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

<table>
<thead>
<tr>
<th>Grammar (CFG)</th>
<th>Lexicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT → S</td>
<td>1.0</td>
</tr>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>NP → DT NN</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → NN NNS</td>
<td>0.5</td>
</tr>
</tbody>
</table>

- 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
Recall: CKY

- Find \( \text{argmax} \ P(T|\mathbf{x}) = \text{argmax} \ P(T, \mathbf{x}) \)
- 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
**Dependency Parsing**

- Still a notion of hierarchy! Subtrees often align with constituents

- **Sentence:** the dog to the house

  - **Dependency:**
    
    - **VBD**
      
      - ran
    
    - **NN**
      
      - dog
    
    - **TO**
      
      - to
    
    - **NN**
      
      - house

  - **Constituency:**

- **Sentence:** the dog to ran the dog

  - **Dependency:**
    
    - **VBD**
      
      - ran
    
    - **NN**
      
      - dog
    
    - **TO**
      
      - to
    
    - **VBD**
      
      - ran
    
    - **NN**
      
      - dog

  - **Constituency:**

**Dependency vs. Constituency:**

- **PP Attachment**

  - **Sentence:** the children ate the cake with a spoon

  - **Dependency:**
    
    - **VBD**
      
      - ate
    
    - **NN**
      
      - children
    
    - **TO**
      
      - the
    
    - **NN**
      
      - cake
    
    - **IN**
      
      - with
    
    - **NN**
      
      - spoon

  - **Constituency:**

- **Coordination**

  - **Sentence:** the children ate the cake with a spoon

  - **Dependency:**
    
    - **VBD**
      
      - ate
    
    - **NN**
      
      - children
    
    - **TO**
      
      - the
    
    - **NN**
      
      - cake
    
    - **IN**
      
      - with
    
    - **NN**
      
      - spoon

  - **Constituency:**

- **Sentence:** dogs in houses and cats

  - **Dependency:**
    
    - **NN**
      
      - dogs
    
    - **IN**
      
      - in
    
    - **NN**
      
      - houses
    
    - **CC**
      
      - and
    
    - **NN**
      
      - cats

  - **Constituency:**

**More predicate-argument focused view of syntax**

- "What’s the main verb of the sentence? What is its subject and object?" — easier to answer under dependency parsing
### Dependency vs. Constituency: Coordination

- **Dependency:** first item is the head
  
  - dogs in houses and cats
  
  - 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

### Universal Dependencies

- Annotate dependencies with the same representation in many languages

  - English
  
  - Bulgarian
  
  - Czech
  
  - Swiss

  [http://universaldependencies.org/](http://universaldependencies.org/)
Graph-Based Parsing

Defining Dependency Graphs

- Words in sentence $x$, tree $T$ is a collection of directed edges $(\text{parent}(i), i)$ for each word $i$
- Parsing = identify $\text{parent}(i)$ for each word $i$
- 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 = $I[\text{head}=\text{to} \& \text{modifier}=\text{house}]$

ROOT the dog ran to the house

Biaffine Neural Parsing

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

Dozat and Manning (2017)

Generalizing CKY

- DP chart with three dimensions: start, end, and head, start $\leq$ head $\leq$ end
- new score = $\text{chart}(2, 5, 4) + \text{chart}(5, 7, 5) + \text{edge score}(4 \rightarrow 5)$
- score$(2, 7, 4) = \max(\text{score}(2, 7, 4), \text{new score})$
- 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

Example:

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**

- **I ate some spaghetti bolognese**

  - **State:** Stack: [ROOT I ate]  Buffer: [some spaghetti bolognese]
  - **Left-arc (reduce):** Let σ denote the stack, σ[w_1] = stack ending in w_1
    - **“Pop two elements, add an arc, put them back on the stack”**
    - \[\sigma[w_{-2}, w_{-1}] \rightarrow \sigma[w_{-1}]\]
      - w_{-2} is now a child of w_{-1}
  - **State:** Stack: [ROOT ate]  Buffer: [some spaghetti bolognese]

**Arc-Standard Parsing**

- **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:** \[\sigma[w_{-2}, w_{-1}] \rightarrow \sigma[w_{-1}]\]
      - w_{-2} is now a child of w_{-1}
    - **Right-Arc:** \[\sigma[w_{-2}, w_{-1}] \rightarrow \sigma[w_{-2}]\]
      - 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**

- **I ate some spaghetti bolognese**

  - **Top of buffer -> top of stack**
  - **pop two, left arc between them**
  - **pop two, right arc between them**

- **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**

  - **Top of buffer -> top of stack**
  - **pop two, left arc between them**
  - **pop two, right arc between them**

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

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

- **[ROOT ate spaghetti] [some] [bolognese]**
Arc-Standard Parsing

- [ROOT] [I ate some spaghetti bolognese]
- [ROOT ate spaghetti bolognese]  
  - top of buffer -> top of stack
  - LA pop two, left arc between them
  - RA pop two, right arc between them
- [ROOT ate spaghetti]
  - some
- [ROOT ate]
  - spaghetti
  - some
  - bolognese
- Final state: [ROOT] [ate [spaghetti [bolognese]]]

Building Shift-Reduce Parsers

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

- [ROOT ate some spaghetti] [bolognese]
  - \( \arg\max_{a \in \{S, LA, RA\}} w^T f(\text{stack, buffer, } a) \)
  - Can turn a tree into a decision sequence \( a \) 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

**State space**

- Start state
- Gold end state

**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>Malt:sp</td>
<td>90.3</td>
<td>89.2</td>
<td>89.9</td>
<td>88.6</td>
<td>63</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>MSTParser</td>
<td>92.1</td>
<td>90.8</td>
<td><strong>92.0</strong></td>
<td>90.5</td>
</tr>
<tr>
<td>Neural S-R</td>
<td>Our parser</td>
<td><strong>92.2</strong></td>
<td><strong>91.0</strong></td>
<td><strong>92.0</strong></td>
<td><strong>90.7</strong></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)

**Shift-Reduce Constituency**

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

Cross and Huang (2016)

**Shift-Reduce Constituency**

- “Tetra tagging”: four possible tags to get unlabeled binary trees

Kitaev and Klein (2020)
**State-of-the-art 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**

**SoMax layer:**
\[ p = \text{softmax}(W_2 h) \]

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

**Input layer:** \([x^w, x^l, x^b]\)

**Configuration**

- ROOT
- has.VBZ
- good.JJ
- control.NN
- nsubj
- He.PRPR

Danqi 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)
Challenges in other languages

- Swiss German example: note that the arcs cross, unlike in our English examples, which were almost entirely projective
- (Swiss German also has famous non-context-free constructions)
- As a result: some different transition-based algorithms are needed

Multilingual Parsing

- Interest in multilingual dependency parsing as far back as CoNLL 2006 shared task
- Now: can parse many languages with one pre-trained model

Reflections on Structure

- What is the role of it now?
- Systems still make these kinds of judgments, just not explicitly
- To improve systems, do we need to understand what they do?

Recap

- Shift-reduce parsing can work nearly as well as graph-based
- Arc-standard system for transition-based parsing
- Strong learning-based parsers, including multilingual parsers