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>
<tr>
<td>NP → NP PP</td>
<td>0.3</td>
</tr>
<tr>
<td>VP → VBP NP</td>
<td>0.7</td>
</tr>
<tr>
<td>VP → VBP NP PP</td>
<td>0.3</td>
</tr>
<tr>
<td>PP → IN NP</td>
<td>1.0</td>
</tr>
<tr>
<td>PRP → ran</td>
<td></td>
</tr>
<tr>
<td>VBP → interest</td>
<td>1.0</td>
</tr>
<tr>
<td>VBZ → raises</td>
<td>1.0</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

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

```
VBD ran
```

```
NN dog
```

```
TO to
```

```
NN house
```

```
DT the
```

```
```

**Dependency vs. Constituency: PP Attachment**

- Constituency: several rule productions need to change

```
NP
```

```
VBD
```

```
NP
```

```
The children ate the cake with a spoon
```

```
```

```
```

**Dependency vs. Constituency: Coordination**

- Constituency: ternary rule NP -> NP CC NP

```
NP
```

```
PP
```

```
and
```

```
NP
```

```
PP
```

```
```

```
```

**Dependency vs. Constituency: PP Attachment**

- Dependency: one word (with) assigned a different parent

```
the children ate the cake with a spoon
```

```
```

- 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*  
  - *dogs in houses and cats*  
  - *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**

  - Dependencies are more universal cross-lingually: Czech was one of the first languages for dep parsing in NLP due to its free word order

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

Defining Dependency Graphs

- Words in sentence $\mathbf{x}$, tree $\mathbf{T}$ is a collection of directed edges $(\text{parent}(i), i)$ 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): $P(\mathbf{T}|\mathbf{x}) = \exp\left(\sum_i w^T f(i, \text{parent}(i), \mathbf{x})\right)$
- Example of a feature = $I[\text{head=to} \& \text{modifier=house}]$

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($i$). $\text{score}(i, \text{parent}(i), \mathbf{x}) = p^T U c$

LSTM looks at words and POS

Dozat and Manning (2017)

Generalizing CKY

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

4 = report
5 = on

wrote a on Mars
4 long report 5 on
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

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

**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, $\sigma|w_{-1} = \text{stack ending in } w_{-1}$
    - “Pop two elements, add an arc, put them back on the stack”
      $$\sigma|w_{-2}, w_{-1} \rightarrow \sigma|w_{-1}, w_{-2} \text{ is now a child of } w_{-1}$$
  - **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: $\sigma|w_{-2}, w_{-1} \rightarrow \sigma|w_{-1}$  $w_{-2}$ 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 []
  - 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**

- **Stack consists of all words that are still waiting for right children, end with a bunch of right-arc ops**
- **Final state:**

**Building Shift-Reduce Parsers**

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

**Features for Shift-Reduce Parsing**

- **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**

- **Can turn a tree into a decision sequence \( \mathbf{a} \) 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>standard eager</td>
<td>89.9</td>
<td>88.7</td>
<td>89.7</td>
<td>88.3</td>
</tr>
<tr>
<td></td>
<td>Malt:sp</td>
<td>90.0</td>
<td>88.8</td>
<td>89.9</td>
<td>88.6</td>
</tr>
<tr>
<td></td>
<td>Malt:eager</td>
<td>90.1</td>
<td>88.9</td>
<td>90.1</td>
<td>88.7</td>
</tr>
<tr>
<td>Optimized S-R</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></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)**

- “•”$: This terminal node is a left-child.
- “•“$: This terminal node is a right-child.
- “•”$: The shortest span crossing this fence-post is a left-child.
- “•”$: The shortest span crossing this fence-post is a right-child.

<table>
<thead>
<tr>
<th>Sents/s</th>
<th>Hardware</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vilares et al. (2019)</td>
<td>942</td>
<td>1x GPU</td>
</tr>
<tr>
<td>Kitaev et al. (2019)*</td>
<td>39</td>
<td>1x GPU</td>
</tr>
<tr>
<td>Zhou and Zhao (2019)*</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>This work*</td>
<td>1200</td>
<td>1x TPU v3-8</td>
</tr>
</tbody>
</table>

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

- Softmax layer:
  \[ p = \text{softmax}(W_2h) \]
- Hidden layer:
  \[ h = (W_1^u x^u + W_1^d x^d + W_1^l x^l + b_1)^3 \]
- Input layer: \([x^w, x^d, x^l]\)

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

Danqi Chen and Manning (2014)
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