CS388: Natural Language Processing
Lecture 11: Dependency Parsing I

Greg Durrett

Administrivia

- Project 1 graded by Tuesday
- Survey results:
  - Some annoyances from projects: slow debugging/training, etc.
  - If you have comments on the code, please send them to me (either anonymously or non-anonymously)
  - Bit rate
  - Clearer slides/notation

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 argmax P(T|x) = argmax 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 -> Y Z to build X in every possible way

He wrote a long report on Mars
Outline

- Discriminative constituency parsing
- Dependency representation, contrast with constituency
- Projectivity
- Graph-based dependency parsers

Discriminative Parsers

CRF Parsing

\[
\text{score}
\begin{pmatrix}
\text{NP} \\
\text{NP} \\
\text{PP}
\end{pmatrix}
= w^T f
\begin{pmatrix}
\text{NP} \\
\text{2} \\
\text{NP} \\
\text{5} \\
\text{PP} \\
\text{7}
\end{pmatrix}
\]

He wrote a long report on Mars

Left child last word = report \wedge \text{NP} \text{NP} \text{PP}

- Can learn that we report [PP], which is common due to reporting on things
- Can “neuralize” this as well like neural CRFs for NER

Joint Discrete and Continuous Parsing

Chart remains discrete!

Parsing a sentence:
- Feedforward pass on nets
- Discrete feature computation
- Run CKY dynamic program

Durrett and Klein (ACL 2015)
Neural CRF Parsing

- Simpler version: score *constituents* rather than rule applications

\[
\text{score}(\text{NP}) = w^T f(\text{He wrote a long report on Mars})
\]

- Use BiLSTMs to compute embeddings of each word, embeddings at edge of span characterize that span
- 91-93 F1, 95 F1 with ELMo (SOTA).
  Great on other langs too!

Stern et al. (2017), Kitaev et al. (2018)

---

Dependency Representation

---

Lexicalized 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)

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)
ConsKtuency:
ternary rule NP -> NP CC NP

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

Universal Dependencies

- Annotate dependencies with the same representation in many languages

Projectivity

- Any subtree is a contiguous span of the sentence <-> tree is projective
- Hunden jagades av katten
- Hund in houses and cats
- [dogs in houses] and cats
- NN dog to NN house

http://universaldependencies.org/
Projectivity

- Projective <-> no “crossing” arcs

- Crossing arcs:
  - dogs in houses and cats
  - the dog ran to the house

- Swiss-German has famous non-context-free constructions

Projectivity in other languages

- Many trees in other languages are nonprojective

<table>
<thead>
<tr>
<th>Formalism</th>
<th>Arabic</th>
<th>Czech</th>
<th>Danish</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Endpoint-Crossing</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>
</tbody>
</table>

- Some other formalisms (that are harder to parse in), most useful one is 1-Endpoint-Crossing
Graph-Based Parsing

Defining Dependency Graphs

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

$$\text{ROOT} \quad \text{the} \quad \text{dog} \quad \text{ran} \quad \text{to} \quad \text{the} \quad \text{house}$$

Generalizing CKY

- Score matrix with three dimensions: start, end, and head, start <= head < end
- new score = score(2, 5, 4) + score(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

$$\text{wrote} \quad \text{a} \quad \text{long} \quad \text{report} \quad \text{on} \quad \text{Mars}$$

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

$$\text{ROOT} \quad \text{DT} \quad \text{NN} \quad \text{VBD} \quad \text{TO} \quad \text{DT} \quad \text{NN}$$
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

\[ + + = \text{ incomplete span} \]

```
ROOT    DT    NN    VBD    TO    DT    NN
the     the  dog   ran    to    the  house
```

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

```
ROOT    DT    NN    VBD    TO    DT    NN
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:

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

Building Systems

- 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
- McDonald et al. (2005) — conjunctions of parent and child words + POS, POS of words in between, POS of surrounding words
- \( \text{HEAD} = \text{TO} \& \text{MOD} = \text{NN} \)
- \( \text{HEAD} = \text{TO} \& \text{MOD} = \text{house} \)
- \( \text{HEAD} = \text{TO} \& \text{MOD-1} = \text{the} \)
- \( \text{ARC} \_ \text{CROSSES} = \text{DT} \)

Higher-Order Parsing

- Track additional state during parsing so we can look at “grandparents” (and siblings). \( O(n^3) \) dynamic program or use approximate search

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

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: 95-96 UAS

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