

CG168 Week 1 homework

Mark Johnson

10th September 2008

Shift-reduce dependency parsing

Nivre et al (2006) propose a shift-reduce dependency parsing that works as follows.

A state of the parser is a triple $(\mathbf{s}, j, \mathbf{h})$, where \mathbf{s} is a stack of indices into the words $\mathbf{w} = (w_1, \dots, w_n)$ to be parsed, j is an integer in $1, \dots, n + 1$ indicating the position of the next word to be read in \mathbf{w} , and $\mathbf{h} \in (0, \dots, n)^n$ (i.e., n integers between 0 and n) is a vector of *governing head* indices.

The *initial state* is $(\epsilon, 1, \mathbf{0})$, i.e., the stack \mathbf{s} is empty, the first word is the next word to be read, and $\mathbf{h} = \mathbf{0}$ is the vector of n zeros.

The *final state* is $((i), n + 1, \mathbf{h})$, i.e., where the stack contains a single index i and all words in \mathbf{w} have been read. In the final state \mathbf{h} indicates the governing head of each word, i.e., w_{h_i} is the governing head of w_i .

The moves of the parser are:

- *Left-arc* : $(\mathbf{s}|i, j, \mathbf{h}) \mapsto (\mathbf{s}, j, \mathbf{h}[i \rightarrow j])$; precondition $j \leq n, h_i = 0$.
- *Right-arc* : $(\mathbf{s}|i, j, \mathbf{h}) \mapsto (\mathbf{s}|i|j, j + 1, \mathbf{h}[j \rightarrow i])$; precondition $j \leq n, h_j = 0$.
- *Reduce* : $(\mathbf{s}|i, j, \mathbf{h}) \mapsto (\mathbf{s}, j, \mathbf{h})$; precondition $h_i \neq 0$.
- *Shift* : $(\mathbf{s}, j, \mathbf{h}) \mapsto (\mathbf{s}|j, j + 1, \mathbf{h})$

Here $\mathbf{s}|i|j$ denotes a stack whose top element is j , which is on top of i , which is in turn on top of the elements \mathbf{s} , and $\mathbf{h}[j \rightarrow i]$ is a vector just like \mathbf{h} except that $h_j = i$.

Predicting the next move

The shift-reduce dependency parser is non-deterministic, i.e., at many states several different moves are possible. An *oracle* is a function from states plus the word string \mathbf{w} and the *part of speech tag* string \mathbf{t} to moves.

Given a state $x = (\mathbf{s}, j, \mathbf{h})$ and words \mathbf{w} and POS tags \mathbf{t} , over the next few weeks you will construct classifiers that serve as *oracles*, i.e., that predict the next move, based on the following context information (Nivre 2006, page 107).

$$\mathbf{c}(x) = (t_{s_2}, t_{s_1}, t_j, t_{j+1}, t_{j+2}, t_{j+3}, w_{h_{s_1}}, w_{s_1}, w_j, w_{j+1})$$

That is, $\mathbf{c}(x)$ consists of the POS tags of the two top stack entries and the four next words in the input, the governing head word of the top stack entry, and the words of the two top stack entries and the next two words of the input. If any of these don't exist (e.g., because the stack doesn't contain two elements), then the corresponding element in $\mathbf{c}(x)$ is the special null value “_”.

Mapping parses to moves

The data to train these classifiers will come from parsed corpora. Given a “gold parse” represented by a governing head vector \mathbf{g} , we need to find a sequence of moves \mathbf{m} that generates \mathbf{h} . Here’s an algorithm sketch that maps states $x = (\mathbf{s}, j, \mathbf{h})$ to moves that will ultimately derive \mathbf{g} :

perform a *Left-arc* or *Right-arc* if possible and compatible with \mathbf{g}
else if there is an $s \in \mathbf{s}$ such that $g_j = s$ or $g_s = j$ then *Reduce*
else *Shift*

The CONLL data files

You can find these on the CS machines in `/courses/cog168/asgn/data/` as `train.conll`, `dev.conll` and `test.conll`. Each of these contains information about one word per line, with sentences separated by blank lines. The second column contains the word, the fourth column its part of speech and the seventh column gives the index of its governing head. For example, the second sentence in `train.conll` is represented as:

```
1 Ms.      - NNP NNP - 2 - 2 -
2 Haag     - NNP NNP - 3 - 3 -
3 plays    - VBZ VBZ - 0 - 0 -
4 Elianti  - NNP NNP - 3 - 3 -
5 .        - .   .   - 3 - 3 -
```

Homework:

1. Write a program that maps a CONLL “gold parse” \mathbf{h} to a sequence of parser moves that generates \mathbf{h} . This program should read the CONLL data files and write a sequence of moves that generates \mathbf{h} , as well as the context generated from the state immediately preceding each move.

For example, the second sentence in `train.conll`, which is “Ms. Haag plays Elianti .” and has the gold parse $\mathbf{h} = (2, 3, 0, 3, 3)$, would have the corresponding sequence of moves and contexts.

```
shift - - NNP NNP VBZ NNP - - Ms. Haag
left  - NNP NNP VBZ NNP . - Ms. Haag plays
shift - - NNP VBZ NNP . - - Haag plays
left  - NNP VBZ NNP . - - Haag plays Elianti
shift - - VBZ NNP . - - - plays Elianti
right - VBZ NNP . - - - plays Elianti .
reduce VBZ NNP . - - - plays Elianti . -
right  - VBZ . - - - - plays . -
reduce VBZ . - - - - plays . - -
```

Please turn in your sequence of moves and contexts for the first sentence of `train.conll`. (Please separate fields by TAB characters, sentences by blank lines, and omit unnecessary whitespace).

2. Write a program that takes a sequence of moves and returns the corresponding governing heads vector \mathbf{h} . Apply your program to the output of the program in (1), and check that the governing heads vector it produces is the same as the “gold” vector for every sentence in `train.conll` and `dev.conll`.
3. (200-level) Are there two different sequences of moves that generate the same governing head vector? If so, give a pair that produces the same governing head vector, otherwise, explain why there is no such pair. Why might this be important in our application?