Dependency Parsing exercises: Implementation of a transition-based parser (part 2)

Deadline for sub-exercises 1–3: 14.06.2021
Deadline for sub-exercises 4–6: 17.06.2021

Please send completed solutions to evang@hhu.de with subject "dependency homework" and attachment parser.py.

Matthew Honnibal writes: "One of the reasons parser.py is so fast is that it does unlabelled parsing. Based on previous experiments, a labelled parser would likely be about 40x slower, and about 1% more accurate. Adapting the program to labelled parsing would be a good exercise for the reader, if you have access to the data." This is exactly what we are going to do in this exercise. Note that in my own experiments, I found the labeled parser to be only about 5x slower than the unlabeled one.

You can use the example solution to exercise 8 as the basis for your work.

There are a number of things that need to be modified for labeled parsing:

1. The `main_train` function (called `main` in the original code) outputs the unlabeled attachment score (UAS) that the final trained model achieves on the validation data. The UAS is the proportion of words (here: not counting punctuation) that have the correct head. Adapt the function to *also* output labeled attachment score (LAS). The LAS is the proportion of words (here: not counting punctuation) that have the correct head *and* the correct label.

2. Similarly, adapt the `train` function and the `Parser.train_one` method to measure and output not only UAS, but also LAS on the training data after each iteration.

3. After making only the above changes, how high is the LAS that the parser achieves? Why?

4. Change the parser so that it actually creates labeled edges.

   (a) To create labeled edges, the parser needs a different set of transitions. Instead of just `SHIFT`, `LEFT`, and `RIGHT`, it needs `SHIFT`, `LEFT-nsubj`, `RIGHT-nsubj`, `LEFT-advmod`, `RIGHT-advmod` and so on, one "left" and one "right" action for every dependency label. Extract the list of labels from the training data and adapt the `MOVES` attribute accordingly. It should still be a list of integers. Also create helper functions `is_left`, `is_right` and `label` that enable the parser to look up for a given transition whether it is a "left" or "right" transition, and which label it has, if any.

   (b) Optional exercise for extra credit: instead of hard-coding `MOVES` and the additional data structures, make it so that the parser determines the list of moves dynamically depending on the labels that occur in the training data, and stores them in attributes of `Parser` objects, rather than module attributes. Note that this information then has to be stored along with the saved model so future invocations of the parser can use it.

   (c) Adapt the `transition` function to use the new set of transitions and create labeled edges. Use the helper functions you created.

   (d) Adapt the functions `get_valid_moves` and `get_gold_moves` (the dynamic oracle) to use the new set of transitions. Hint: the logic of the dynamic oracle stays mostly the same. However, make sure to mark *all* "left" and "right" transitions as costly when they are wrong. Also, when returning a "right" or "left" transition as optimal, make sure that is has the correct label.

(e) Adapt the `main_parse` function to output labels as well as heads.

5. Test the parser. Train it for 1 instead of 15 iterations to save time. On my laptop, this takes about 15 minutes. UAS on the validation set should be about 86.6, and LAS 82.6.

6. Optional exercise for extra credit: why do you think the labeled parser has higher UAS than the unlabeled parser (after one iteration)?