Dependency Parsing (3)

CMSC 470
Marine Carpuat

Fig credits: Joakim Nivre, Dan Jurafsky & James Martin
Dependency Parsing: what you should know

• Transition-based dependency parsing
  • Shift-reduce parsing
  • Transition systems: arc standard, arc eager
  • Oracle algorithm: how to obtain a transition sequence given a tree
  • How to construct a multiclass classifier to predict parsing actions
  • What transition-based parsers can and cannot do
  • That transition-based parsers provide a flexible framework that allows many extensions
    • such as RNNs vs feature engineering, non-projectivity (but I don’t expect you to memorize these algorithms)

• Next: Graph-based dependency parsing
# Generating Training Examples

- What we have in a treebank
- What we need to train an oracle
  - Pairs of configurations and predicted parsing action

## Figure 14.8
Generating training items consisting of configuration/predicted action pairs by simulating a parse with a given reference parse.
Generating training examples

• Approach: simulate parsing to generate reference trees

• Given
  • A current config with stack S, dependency relations Rc
  • A reference parse (V,Rp)

• Do

  \[
  \text{LEFTARC}(r): \text{if } (S_1 \; r \; S_2) \in R_p
  \]

  \[
  \text{RIGHTARC}(r): \text{if } (S_2 \; r \; S_1) \in R_p \text{ and } \forall r', w \text{ s.t. } (S_1 \; r' \; w) \in R_p \text{ then } (S_1 \; r' \; w) \in R_c
  \]

  \[
  \text{SHIFT}: \text{otherwise}
  \]
Graph-based Dependency Parsing
Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph $G = (V, A)$, is a subgraph $G' = (V', A')$ such that:
  - $V' = V$
  - $A' \subseteq A$, and $|A'| = |V'| - 1$
  - $G'$ is a tree (acyclic)

- A spanning tree of the following (multi-)digraphs

![Directed Graphs](image.png)
Dependency Parsing as Finding the Maximum Spanning Tree

• Views parsing as finding the best directed spanning tree
  • of multi-digraph that captures all possible dependencies in a sentence
  • needs a score that quantifies how good a tree is

• Assume we have an arc factored model
  i.e. weight of graph can be factored as sum or product of weights of its arcs

• Chu-Liu-Edmonds algorithm can find the maximum spanning tree for us
  • Recursive algorithm
  • Naïve implementation: $O(n^3)$
Chu-Liu-Edmonds illustrated (for unlabeled dependency parsing)
Chu-Liu-Edmonds illustrated

- Find highest scoring incoming arc for each vertex

- If this is a tree, then we have found MST!!
Chu-Liu-Edmonds illustrated

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle
Chu-Liu-Edmonds illustrated

- Outgoing arc weights
  - Equal to the max of outgoing arc over all vertexes in cycle
  - e.g., John → Mary is 3 and saw → Mary is 30
Chu-Liu-Edmonds illustrated

- Incoming arc weights
  - Equal to the weight of best spanning tree that includes head of incoming arc, and all nodes in cycle
  - root → saw → John is 40 (**)  
  - root → John → saw is 29
This is a tree and the MST for the contracted graph!!

Go back up recursive call and reconstruct final graph
Arc weights as linear classifiers

\[ w_{ij}^k = e^{w \cdot f(i,j,k)} \]

- Arc weights are a linear combination of features of the arc, \( f \), and a corresponding weight vector \( w \).
- Raised to an exponent (simplifies some math ...)
- What arc features?
Example of classifier features

- Features from [McDonald et al. 2005]:
  - Identities of the words $w_i$ and $w_j$ and the label $l_k$

head = saw & dependent = with
Typical classifier features

• Word forms, lemmas, and parts of speech of the headword and its dependent
• Corresponding features derived from the contexts before, after and between the words
• Word embeddings
• The dependency relation itself
• The direction of the relation (to the right or left)
• The distance from the head to the dependent
• …
How to score a graph $G$ using features?

**Arc-factored model assumption**

$$
G = \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} w_{ij}^k = \arg\max_{G \in T(G_x)} \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}
$$

**By definition of arc weights as linear classifiers**

$$
= \arg\max_{G \in T(G_x)} \log \prod_{(i,j,k) \in G} e^{w \cdot f(i,j,k)}
$$

$$
= \arg\max_{G \in T(G_x)} \sum_{(i,j,k) \in G} w \cdot f(i,j,k)
$$

$$
= \arg\max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) = \arg\max_{G \in T(G_x)} w \cdot f(G)
$$
Learning parameters with the Structured Perceptron

Training data: \( \mathcal{T} = \{ (x_t, G_t) \}_{t=1}^{\mathcal{T}} \)

1. \( w^{(0)} = 0; \quad i = 0 \)
2. for \( n : 1..N \)
3. \quad for \( t : 1..T \)
4. \quad Let \( G' = \arg \max_{G'} w^{(i)} \cdot f(G') \)
5. \quad if \( G' \neq G_t \)
6. \quad \quad \quad \quad w^{(i+1)} = w^{(i)} + f(G_t) - f(G') \)
7. \quad \quad \quad \quad i = i + 1
8. return \( w^i \)
Dependency parsing algorithms

**Transition-based**
- Locally trained
- Use greedy search algorithms
- Define features over a rich history of parsing decisions

**Graph-based**
- Globally trained
- Use exact (or near exact) search algorithms
- Define features over a limited history of parsing decisions
Dependency Parsing: what you should know

• Interpreting dependency trees

• Transition-based dependency parsing
  • Shift-reduce parsing
  • Transition system: arc standard, arc eager
  • Oracle
  • Learning/predicting parsing actions

• Graph-based dependency parsing

• A flexible framework that allows many extensions
  • RNNs vs feature engineering, non-projectivity