k.a. SyntaxNet)

- 94.61 UAS on the Penn Treebank using a global transition-based system with early updating (compared to 95.8 for Dozat, 93.7 for Koo in 2009)
- Additional data harvested via “tri-training”, form of self-training
- Feedforward neural nets looking at words and POS associated with words in the stack / those words’ children / words in the buffer
- Feature set pioneered by Chen and Manning (2014), Google fine-tuned it

Andor et al. (2016)

### Shift-Reduce Constituency

![Diagram of shift-reduce constituency]

- Can do shift-reduce for constituency as well, reduce operation builds constituents

Cross and Huang (2016)
Pre-trained Models

- Improves the neural CRF by using a transformer layer (self-attentive), character-level modeling, and ELMo
- 95.21 on Penn Treebank dev set — much better than past parsers! (~92-93)
- This constituency parser with BERT is one of the strongest today, or use a transition-based version due to Kitaev and Klein (2020)

Recap

- Shift-reduce parsing can work nearly as well as graph-based
- Arc-standard system for transition-based parsing
- Purely greedy or more “global” approaches
- Next time: semantic parsing