NLP HW-4

Meili Liu

October 25, 2022

1 try a state-of-art parse in English

(a)interesting or surprising about the style of trees produced by the parser
we tried the Barkeley Neural Parser, whose output is quite similar with our parser, but with fancier output form, shown on figure 1.

(b.1)some sentence that the parser got wrong
1. Fruit files like a banana. we got wrong parse structure
2. Time flies like an arrow. we got right parse structure
The result is shown on figure 2.
For the sentence "Fruit files like a banana.". The Barkeley Neural Parser thinks that "files" is a verb, just like files in sencond sentence. However, we know that fruit cannot fly. so the parser structure is wrong.

(b.2)some hard sentence that the parser managed to get right
" It 's very real , otherwise we would n't be doing it , " this official said .
the result is shown on figure 3.
the sentence is from wallstreet corpus, which may be the corpus of the parser. so the parser works pretty well.

(c)"adversarial" sentences
One morning I shot an elephant in my pajamas.
this sentence is quite famous, and the parse result in shown on Figure. 4.
The result is obviously wrong, because the elephant is too big to be in a person's pajamas. So the correct sentence meaning should be "One morning I shot an elephant [while I was wearing] my pajamas."

2 probabilistic Earley parser

(a) before adding an item to the parse table, check in o(1) time whether another copy is alreay there

This can be done by duplicate check in *Agenda.push*() function. The *push* function is responsible for adding new item into the *chart*. For a new item, first check $if\ item\ not\ in self.\_index$:, if the answer is $False$, then we can add new item safely, but if the answer is $True$, we then compare the weight of old item with new item, and update the old item if the new item has less weight.

(b) only have o(1) to add the item to the appropriate column

This is easy in Python, just append the new item to $self.\_items$, and also remember to append $self.\_backpointerandself.\_weight$ for this item, and modify $self.\_index$
If the new item is a duplicate but with less weight, after updating the old item, we append the index of the updated item in to a reprocessing buffer. So we can reprocess it again.
all of these operations can be done in constant time.

(c) For each item in the parse chart, you must keep track of that item's current best parse and

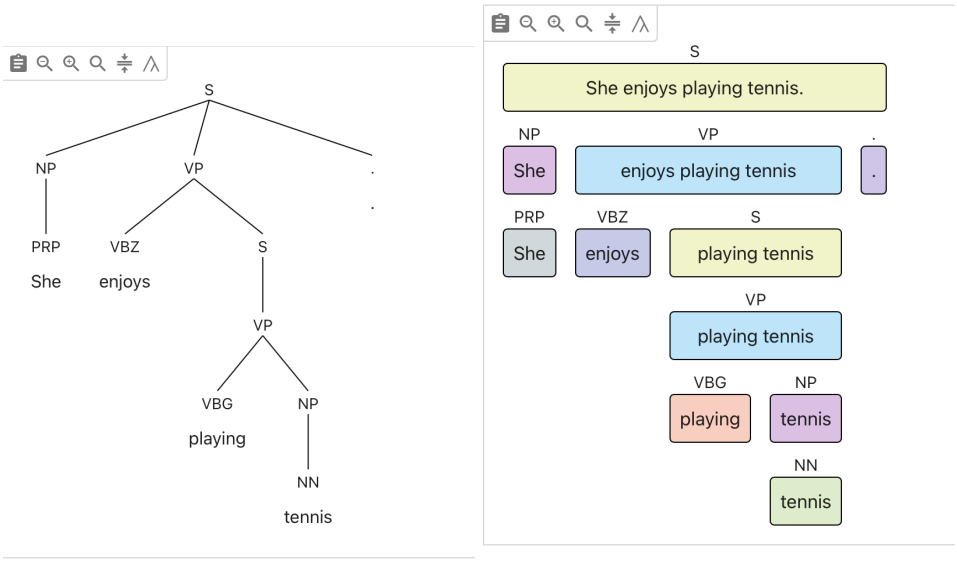

Figure 1: output of Barkeley Neural Parser

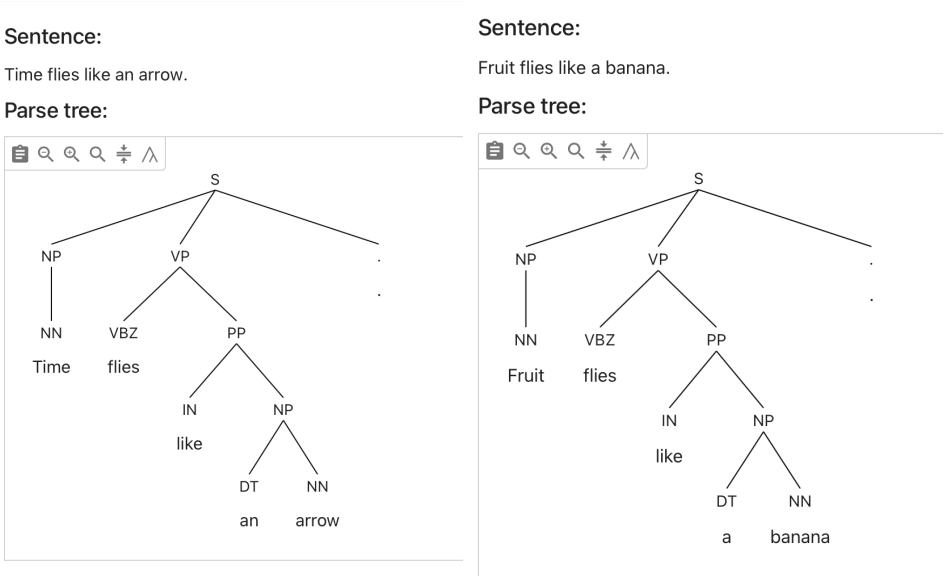

Figure 2: wrong output of Barkeley Neural Parser

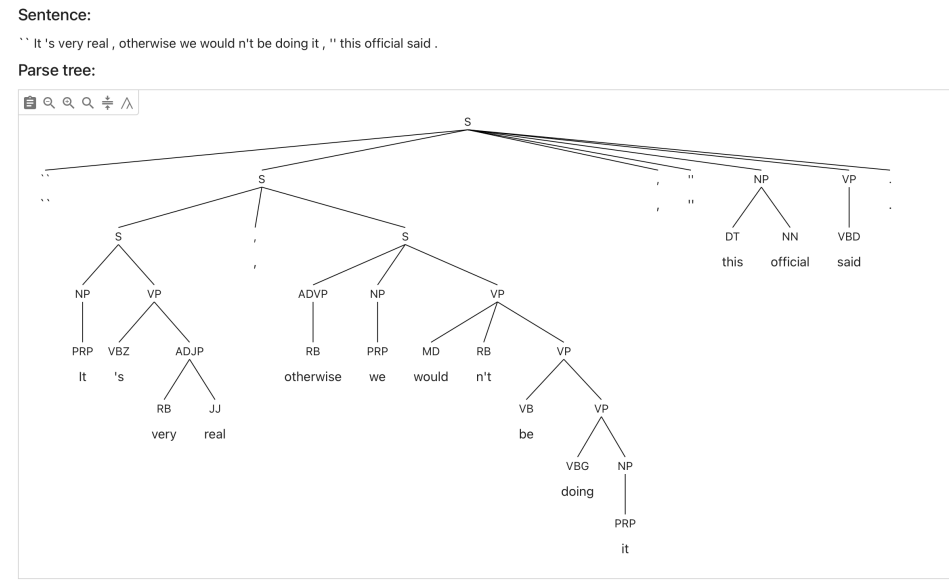

Figure 3: the output of hard sentence for Barkeley Neural Parser

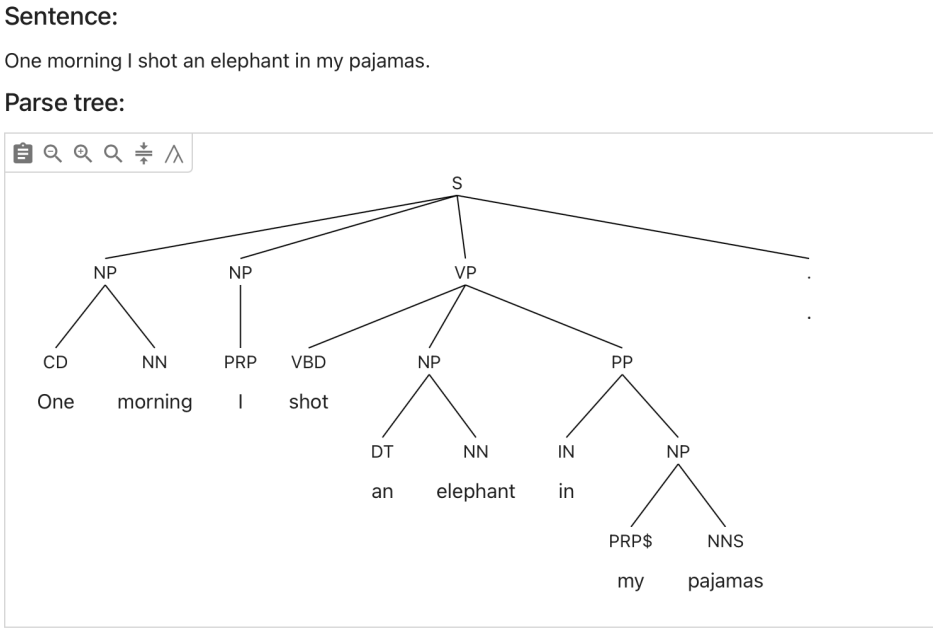

Figure 4: adversarial output of Barkeley Neural Parser

the total weight of that best parse. These values may have to be updated if you find a better parse for that item
we add two more element in $self.\_backpointerandself.\_weight$, which can be used to store the item's best parse and total weight. Each time we update a old item with less weight, we also update the $self.\_backpointerandself.\_weight$.

3 speed up parser

1. we've implement 2 speedup method, batch deplicate check and left-corner filtering

2. estimate how much speedup you got on short sentences
on sentence "The very biggest companies are not likely to go under ."
running time for $parse.py$ 271.23s
running time for $parse2.py$ 23.46ss
From the above result, we can see that the computation time has been reduced to 1/10 after employing speedup methods