Dependency grammar and dependency parsing

Natural Language Processing (5LN710)

2012-12-13

Marco Kuhlmann
Department of Linguistics and Philology
Dependency grammar
Dependency grammar

• The term ‘dependency grammar’ does not refer to a specific grammar formalism.
• Rather, it refers to a specific way to describe the syntactic structure of a sentence.
The notion of dependency

• The basic observation behind *constituency* is that groups of words may act as one unit.

  *Example:* noun phrase, prepositional phrase

• The basic observation behind *dependency* is that words have grammatical functions with respect to other words in the sentence.

  *Example:* subject, modifier
In an arc $h \rightarrow d$, the word $h$ is called the head, and the word $d$ is called the dependent.

The arcs form a rooted tree.
The history of dependency grammar

• The notion of dependency can be found in some of the earliest formal grammars.

• Modern dependency grammar is attributed to Lucien Tesnière (1893–1954).

• Recent years have seen a revived interest in dependency-based description of natural language syntax.
Linguistic resources

• Descriptive dependency grammars exist for some natural languages.

• Dependency treebanks exist for a wide range of natural languages.

• These treebanks can be used to train accurate and efficient dependency parsers.
Dependency parsing
Ambiguity

Just like phrase structure parsing, dependency parsing has to deal with ambiguity.

I booked a flight from LA
Just like phrase structure parsing, dependency parsing has to deal with ambiguity.

I booked a flight from LA
Disambiguation

• We need to disambiguate between alternative analyses.

• We develop mechanisms for scoring dependency trees, and disambiguate by choosing a dependency tree with the highest score.
Scoring models and parsing algorithms

Distinguish two aspects:

• **Scoring model:**
  How do we want to score dependency trees?

• **Parsing algorithm:**
  How do we compute a highest-scoring dependency tree under the given scoring model?
The arc-factored model

- To score a dependency tree, score the individual arcs, and combine the score into a simple sum.

\[ \text{score}(t) = \text{score}(a_1) + \ldots + \text{score}(a_n) \]

- Define the score of an arc \( h \rightarrow d \) as the weighted sum of all features of that arc:

\[ \text{score}(h \rightarrow d) = f_1w_1 + \ldots + f_nw_n \]
Examples of features

- ‘The head is a verb.’
- ‘The dependent is a noun.’
- ‘The head is a verb and the dependent is a noun.’
- ‘The head is a verb and the predecessor of the head is a pronoun.’
- ‘The arc goes from left to right.’
- ‘The arc has length 2.’
Training using structured prediction

- Take a sentence $w$ and a gold-standard dependency tree $g$ for $w$.
- Compute the highest-scoring dependency tree under the current weights; call it $p$.
- Increase the weights of all features that are in $g$ but not in $p$.
- Decrease the weights of all features that are in $p$ but not in $g$. 
Arc-factored dependency parsing

Parsing algorithms

• Collins’ algorithm
• Eisner’s algorithm
Collins’ algorithm

Signatures

\[[\text{min}, \text{max}, C]\]
Signatures

Collins’ algorithm

[min, max, root]
Collins’ algorithm

Initialization

I booked a flight from LA

[0, 1, I] [1, 2, booked] [2, 3, a] [3, 4, flight] [4, 5, from LA]
Collins’ algorithm

Adding a left-to-right arc

I booked a flight from LA

[3, 4, flight] [4, 5, from LA]
Collins’ algorithm

Adding a left-to-right arc

1 booked a flight from LA

\[ [3, 5, \text{flight}] \]
Collins’ algorithm

Adding a left-to-right arc
Collins’ algorithm

Adding a left-to-right arc

\[ \text{score}(t) = \text{score}(t_1) + \text{score}(t_2) + \text{score}(l \rightarrow r) \]
Collins’ algorithm

Adding a left-to-right arc

for each [min, max] with max - min > 1 do

for each l from min to max - 2 do

    double best = score[min][max][l]

for each r from l + 1 to max - 1 do

    for each mid from l + 1 to r do

        t₁ = score[min][mid][l]

        t₂ = score[mid][max][r]

    double current = t₁ + t₂ + score(l → r)

    if current > best then

        best = current

        score[min][max][l] = best
Collins’ algorithm

Adding a right-to-left arc

\begin{center}
\begin{tikzpicture}
\node (l) at (0,0) {$l$};
\node (r) at (4,0) {$r$};
\node (t1) at (1,2) {$t_1$};
\node (t2) at (3,2) {$t_2$};
\draw (l) -- (t1);
\draw (r) -- (t2);
\draw (t1) -- (t2);
\end{tikzpicture}
\end{center}

\text{Min} \quad \text{Mid} \quad \text{Max}
Complexity analysis

- **Space requirement:**
  \[ O(|w|^3) \]

- **Runtime requirement:**
  \[ O(|w|^5) \]
In Collins’ algorithm, adding a left-to-right arc is done in one single step, specified by 5 positions.
In Collins’ algorithm, adding a left-to-right arc is done in one single step, specified by 5 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
In Eisner’s algorithm, the same thing is done in three steps, each one specified by 3 positions.
Transition-based dependency parsing

- Eisner’s algorithm runs in time $O(|w|^3)$. This may be too much if a lot of data is involved.

- **Idea:** Design a dumber but really fast algorithm and let the machine learning do the rest.

- Eisner’s algorithm searches over many different dependency trees at the same time.

- A transition-based dependency parser only builds *one* tree, in *one* left-to-right sweep over the input.
The parser starts in an initial configuration.

At each step, it asks a guide to choose between one of several transitions (actions) into new configurations.

Parsing stops if the parser reaches a terminal configuration.

The parser returns the dependency tree associated with the terminal configuration.
Configuration \( c = \text{parser}.\text{getInitialConfiguration}(\text{sentence}) \)

while \( c \) is not a terminal configuration do

    Transition \( t = \text{guide}.\text{getNextTransition}(c) \)

    \( c = c.\text{makeTransition}(t) \)

return \( c.\text{getGraph()} \)
Guides

• We need a guide that tells us what the next transition should be.

• The task of the guide can be understood as classification: Predict the next transition (class), given the current configuration.
Training a guide

• We let the parser run on gold-standard trees.
• Every time there is a choice to make, we simply look into the tree and do ‘the right thing’™.
• We collect all (configuration, transition) pairs and train a classifier on them.
• When parsing unseen sentences, we use the trained classifier as a guide.
• The number of (configuration, transition) pairs is far too large.

• We define a set of features of configurations that we consider to be relevant for the task of predicting the next transition.

  *Example*: word forms of the topmost two words on the stack and the next two words in the buffer

• We can then describe every configuration in terms of a feature vector.
Training a guide

Transition-based dependency parsing

configurations in which we want to do la

configurations in which we want to do ra

score for feature 1

score for feature 2
Training a guide

Transition-based dependency parsing

Montag, 10. Dezember 12
Training a guide

• In practical systems, we have thousands of features and hundreds of transitions.

• There are several machine-learning paradigms that can be used to train a guide for such a task.

  Examples: perceptron, decision trees, support-vector machines
The arc-standard algorithm

• The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.
• It is very similar to shift–reduce parsing as it is known for context-free grammars.
• It is implemented in most practical transition-based dependency parsers, including MaltParser.
Configurations

A configuration for a sentence \( w = w_1 \ldots w_n \) consists of three components:

- a **buffer** containing words of \( w \)
- a **stack** containing words of \( w \)
- the **dependency graph** constructed so far
Configurations

• **Initial configuration:**
  • All words are in the buffer.
  • The stack is empty.
  • The dependency graph is empty.

• **Terminal configuration:**
  • The buffer is empty.
  • The stack contains a single word.
The arc-standard algorithm

Possible transitions

- **shift (sh):** push
  the next word in the buffer onto the stack

- **left-arc (la):** add an arc
  from the topmost word on the stack, $s_1$, to the second-topmost word, $s_2$, and pop $s_2$

- **right-arc (ra):** add an arc
  from the second-topmost word on the stack, $s_2$, to the topmost word, $s_1$, and pop $s_1$
The arc-standard algorithm

Example run

Stack

Buffer

I booked a flight from LA

sh
The arc-standard algorithm

Example run

Stack

Buffer

booked  a  flight  from LA

I booked a flight from LA

sh
Example run

Stack

Buffer

I booked a flight from LA

la-subj
The arc-standard algorithm

Example run

Stack

Buffer

subj

l booked a flight from LA

sh
The arc-standard algorithm

Example run

Stack

Buffer

<table>
<thead>
<tr>
<th>booked</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>flight</td>
<td>from LA</td>
</tr>
</tbody>
</table>

I booked a flight from LA

sh
The arc-standard algorithm

Example run

Stack

Buffer

booked  a  flight

from LA

subj

I  booked  a  flight  from LA

la-det
The arc-standard algorithm

Example run

Stack

Buffer

subj

I booked

det

a flight from LA

sh

from LA
The arc-standard algorithm

Example run

Stack

Buffer

booked  flight  from LA

ra-pmod

Montag, 10. Dezember 12
The arc-standard algorithm

Example run

Stack

Buffer

ra-dobj

I booked a flight from LA

subj

det

pmod

Montag, 10. Dezember 12
The arc-standard algorithm

**Example run**

Stack: booked

Buffer:

```
  subj  
  I    booked
  dobj
    det
      a
      flight
      pmod
        from LA
```

done!
Summary

- Two approaches to dependency parsing:
  - graph-based dependency parsing
  - transition-based dependency parsing
- Two different feature representations:
  - score graph parts (such as individual arcs)
  - score (configuration, transition) pairs