Minimalist Grammars in Computational Syntax

Kendra Chalkley

A discussion of Stanojevic and Stabler's 'A Sound and Complete Left-Corner Parsing for Minimalist Grammars' (ACL 2018) and Torr's PHD presentation 'Broad Coverage Statistical Parsing with Minimalist Grammars' (2016) w.r.t previous coursework

Intro

This paper summarizes a few developments in the use of theoretical linguistics' minimalist syntactic theory in computational representations of and operations on language and how these developments relate to other papers this course has covered this quarter. Minimalist developments in computational linguistics include 1. formalization of a computation friendly representation of minimalist theory 2. development of a minimalist dependency parser and (Stanojević and Stabler) 3. creation of a minimalist grammar training set, MG bank. Here, I'll start by reviewing each of these developments, and in the case of the minimalist dependency parser, comparing this parser to one by Ballesteros et al, which was read, but not yet reviewed for this course, and conclude by relating this work to a few other topics covered earlier in the quarter.

Computation-friendly minimalism

A minimalist grammar defines an acceptor for a mildly context sensitive language. A complete grammar can be represented with 4 sets: V, the vocabulary or set of non syntactic features, generally represented as words; Cat, the set of syntactic features including selectees, selectors, licensees and licensors; Lex, a lexicon including all valid combinations of V and Cat; and F, the set of rules by which other parts of the grammar interact. Consider the example MG which follows:

- V: *helped, who, Jack, has, [int]*
- Cat:
    - Selectees(x): V, D, T, C

    - Selectors (=X, X=): D= =D, V=, T=
    - Licensees (-y): -wh, -case
    - Licensors (+y): +wh, +case
- Lex:
    - *helped*:: D= =D V
    - *who*:: D -wh
    - *Jack*:: D -case
    - *has*:: V= +case T
    - *[int]*:: T= +wh C
- Rules:
    - Merge(x,y)
    - Move(x)

We'll return to more thorough definition of merge and move momentarily. For now, notice the words of the lexicon. 'Helped', a transitive verb, selects two determiner phrases, one to its left (D= subject) and one to the right (=D object) and is itself a verb selectee. 'Who' is a determiner selectee with a -wh feature which must be licenced, and similarly 'Jack' has a -case feature. 'Has' licenses a verb to its right and is a +case selector that is itself a Tense word, and the interrogative '[int]' is a C which licences tense and carries a +wh feature. This grammar accepts the phrase "who Jack has helped," according to John Torr's review slides available here. How one would derive "who has Jack helped" is not immediately obvious, but should depend on some variation in the case licensing within the lexicon.

Note, John Torr's slides differ from Stabler's review of MG at the beginning of his 'A Sound and Complete Left-Corner Parsing for Minimalist Grammars" in that Stabler does not bother with the difference between =D and D= and manages the changing word orders differently (Stanojević and Stabler). This is an important topic to review for more clarity on the source of the difference in the future. Another difference is that Stabler includes a start selectee category, which Torr forgets or ignores in the top of his

example tree. In the big picture, minimalist grammars motivate argument relationships with selection and licencing, and I refer the reader to Stabler's work for further exploration of the details.

In addition to the lexicon and its composition, there are two fundamental rules of MGs governing the derivation of phrases, merge and move. At a high level, merge is the process by which new lexical items are introduced to the tree based on selection, and move is an operation by which elements of the tree so far re-merge (forgive the wording which is not entirely accurate) with other elements of the tree to check licensing features. In any grammar, multiple more specific formulations of merge and move are required to fully cover all situations in which these rules can apply. Below are a few examples of Stabler's merge and move rules to illustrate how the rules are specified (Stanojević and Stabler).

- Merge3 (one item, s, selects another, t, with unchecked licensee features =F, t merges with s and the selection is resolved, but the items remain separate, as t will continue to move up the tree to check its remaining features (d)):
    - s: y, a_1, ... a_k, t: d, i_1, ... i_l <-
        - s **=F** y, a_1, ... a_k t **F**d, i_1, ... i_l
- Move1 (final move of t, so its -f chain is eliminated, it looks like this rule has an off by 1 error, because a_i is missing, but it doesn't seem important.):
    - ts : y, a_1, ... a_{i-1}, a_{i+1}, ... a_k <- s: +fya_1,...a_{i-1}, t: -f, a_{i+1}, ... , a_k

Given a fully specified lexicon and a small set of merge and move rules, we should be able to derive complete, minimalist parse trees. The Stabler paper does so with a left corner parser.

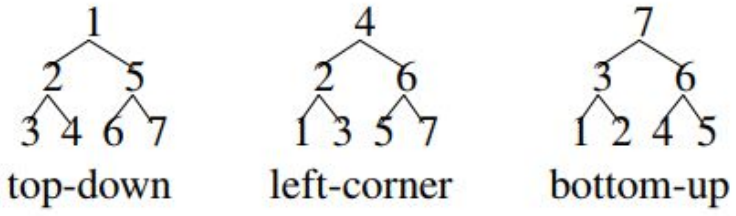
Left-corner minimalist parser

The summary of top, bottom, and mixed parsers given in Stabler's introduction is tough to improve upon. He says:

> "The literature presents bottom-up and top-down parsers for MGs (Harkema, 2001b), which differ in the order in which derivations are constructed, and

consequently they may differ in their memory demands at each point in the parse. But partly because of those memory demands, parsers that mix top-down and bottom-up steps are often regarded as psycholinguistically more plausible (Hale, 2014; Resnik, 1992; Abney and Johnson, 1991).

"Among mixed strategies, left-corner parsing (LC) is perhaps the best known (Rosenkrantz and Lewis, 1970). A left-corner parser does not begin by guessing what's in the string, as a top-down parser does. But it also does not just reduce elements of the input, as a bottom-up parser does. A left-corner parser [first completes the left-most constituent of a string, bottom up] and then predict[s] the sisters of that element (top-down), if any. The following CFG trees have nodes numbered in the order they would be constructed by bottom-up, left-corner and top-down strategies:



" (Stanojević and Stabler, 1)

Left-corner parsing starts from the left of the string and moves up the hierarchy until it's able to complete the first constituent for the leftmost item in the string. It then moves on to finish the next order constituent until it reads the rightmost end of the string. The minimalist aspect of Stabler's parser is that there are only two possible labels at each node of the derived tree, merge or move. Refer to section 3 of the Stabler paper for the complete structure of the parsing algorithm. For now, an example from the beginning of the parse of the phrase "Aca knows what Bebe likes" is below.

- Lexicon:
    - *Aca*:: D
    - *Bibi*::D
    - *what* :: D, -wh

    - *likes*:: =D, =D, V
    - *knows*:: =C, =D, V
    - '':: =V, C
    - '':: =V, +wh, C
- Step 1: shift the empty C head which selects a V onto the queue.
- Step 2: Using merge 1, change the first element of the queue to reflect that merging a V with this node will give a C with the same chain sequence.
- Step 3: shift the first element of the string 'Aca' onto the queue, into the space between 0 and 1.
- Step 3: merge the two elements in the queue. Producing:
    - '1-_:=D V _M -> 0-_C_M'
    - which states that a D selecting verb with a chain _M will fill positions 1-? And be headed by a C with the same chain in position 0.
- Step 4 and 5: shift 'knows' onto the queue and merge it,
    - This satisfies the need for a verb with a free =D slot, fitting it in the 1 position, and our tree now needs only the C licensed by knows, resolving to our original C in the 0 position.
    - 2-_.C_M -> 0-_:C _M
- so forth

The process itself can be difficult to follow, but the relationship between the resulting structures and the long distance, movement dependent relationships in theoretical linguistics is a promising step in the direction of coordination between computation and linguistic theory

MG Bank

To actually use this algorithm on data, we'll need detailed data sets with complete lexicons and preferably reference parses to train on and compare against. Fortunately, work has started on this. John Torr started his PHD project on the construction of MG bank, a MG based version of Penn Treebank. Unfortunately, his work is not yet

complete and as such there are no practical application papers yet available to evaluate the usability of such parsers.

I personally suspect that the lexicon may be the most valuable part of the finished MG bank because the process of identifying the features necessary for these minimalist durations is the most complex and arguably most meaningful part of the minimalist structures. Further, in these papers the semantic and phonological features of the vocabulary are currently represented as words. A move to learning these sub-components of words would further nudge NLP in the direction of linguistic theory and could result in a much more robust and multilingual natural language representation. (Un)fortunately that work still remains to be done.

Relation to other readings

So far this quarter I've read LSTM related papers that varied widely between the theoretical and the practical. We've covered LSTM language recognition power both purely theoretically as well as with certain memory and compute limitations, as well as a method of extracting DFAs from RNNs trained to recognize regular languages (Weiss, Goldberg, and Yahav). On the more practical side of things, the Ballsteros et al paper described an LSTM based dependency parser, and Kadar et al investigated patterns learned by an LSTM trained on image captioning.

The final topic of the quarter includes a theoretical basis for minimalist dependency parser, the implementation of which is incredibly practical in that it applies actual linguistic theory to computational syntax. For this system to be used, an even more practical a dataset for minimalist training parses is currently under development: MG Bank.

Of course, these papers have more in common than their relationship on a theory/implementation hierarchy. We started the quarter by talking about the theoretical ability of RNNs to learn languages throughout Chomsky's language hierarchy from regular languages to turing completeness. One of the great benefits of minimalist grammars is that they are a mildly context sensitive way of representing natural language, which suggests a real possibility that RNN architectures are theoretically able

to lean them fully.

Beyond that, we've also discussed, though unfortunately not summarized in this essay collection, a paper by Ballesteros et al which describes an LSTM implementation of a standard dependency parser. The Ballesteros paper is several steps ahead of minimalist grammar parsing, in that it uses datasets of arc transition datasets to train and test performance of a particular LSTM configuration in learning to analyse dependencies, cross linguistically. To do such a project with minimalist grammar, one would first need not only a complete MG bank, but banks in each other language. A further difference between the two models is dependency relation labels themselves. Labels in the arc system include part of speech information between every pair of related words. By contrast, such information is largely contained in the lexicon and chains/histories of the minimalist grammar.

We've also talked about Kadar et al's 'Representation of Linguistic Form and Function' which analyzed the importance of any given word to a neural representation of an image caption and further investigated the correlation between that score and the related dependency label(s). While Kadar's paper may have suffered some design flaws based on the nature of image caption language, replacing arc standard dependencies with minimalist relationships and features would be an interesting extension of the project.

In all, the incorporation of minimalist linguistic theory to computational syntax is a promising direction for the field, with many exciting applications ahead.

Biblios

Direct Citations:

Ballesteros, Miguel, et al. "Greedy transition-based dependency parsing with stack lstms." *Computational Linguistics* 43.2 (2017): 311-347.

Kádár, Akos, Grzegorz Chrupała, and Afra Alishahi. "Representation of linguistic form and function in recurrent neural networks." *Computational Linguistics* 43.4 (2017): 761-780.

Stanojević, Miloš, and Edward Stabler. "A Sound and Complete Left-Corner Parsing for Minimalist Grammars." *Proceedings of the Eight Workshop on Cognitive Aspects of Computational Language Learning and Processing*. 2018.

Torr, John. "Broad Coverage Statistical Parsing with Minimalist Grammars." http://homepages.inf.ed.ac.uk/s1344326/presentations/1stYearReviewPresentation.pdf (Presented 2016, accessed 2019)

Weiss, Gail, Yoav Goldberg, and Eran Yahav. "On the practical computational power of finite precision RNNs for language recognition." *arXiv preprint arXiv:1805.04908* (2018).

Citations of Stabler's Citations:

John T. Hale. 2014. Automaton Theories of Human Sentence Comprehension. CSLI, Stanford.

Henk Harkema. 2001a. A characterization of minimalist languages. In P. de Groote, G. Morrill, and C. Retor´e, editors, Logical Aspects of Computational Linguistics. Springer, NY, LNCS 2099, pages 193–211

Philip Resnik. 1992. Left-corner parsing and psychological plausibility. In Proceedings of the 14th International Conference on Computational Linguistics, COLING 92. pages 191–197.

D. J. Rosenkrantz and P. M. Lewis. 1970. Deterministic left corner parsing. In IEEE Conference

Record of the 11th Annual Symposium on Switching and Automata Theory. pages 139–152.