Lab 3

Status: Completed
Your identity is visible during marking.
Marks: 7 / 7

Submission deadline: 30 Aug 2019 23:55, 45 days left

Transition-based dependency parsing

This assignment is inspired by one developed by Joakim Nivre and Sara Stymne at Uppsala University. Updated by Yves Scherrer, minor updates: Kathrein Abu Kwaik.

In this assignment, you are going to implement some key components of a transition-based dependency parser and analyze its behaviour. The file transition_parser.py contains some starter code. You are expected to modify and add code in this file only.

If you want to read more about the arc-eager system and static and dynamic oracles, we recommend this paper. Reading the paper is **not** a requirement for solving the lab.

1 Implement and test the arc-eager transition system for parsing

A transition system for dependency parsing consists of a set of **configurations** and a set of **transitions** for moving between configurations. You are going to implement the **arc-eager** transition system, where configurations consist of a **stack**, a **buffer** and an **arc set**, all represented as lists. Tokens are represented by integers corresponding to sentence positions (with 0 for the special root node), and dependency arcs are represented as triples (h, d, l), where his the head token, d the dependent token, and l the dependency label.

In this part of the assignment, you are going to implement an algorithm for parsing a sentence using the arc-eager transition system. Algorithm 1 defines the general structure of the parsing algorithm; the definitions and actions of the four transition types can be looked up in the course slides. You are expected to complete the parse_arc_eager() procedure according to this algorithm.

Algorithm 1 PARSER-ARC-EAGER (sentence, transitions)

1. initialize configuration: stack, buffer, arcs

2. **while** configuration is not final **do**
    1. *t* = pop(transitions)
    2. *i* = top element of the stack
    3. *j* = first element of the buffer
    4. **if** *t* = SHIFT **then**
        1. move *j* from the buffer to the stack
    5. **else if** *t* = ... **then**
        1. ...
    6. **return** arcs

The procedure test_parse() defines an example sentence and its corresponding transition sequence. Test your parser using this test procedure. The expected output looks as follows:

```
cat ---det--> the
sits ---nsubj--> cat mat ---det--> the
mat ---case--> on
sits ---nmod--> mat sits ---advmod--> today
```

2 Implement an oracle based on the arc-eager transition system

In the first part, you have implemented an algorithm that produces a dependency tree from a sentence and a list of transitions. In this second part, you should implement the opposite, i.e. an algorithm that produces the transitions from a dependency tree. This type of algorithm is called an **oracle**. The algorithm for the arc-eager oracle looks as follows:

Algorithm 2 ARC-EAGER-ORACLE (stack, buffer, head, deprel)

1. *i* = top element of the stack
2. *j* = first element of the buffer
3. **if** *head*(*i*) = *j* **then**
    1. **return** LEFT-ARC-*deprel*(*i*)

4. **else if** *head*(j) = i **then**
    1. **return** RIGHT-ARC-*deprel*(j)
5. **else if** $\exists k \in$ stack(*head*(j) = k $\vee$ *head*(k) = j) **then**
    1. **return** REDUCE
6. **else**
    1. **return** SHIFT

Note that *head* and *deprel* are functions which return the head or the dependency relation of a given word. This information is available in the parsed sentence. Complete the procedure arc_eager_oracle() with Algorithm 2.

The procedure test_oracle() defines an example sentence with its dependency tree, and three example configurations. The expected output is as follows:

```
SHIFT
LEFT-ARC-det
RIGHT-ARC-nmod
```

3 Parsing with an oracle

The parsing algorithm (Algorithm 1) can be modified to use the oracle prediction instead of the predefined list of transitions:

Algorithm 3 PARSE-ARC-EAGER (sentence)

1. initialize configuration: stack, buffer, arcs
2. **while** configuration is not final **do**
    1. t = oracle(stack, buffer, sentence)
    2. i = top element of the stack
    3. j = first element of the buffer
    4. **if** t = SHIFT **then**
        1. move j from the buffer to the stack
    5. **else if** t = ... **then**

```
            1. ...
        6. return arcs
```

The code in parse_arc_eager() combines both algorithms: if a transition list is given, it is used, otherwise, the oracle is called. Test this combination using test_parse_oracle(). The arcs predicted by the parsing algorithm should be the same as those of the original dependency tree.

The expected output is as follows:

```
Predicted arcs:
dog ---det--> the
sat ---nsubj--> dog
couch ---det--> the
couch ---case--> on
sat ---nmod--> couch
sat ---advmod--> yesterday

Original arcs:
dog ---det--> the
sat ---nsubj--> dog
couch ---case--> on
couch ---det--> the
sat ---nmod--> couch
sat ---advmod--> yesterday
```

Note: This part of the assignment may look circular: given a dependency tree, the oracle predicts transitions, and given these transitions, the parser predicts a dependency tree, which happens to look exactly the same as the one we have started with. This is partially a result of making this assignment self-contained. In real-world scenarios, there are two disconnected processes:

1. The test_parse_oracle() procedure is used for producing training data for the classifier. Such training data associates configurations with the selected transitions and parsing are required to produce the configurations. In this setting, we are interested in the configurations and transitions, not in the arcs.
2. For testing, a procedure similar to test_parse_oracle() is used, but the transitions are predicted by the classifier instead of the oracle. The oracle procedure would not work, as the dependency relations are not available in this setting. Here, we are interested in the dependency arcs, not in the configurations and transitions.

Hemanth Kumar Battula , 9 Jan 2019 17:15

File name: lab3-battula.py (4,3 KB)

Status set to: To be marked

Chatrine Qwaider , 18 Jan 2019 12:26

Status set to: Completed

Grade set to: VG

Comment: You have done a great job.
Your answer is well organized and easy to follow and understand.
Keep going. Your programming skills are very good.

Chatrine Qwaider , 25 Jan 2019 15:28

Mark set to: 7