Accelerated Natural Language Processing 2018

Lecture 14: CKY parsing, probabilistic CF-PSGs

Henry S. Thompson
16 October 2018

1. Visualising CKY parsing: The empty chart

Just the empty matrix

	1	2	3	4
0				
1				
2				
3				
	the	frogs	ate	fish

2. Visualising the Chart (0,1)

Unary branching rules: Det → the

	1	2	3	4
0	Det			
1				
2				
3				
	the	frogs	ate	fish

3. Visualising the Chart (1,2)

Unary branching rules: N → frogs, Nom → N, NP → Nom

	1	2	3	4
0	Det			
1		N Nom NP		
2				
3				
	the	frogs	ate	fish

4. Visualising the Chart (2,3)

Unary branching rules: TV → ate

	1	2	3	4
0	Det			
1		N Nom NP		
2			TV	
3				
	the	frogs	ate	fish

5. Visualising the Chart (3,4)

Unary branching rules: N → fish, Nom → N, NP → Nom, IV → fish, VP → IV

	1	2	3	4
0	Det			
1		N Nom NP		
2			TV	
3				N IV Nom VP NP
	the	frogs	ate	fish

6. Visualising the Chart (0,2)

Binary branching rule: NP → Det Nom

- **(0,1) & (1,2) ⇒ (0,2)**

	1	2	3	4
0	Det	NP		
1		N Nom NP		
2			TV	
3				N IV Nom VP NP
	the	frogs	ate	fish

7. Visualising the Chart (1,3)

- (1,2) & (2,3) ¬⇒

	1	2	3	4
0	Det	NP		
1		N Nom NP		
2			TV	
3				N IV Nom VP NP
	the	frogs	ate	fish

8. Visualising the Chart (2,4)

Binary branching rule: VP → TV NP

- (2,3) & (3,4) ⇒ (2,4)

	1	2	3	4
0	Det	NP		
1		N Nom NP		
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

9. Visualising the Chart (0,3)

(0,1) & (1,3) ¬⇒

	1	2	3	4
0	Det	NP		
1		N Nom NP		
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

(0,2) & (2,3) ¬⇒

	1	2	3	4
0	Det	NP		
1		N Nom NP		
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

10. Visualising the Chart (1,4)

Binary branching rule: S → NP VP

(1,3) & (3,4) ¬⇒

	1	2	3	4
0	Det	NP		
1		N Nom NP		
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

(1,2) & (2,4) ⇒ (1,4)

	1	2	3	4
0	Det	NP		
1		N Nom NP		S
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

11. Visualising the Chart (0,4)

Binary branching rule: S → NP VP

(0,1) & (1,4) ¬⇒

	1	2	3	4
0	Det	NP		
1		N Nom NP		S
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

(0,3) & (3,4) ¬⇒

	1	2	3	4
0	Det	NP		
1		N Nom NP		S
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

(0,2) & (2,4) ⇒ (0,4)

	1	2	3	4
0	Det	NP		S
1		N Nom NP		S
2			TV	VP
3				N IV Nom VP NP
	the	frogs	ate	fish

12. From CKY Recogniser to CKY Parser

We cannot tell from the CKY chart as specificed, the syntactic analysis of the input string.

We just have a chart **recogniser**, a way of determining whether a string belongs to the language generated by the grammar.

Changing this to a **parser** requires recording which existing constituents were combined to make each new constituent. This requires another field to record the one or more ways in which a constituent spanning (i,j) can be made from constituents spanning (i,k) and (k,j).

13. Ambiguity is still the problem

Features help express the grammar neatly, but they don't change the ambiguity problem

Active chart parsing gives us flexibility, but that doesn't solve the ambiguity problem

Examples of ambiguity:

global

PP attachment

> “One morning I shot an elephant in my pyjamas ...” (Groucho Marx)

- Was the elephant in Groucho's pyjamas?!
- No, Groucho was *wearing* the pyjamas
- Contrast "We flew home in a big jet" with "We were upgraded to seats in business class"

gerundive VP attachment

> We saw penguins flying over Antarctica

- Who was flying, us or the penguins?
- Contrast "We saw penguins carrying fish in their beaks" with "We saw penguins using high-powered binoculars"

coordination

> hot tea and coffee

vs

> empty bottles and fag-ends

The most likely readings are that both beverages are hot, but only the bottles are empty

14. Compositional semantics, again

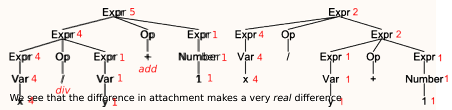
The significance of attachment ambiguity is clear when we look at semantics

Back to Python arithmetic expressions, with an even simpler grammar:

```
Expr → Expr Op Expr | Var | Number
Op → '+' | '-' | '*' | '/'
```

Assuming some more work to **tokenise** the input, this will give us two analyses for the string `"x/y+1"`:

If we associate the appropriate computation with each production
- (skipping a lot of details here, some of which will be covered in a few weeks)
- And assuming that `x` is `4` and `y` is `1`



We see that the difference in attachment makes a very *real* difference

We call that kind of semantic computation a **compositional** one

> "The meaning of the whole [constituent] is a function of the meaning of the parts [children]"

15. Compositional semantics, cont'd

How does this relate to natural language?

- Many approaches to language *understanding* proceed in a similar way
  - In what's called a **rule-to-rule** way
  - Associating a (compositional) *semantic* rule with each *syntactic rule*

So, for example, for the kind of PP attachment ambiguity we keep encountering

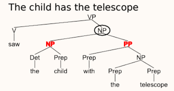
- Where the PP is attached determines where its semantic contribution is made

In other words, with respect to "saw the child with the telescope"

VP → V NP, NP → NP PP

The semantics of the PP contributes to the semantics of the NP

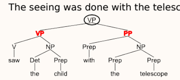
- The child has the telescope



VP → VP PP

The semantics of the PP contributes to the semantics of the (higher) VP

- The seeing was done with the telescope



16. Back to ambiguity

We need a way to choose the best analysis from among many

- *Very* many
- The average sentence in the Wall Street Journal dataset we will use in the labs this week is just under *26* words long
- The exponential consequence of multiple local ambiguities given a broad-coverage CF-PSG
- Will be 1000s, if not 100s of 1000s, of parses

And we need a sound basis for ranking these

A gold standard provides at least partial solutions. . .

17. Treebanks: big investment, big reward

Mitch Marcus at the Univ. of Pennsylvania took this seriously

- As did the US DARPA (Defense Advanced Research Projects Agency)
  - In the context of their evaluation-led research funding approach

The Penn Treebank project was launched in 1989

- Over a number of phases it has annotated *many* datasets, including
    - the Brown Corpus, *Wall Street Journal* archives, air-travel request data, phone conversation data
- and has spawned many follow-ons for other languages

18. Treebank examples and evolution

The first release contained 'skeletal' parses, that is, simple syntactic trees

- With a few quirks based on a choice of underlying grammatical theory
- Now somewhat dated, this mostly involves the use of what are called **traces** to indicate 'missing' material
- For example in relative clauses and some complement clauses
    - "I liked the show that we watched * last night"
    - "Robin found it difficult to * lift the boxes"

Here's an example:

```
(S
  (NP
    (ADJP Battle-tested Japanese industrial)
     managers)
  here always
  (VP buck up
     (NP nervous newcomers)
     (PP with
        (NP the tale
           (PP of
              (NP the
                 (ADJP first
                    (PP of
                       (NP their countrymen)))
                 (S (NP *)
                    to
                    (VP visit
                       (NP Mexico))))))))
```

This annotation was produced by manually correcting the output of a simple chunking parser, then removing the POS-tags and some NP-internal structure

19. Penn Treebank, mark two

The second release added much more information, including some hyphenated functional indications:

```
(S (NP-SBJ (NP Battle-tested Japanese industrial managers)
           (ADVP-LOC here))
   (ADVP-TMP always)
   (VP buck
       (PRT up)
       (NP nervous newcomers)
       (PP-CLR with
               (NP (NP the tale)
                   (PP of
                       (NP (NP the
                               first)
                           (PP of
                               (NP their countrymen))
                           (SBAR (WHNP-1 0)
                                 (S (NP-SBJ *T*-1)
                                    (VP to
                                        (VP visit
                                            (NP Mexico))))))))))
```

The functional markers here are

SBJ Subject argument
LOC Locative modifier
TMP Temporal modifier
CLR Sort-of argument

20. Deriving a grammar from a treebank

Trivial, really

For every node in every tree with non-lexical children

- Add a rule to the grammar (or increment the count of an existing rule)
- **NodeLab → Child1Lab Child2Lab** . . .

So, for the tree in the previous slide, we'd get e.g.

NP-SBJ → NP ADVP-LOC
NP → NP PP
NP → NP PP SBAR

The result is guaranteed to give at least one parse to every sentence in the Treebank

21. Simple Probabilistic CF-PSGs

Given a treebank, we can easily compute a simple kind of probabilistic grammar

- Either directly with respect to treebank-derived rules
- Or by mapping from treebank statistics to some other ruleset

For a treebank-derived ruleset, the maximum likelihood probability estimate is simple:

$$P(\mathrm{NT} \rightarrow C_1 C_2 \ldots C_n \mid \mathrm{NT}) = \frac{\mathrm{count}(\mathrm{NT} \rightarrow C_1 C_2 \ldots C_n)}{\mathrm{count}(\mathrm{NT})}$$

So, for example, given that in the NLTK treebank subset, there are 52041 subtrees labelled **S**

And in 29201 of these there are exactly two children of **S**, labelled **NP-SBJ** followed by **VP**, we get

$$P(S \rightarrow \text{NP-SBJ VP} \mid S) = \frac{29201}{52041} = 0.56$$

The probability of a whole parse tree is then just the product of the probabilities of every non-terminal node in that tree

- Or we can use the sum of the costs for simpler calculations
    - Recall that **costs** are negative log probabilities
    - We sum them, instead of multiplying
    - And prefer lower costs (== higher probabilities)

It's not hard to modify a chart parser to use probabilities to actually guide parsing

- So that you get the most likely parse first
- By maintaining the list of pending alternatives in sorted order
    - Instead of strictly LIFO or FIFO
- And always taking the lowest-cost entry to put in the chart next

22. Evaluation against a treebank

No-one's grammatical theory/detailed grammar is going to give exactly the same results as a treebank

- Unless it's *derived* automatically *from* the treebank
- And even then there may well be equi-probable alternatives at some points

So how do you evaluate a parser against a treebank?

It turns out just looking at major constituent boundaries is surprisingly good

- Basically, because most parsers are pretty *bad*

The same (D)ARPA push towards evaluation-driven funding drove the development of the **PARSEVAL** metric

PARSEVAL just compares bracketings, without regard to labels

- In its simplest form, it just counts parenthesis-crossing
    - (A (B C)) vs. ((A B) C)
    - Fewer is better
- And constituent recall
    - (A B C) vs. (A (B C))
    - More is better

Each of these is effectively penalising an attachment mistake

23. PARSEVAL examples

(a child with a telescope)

will be penalised wrt

(a child (with a telescope))

- by the second rule above, because of the mismatch in number of sub-constituents

((hot tea) and coffee)

will be penalised wrt

(hot (tea and coffee))

- because of the crossing parentheses

24. Treebank grammar problems

The bad news: just using the treebank subset that ships with NLTK, there are 1798 different expansions for **S**

- Some are very common, such as S → NP-SBJ VP
- Most occur only once (Zipf again)

As always, there's a tradeoff between specificity and statistical significance

- More detailed symbols
    - means more symbols
    - means fewer examples of each

For getting good PARSEVAL scores

- learned-from-treebank grammars tended to collapse many of the release-2 categories back to something more like their release-1 counterparts