CS388: Natural Language Processing

Lecture 12: Dependency I

Greg Durrett
Administrivia

- Project 1 graded, discussion at end of lecture
- Mini 2 due tonight
- Final project proposals due next Tuesday
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: 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 $X \to YZ$ to build $X$ in every possible way
Recall: Top-down Parsing

- Can score split points and also labels
- Dynamic programming version:
  \[ s_{\text{best}}(i, j) = \max_{\ell, k} [s_{\text{label}}(i, j, \ell) + \tilde{s}_{\text{split}}(i, k, j)] \]
  
  (best way of building \( i \) and \( j \) involves maxing over split point and a \textit{single} label)
- Greedy top-down version: at each stage, predict split point \( k \) and label \( l \)
  \[ (\hat{l}, \hat{k}) = \arg\max_{\ell, k} [s_{\text{label}}(i, j, \ell) + s_{\text{split}}(i, k, j)] \]
Outline

- Dependency representation, contrast with constituency
- Projectivity
- Graph-based dependency parsers
Dependency Representation
Lexicalized Parsing

S(ran)
  
  VP(ran)
    
    PP(to)
      
      NP(house)
        
        DT(the)
        
        NN(house)
          
          VBD(ran)
            
            TO(to)
              
              DT(the)
                
                NN(dog)
                  
                  DT(the)
                    
                    NN(dog)
                      
                      the
                        
                        dog
                          
                          the
                            
                            ran
                              
                              to
                                
                                the
                                  
                                  house
                                    
                                    the
                                      
                                      dog
Dependency Parsing

- 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)
Constituency: several rule productions need to change
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

- Constituency: ternary rule NP -> NP CC NP
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
Dependency vs. Constituency

- 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

<table>
<thead>
<tr>
<th>Language</th>
<th>Sentence</th>
<th>Universal Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>The dog was chased by the cat</td>
<td>![Diagram of English sentence]</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>Кучето се преследваше от котката</td>
<td>![Diagram of Bulgarian sentence]</td>
</tr>
<tr>
<td>Czech</td>
<td>Pes byl honěn kočkou</td>
<td>![Diagram of Czech sentence]</td>
</tr>
<tr>
<td>Swiss</td>
<td>Hunden jagades av katten</td>
<td>![Diagram of Swiss sentence]</td>
</tr>
</tbody>
</table>

[http://universaldependencies.org/](http://universaldependencies.org/)
Any subtree is a contiguous span of the sentence <-> tree is *projective*
Projectivity

- Projective <-> no “crossing” arcs

- Crossing arcs:

  - dogs in houses and cats
  - the dog ran to the house

credit: Language Log
Projectivity in other languages

- Swiss German example

- (Swiss German also has famous non-context-free constructions)

credit: Pitler et al. (2013)
## Projectivity

- Number of trees produceable under different formalisms

<table>
<thead>
<tr>
<th></th>
<th>Arabic</th>
<th>Czech</th>
<th>Danish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projective</td>
<td>1297 (88.8)</td>
<td>55872 (76.8)</td>
<td>4379 (84.4)</td>
</tr>
<tr>
<td>Sentences</td>
<td>1460</td>
<td>72703</td>
<td>5190</td>
</tr>
</tbody>
</table>

- Many trees in other languages are nonprojective

Pitler et al. (2013)
Projectivity

- Number of trees produceable under different formalisms

<table>
<thead>
<tr>
<th>1-Endpoint-Crossing</th>
<th>Arabic</th>
<th>Czech</th>
<th>Danish</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1457 (99.8)</td>
<td>71810 (98.8)</td>
<td>5144 (99.1)</td>
</tr>
<tr>
<td>Well-nested, block degree 2</td>
<td>1458 (99.9)</td>
<td>72321 (99.5)</td>
<td>5175 (99.7)</td>
</tr>
<tr>
<td>Gap-Minding</td>
<td>1394 (95.5)</td>
<td>70695 (97.2)</td>
<td>4985 (96.1)</td>
</tr>
<tr>
<td>Projective</td>
<td>1297 (88.8)</td>
<td>55872 (76.8)</td>
<td>4379 (84.4)</td>
</tr>
<tr>
<td>Sentences</td>
<td>1460</td>
<td>72703</td>
<td>5190</td>
</tr>
</tbody>
</table>

- Many trees in other languages are nonprojective
- Some other formalisms (that are harder to parse in), most useful one is 1-Endpoint-Crossing

Pitler et al. (2013)
Graph-Based Parsing
Defining Dependency Graphs

- Words in sentence $\mathbf{x}$, tree $T$ is a collection of directed edges $(\text{parent}(i), i)$ for each word $i$
  - Parsing = identify $\text{parent}(i)$ for each word
  - Each word has exactly one parent. Edges must form a projective tree

- Log-linear CRF (discriminative): $P(T|\mathbf{x}) = \exp \left( \sum_i w^T f(i, \text{parent}(i), \mathbf{x}) \right)$

- Example of a feature = $I[\text{head=to} \land \text{modifier=house}]$ (more in a few slides)

ROOT the dog ran to the house
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)
- Time complexity of this?
- Many spurious derivations: can build the same tree in many ways...need a better algorithm
Eisner’s Algorithm: $O(n^3)$

- Cubic-time algorithm
- Maintain two dynamic programming charts with dimension $[n, n, 2]$:
  - **Complete items**: head is at “tall end”, may be missing children on tall side
  - **Incomplete items**: arc from “tall” to “short” end, word on short end may also be missing children
Eisner’s Algorithm: $O(n^3)$

- **Complete item**: all children are attached, head is at the “tall end”
- **Incomplete item**: arc from “tall end” to “short end”, may still expect children

- Take two adjacent complete items, add arc and build incomplete item

- Take an incomplete item, complete it

ROOT

<table>
<thead>
<tr>
<th>DT</th>
<th>NN</th>
<th>VBD</th>
<th>TO</th>
<th>DT</th>
<th>NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>dog</td>
<td>ran</td>
<td>to</td>
<td>the</td>
<td>house</td>
</tr>
</tbody>
</table>
Eisner’s Algorithm: \(O(n^3)\)

1) Build incomplete span

2) Promote to complete

3) Build incomplete span
Eisner’s Algorithm: $O(n^3)$

4) Promote to complete
Eisner’s Algorithm: $O(n^3)$

- Attaching to ROOT makes an incomplete item with left children, attaches with right children subsequently to finish the parse.

- We’ve built left children and right children of *ran* as complete items.
Eisner’s Algorithm

Right complete

Left complete

Right incomplete

Left incomplete

ROOT the dog ran to the house

ROOT the dog ran to the house
Eisner’s Algorithm

- Eisner’s algorithm doesn’t have split point ambiguities like CKY does.
- Left and right children are built independently, heads are edges of spans.
- Charts are $n \times n \times 2$ because we need to track arc direction / left vs right.

Eisner:

- $n^5$
- ROOT
- DT the
- NN dog
- VBD ran
- TO to
- DT the
- NN house
Can implement decoding and marginal computation using Eisner’s algorithm to max/sum over projective trees

Conceptually the same as inference/learning for sequential CRFs for NER, can also use margin-based methods
Features in Graph-Based Parsing

- Dynamic program exposes the parent and child indices

\[ f(i, \text{parent}(i), \mathbf{x}) \]

- McDonald et al. (2005) — conjunctions of parent and child words + POS, POS of words in between, POS of surrounding words
  - HEAD=TO & MOD=NN
  - HEAD=TO & MOD=house
  - HEAD=TO & MOD-1=the
  - ARC_CROSSES=DT

ROOT DT NN VBD TO DT NN
the dog ran to the house
Higher-Order Parsing

\[ f(i, \text{parent}(i), \text{parent}(\text{parent}(i)), x) \]

- Track additional state during parsing so we can look at "grandparents" (and siblings). O(n^4) dynamic program or use approximate search

Koo and Collins (2009)
Neural CRFs for dependency parsing: let $c = \text{LSTM embedding of } i$, $p = \text{LSTM embedding of parent}(i)$. $score(i, \text{parent}(i), x) = p^T U c$

LSTM looks at words and POS

Dozat and Manning (2017)
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
Head-driven phrase structure grammar (HPSG): very complex grammar formalism which annotates large feature structures over tree.

Very little work on HPSG in NLP.
Parsing with “HPSG”

- Joint model of constituency and dependency combining ideas from Dozat + Manning and Stern et al.

Zhou and Zhao (2019)
Parsing with “HPSG”

- Slightly stronger results than Dozat + Manning, significantly better results on Chinese

<table>
<thead>
<tr>
<th>Model</th>
<th>English</th>
<th></th>
<th>Chinese</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UAS</td>
<td>LAS</td>
<td>UAS</td>
</tr>
<tr>
<td>Chen and Manning (2014)</td>
<td>91.8</td>
<td>89.6</td>
<td>83.9</td>
</tr>
<tr>
<td>Andor et al. (2016)</td>
<td>94.61</td>
<td>92.79</td>
<td>-</td>
</tr>
<tr>
<td>Zhang et al. (2016)</td>
<td>93.42</td>
<td>91.29</td>
<td>87.65</td>
</tr>
<tr>
<td>Cheng et al. (2016)</td>
<td>94.10</td>
<td>91.49</td>
<td>88.1</td>
</tr>
<tr>
<td>Kuncoro et al. (2016)</td>
<td>94.26</td>
<td>92.06</td>
<td>88.87</td>
</tr>
<tr>
<td>Ma and Hovy (2017)</td>
<td>94.88</td>
<td>92.98</td>
<td>89.05</td>
</tr>
<tr>
<td>Dozat and Manning (2017)</td>
<td>95.74</td>
<td>94.08</td>
<td>89.30</td>
</tr>
<tr>
<td>Li et al. (2018a)</td>
<td>94.11</td>
<td>92.08</td>
<td>88.78</td>
</tr>
<tr>
<td>Ma et al. (2018)</td>
<td>95.87</td>
<td>94.19</td>
<td>90.59</td>
</tr>
<tr>
<td>Our (Division)</td>
<td>94.32</td>
<td>93.09</td>
<td>89.14</td>
</tr>
<tr>
<td>Our (Joint)</td>
<td><strong>96.09</strong></td>
<td><strong>94.68</strong></td>
<td><strong>91.21</strong></td>
</tr>
<tr>
<td>Our (Division*)</td>
<td>-</td>
<td>-</td>
<td>91.69</td>
</tr>
<tr>
<td>Our (Joint*)</td>
<td>-</td>
<td>-</td>
<td>93.24</td>
</tr>
</tbody>
</table>

Zhou and Zhao (2019)
Takeaways

- Dependency formalism provides an alternative to constituency, particularly useful in how portable it is across languages.

- Dependency parsing also has efficient dynamic programs for inference.

- CRFs + neural CRFs (again) work well.
Proj 1 Results

Jiaming Chen: 82.46 F1

Po-Yi Chen: 82.02 F1

Ting-Yu Yen: 81.57 F1

Prakhar Singh: 81.54 F1

All others <81

- WordPair features, larger window for POS tag extraction ([-2, 2])
- Also larger window and data shuffling in between epochs
- Unregularized Adagrad worked best
- City gazetteer, generic date recognizer