Proceedings of the Society for Computation in Linguistics

Volume 4 Article 34

2021

# CCG Supertagging as Top-down Tree Generation

Jakob Prange
*Georgetown University*, jp1724@georgetown.edu

Nathan Schneider
*Georgetown University*, nathan.schneider@georgetown.edu

Vivek Srikumar
*University of Utah*, svivek@cs.utah.edu

Follow this and additional works at: https://scholarworks.umass.edu/scil

Part of the Computational Linguistics Commons

## Recommended Citation

Prange, Jakob; Schneider, Nathan; and Srikumar, Vivek (2021) "CCG Supertagging as Top-down Tree Generation," *Proceedings of the Society for Computation in Linguistics*: Vol. 4 , Article 34.
Available at: https://scholarworks.umass.edu/scil/vol4/iss1/34

This Abstract is brought to you for free and open access by ScholarWorks@UMass Amherst. It has been accepted for inclusion in Proceedings of the Society for Computation in Linguistics by an authorized editor of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.

# CCG Supertagging as Top-down Tree Generation

**Jakob Prange** **Nathan Schneider**
Georgetown University
{jp1724, nathan.schneider}@georgetown.edu

**Vivek Srikumar**
University of Utah
svivek@cs.utah.edu

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a strongly-lexicalized grammar formalism, whose syntax-semantics interface has been attractive for downstream tasks such as semantic parsing and machine translation. Most CCG parsers operate as a pipeline whose first task is 'supertagging', i.e., sequence labeling with a large search space of complex tags. Given these supertags, all that remains to parsing is applying general rules of (binary) combination between adjacent constituents until the entire input is covered. Supertagging thus represents the crux of the overall parsing process.

A CCG supertag consists of atomic categories like S and NP, which are related by slashes to (recursively) form functional categories. By convention, the infix-notation (S\NP)/NP is equivalent to the tree in fig. 1a where the right child of any slash is the argument, and the left child is the result of combining the category with its argument. Slash directionality determines the linear order of combination. E.g., the transitive verb category (S\NP)/NP expects two noun phrases (to the right and left) to form a clause (S). But this flexibility leads to infinite possible supertags; in practice, they follow a power law distribution. CCG treebanks contain numerous rare supertags, including several that occur only in the test sets. Still others can be expected to occur in a much larger corpus. This *long tail* of the distribution is particularly challenging for taggers, due to its sparseness and relatively high complexity of categories.

In most previous work, CCG supertaggers have skirted this problem by treating categories as a fixed set of opaque labels (fig. 1b) and ignoring those occurring fewer than a certain threshold (following Clark, 2002). Conversely, Kogkalidis et al. (2019) and Bhargava and Penn (2020) have recently proposed different methods of *constructive* supertagging, where supertags are constructed as sequences of minimal pieces and there is no constraint that predicted supertags must be known (fig. 1c).

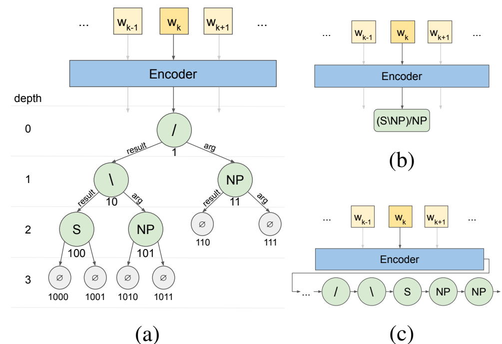


Figure 1: Schematic of our tree-structured supertagger (left) in contrast with unstructured (top right) and sequential (bottom right) models.

We take this idea one step further by introducing ***tree-structured*** **constructive supertagging** (Prange et al., 2021):[1] Given a sequence of words, we generate each word's supertag as a tree, from the top down (fig. 1a). At the $t^{th}$ step, the model greedily chooses the most likely node labels at depth $t$, conditioned on the word encoding and the ancestors predicted so far. The first decision ($t = 0$) is either an atomic category, or the main functor. In the latter case, the model then moves on to select the argument and result types, which may be atomic categories or functors themselves. We are thus guaranteed to always generate well-formed categories (as opposed to sequence generators, which can *learn* to predict properly structured outputs but are *not guaranteed* to always do so).

**Modeling.** All supertagging models we compare consist of a sequence encoder (we finetune RoBERTa-base, Liu et al., 2019), an **output-positional encoder**, an attention layer over the sequence encoder, and a fully-connected 2-layer per-

[1]This abstract is a condensed version of Prange et al. (2021). For more details on modeling, data, and analysis, we refer to the full paper.



*Proceedings of the Society for Computation in Linguistics (SCiL) 2021*, pages 351-354.
Held on-line February 14-19, 2021

ceptron (MLP) with a final softmax layer which maps hidden representations to output probability distributions. We use the term *(output) position* to refer to any atomic part of a category for which a labeling decision has to be made. This could be, for example, the positions of the `S` category in figs. 1a and 1c, or the single output in fig. 1b. We experiment with two alternative ways of deriving the hidden state $\mathbf{h}_{k,i}$ for position $i$ within the category of word $k$: a tree-structured recursive neural network (**TreeRNN**) and a deterministic addressing function that accesses each node directly (**AddrMLP**). For the latter, each node in a category's tree representation is addressed by a sequence of bits corresponding to a top-down traversal of the tree. E.g., in fig. 1a, the inner `NP` argument (the argument of the top-level result) is addressed as 101. The sequence of slashes in a node's ancestors ([/, \] for the inner NP in fig. 1a) is mapped to a binary vector in a similar way. We then use a single linear layer to project these features into the encoder's hidden space before adding it to the word's contextualized encoding.

**Data.** A limitation of standard CCG evaluation datasets is that they contain very few tokens of categories seen less than 10 times in training. Thus, scores computed over these small samples may not reliably estimate the models' generalization capacity. To correct for this, we investigate what happens if the models are trained on sentences containing exclusively the higher-frequency (≥10) categories, and evaluated only on sentences with at least one rare category. We split the (English) CCG Rebank training set (WSJ sections 02–21; Honnibal et al., 2010) in this way.

**Baselines.** We compare our TreeRNN and AddrMLP models to the following baselines: 1) **Nonconstructive** (MLP): We compute the output probabilities for complete categories directly from the encoder's hidden state. 2) **Sequential**: Kogkalidis et al. (2019) construct type-logical supertags by generating for each sentence a single sequence of atomic types and functors. We adapt their implementation of the sequence-to-sequence Transformer model (Vaswani et al., 2017) to our problem ("K+19"). We also implement a simplified version of Bhargava and Penn's (2020) tagger, where each word's supertag is generated separately by a GRU ("RNN").

**Findings.** Table 1 shows our results. The nonconstructive baseline performs well on frequent

| | All | ≥100 | 10–99 | 1–9 | 0 |
|---|---|---|---|---|---|
| Tokens | $n$=53,765 | $n$=50,754 | $n$=989 | $n$=292 | $n$=1,730 |
| Types | $N$=1,351 | $N$=188 | $N$=240 | $N$=118 | $N$=805 |
| **Nonconstructive** | | | | | |
| MLP | 88.79 | **92.87** | **55.61** | 19.29 | – |
| **Sequential** | | | | | |
| K+19 | 80.20 | 83.49 | 47.72 | 25.11 | **11.62** |
| RNN | 88.73 | 92.64 | 52.92 | 23.52 | 5.38 |
| **Tree-structured** | | | | | |
| TreeRNN | 88.78 | 92.54 | 49.90 | 20.55 | 9.62 |
| AddrMLP | **89.01** | 92.70 | 54.03 | **26.48** | 10.96 |

Table 1: Accuracy on our redistributed Rebank evaluation set (avg. over 3 runs). Scores are computed for bins based on the order of magnitude of occurrences of categories in training.

categories but suffers severely from sparsity on the long tail and has no chance of generating unseen tags. Constructive taggers are always required to make multiple atomic decisions whenever assigning a complex category, all of which need to be correct in order for the full category to be counted as correct. More complex categories tend to be rarer and thus are more difficult than simple ones in general, for all models.

Surprisingly however, it is not dramatically more difficult for constructive systems to generate complex categories than it is for nonconstructive systems to simply assign them. Both types of models often only err in a single decision for an atomic category or slash, rather than misinterpreting the entire syntactic structure. And even the few cases of structural divergences between prediction and ground truth tend to be systematic and consistent with other predictions in the same sentence.[2]

The sequence-to-sequence model by K+19 does a lot better than the MLP on the tail and even retrieves some unseen categories, but at the cost of frequent ones, most likely due to the lack of hard alignments between words and tags. The tag-wise recurrent and tree-recursive generators (RNN and TreeRNN) come close to the nonconstructive classifiers, but do not convincingly improve over them. The AddrMLP model, finally, performs competitively across all frequency bands, proving its robustness in terms of overall accuracy. It is also the computationally most lightweight model in that it has the fewest parameters out of all the models, and its inference speed is comparable to that of the nonconstructive taggers.

[2]See appendix A for supporting figures and examples.

## References


Aditya Bhargava and Gerald Penn. 2020. Supertagging with CCG primitives. In *Proc. of RepL4NLP*, pages 194–204, Online.

Stephen Clark. 2002. Supertagging for Combinatory Categorial Grammar. In *Proc. of TAG+*, pages 19–24, Universitá di Venezia.

Matthew Honnibal, James R. Curran, and Johan Bos. 2010. Rebanking CCGbank for improved NP interpretation. In *Proc. of ACL*, pages 207–215, Uppsala, Sweden.

Konstantinos Kogkalidis, Michael Moortgat, and Tejaswini Deoskar. 2019. Constructive type-logical supertagging with self-attention networks. In *Proc. of RepL4NLP*, pages 113–123, Florence, Italy.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *Preprint: arXiv:1907.11692*.

Jakob Prange, Nathan Schneider, and Vivek Srikumar. 2021. Supertagging the long tail with tree-structured decoding of complex categories. *TACL*. Preprint: arXiv:2012.01285.

Mark Steedman. 2000. *The Syntatic Process*. MIT Press, Cambridge, MA, USA.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NeurIPS*, pages 5998–6008, Long Beach, CA, USA.


## A Extended Analysis and Examples

In Table 2 we quantify the structural and labeling errors. A substantial portion of erroneous categories actually do have the correct *structure* (✓struct).[3] For these cases, we perform a detailed error analysis, whose results we present in fig. 2. In fact, if the structure is correct, the predicted category is often only off by the direction of a single slash or the attribute of a single atomic category. K+19 additionally struggles with atomic decisions beyond just differences in attributes.

By manually searching the corpus, we find that even in the cases where a tagger assigns a category with an incorrect structure, there are systematic confusions such as between argument and adjunct PPs and between fixed particle verbs and (aspectual) adjunct particles. This is difficult to measure at a large scale, but we present two examples in Tables 3 and 4.

| | | Incorrect | | |
|---|---|---|---|---|
| **Model** | **Correct** | ✓struct | ✓formed | ✗formed |
| MLP | 47,552 | 1,401 | 4,811 | – |
| K+19 | 43,120 | 2,706 | 7,812 | 127 |
| RNN | 47,704 | 1,395 | 4,661 | 5 |
| TreeRNN | 47,733 | 1,373 | 4,659 | 1 |
| AddrMLP | 47,851 | 1,352 | 4,562 | 1 |

Table 2: Analysis of predicted supertag structures in the redistributed evaluation set. Incorrect predictions are broken down in terms of having the correct structure (✓struct: the same number and arrangement of slashes, arguments, and results as the gold category), an incorrect but well-formed structure (✓formed: diverging arrangement of arguments, but valid tree structure), or an invalid structure (✗formed, e.g., missing arguments).

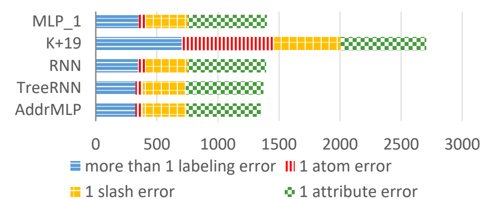


Figure 2: Fine-grained analysis of correctly-structured but incorrectly labeled predictions ('✓struct' in Table 2). 'Attribute error' means that the predicted atomic category is correct except for a wrong or missing linguistic attribute (e.g., `S` vs. `S[dcl]`); 'atom error' means that an entirely wrong atomic category has been chosen (e.g., `PP` vs. `NP`); and 'slash error' means confusing / and \.

[3]E.g., for "piling" in Table 4 the RNN predicts `(S[ng]\NP)/PP`, which exhibits the correct structure `(X\X)/X` with an incorrect atomic label (`PP` instead of `PR`).

| | garnered | from | 1984 to 1986 |
|---|---|---|---|
| **Gold** | **(S[pss]\NP)** | **(ADV/ADV)/NP** | |
| MLP | ✓ | ✓ | |
| K+19 | ✓ | ✓ | |
| RNN | ✓ | ✓ | |
| AddrMLP | (S[pss]\NP)/PP | (PP/ADV)/NP | |

Table 3: AddrMLP treats "garnered" as expecting a PP argument (which would be correct for a source-PP, e.g. "garnered information from the internet", but this is a different sense of "from"). The other models correctly identify "garnered" as an intransitive passive verb with "from" introducing an adverbial PP adjunct. The gold category of "from" is so complicated because it is correlated with "to": First it expects an NP object on the right ("1984"), then an adverbial adjunct on the right (the to-PP), after which it produces an adjunct to a VP.[4] AddrMLP's predictions for "garnered" and "from" are consistent in treating the entire construction "from 1984 to 1986" as an argument of the verb.

| | orders began | piling | up |
|---|---|---|---|
| **Gold** | | **(S[ng]\NP)/PR** | **PR** |
| MLP | | S[ng]\NP | ADV |
| K+19 | | S[ng]\NP | ADV |
| RNN | | (S[ng]\NP)/PP | S[adj]\NP |
| AddrMLP | | ✓ | ✓ |

Table 4: Here, the intended treatment of the particle (PR) "up" is as an argument selected by the predicate. Only AddrMLP gets this right. We assume this is preferable over treating it as a VP adjunct (as the non-constructive and K+19 taggers do) from a semantic perspective, because "pile up" is a fixed expression with a meaning distinct from that of "(to) pile" or "pile in". The RNN categories are both wrong and inconsistent (the "piling" category expects a PP and the "up" category is predicative).

[4] ADV is not an actual atomic category. We use it to abbreviate the VP-adjunct category (S\NP)\(S\NP). PP is a conventionalized atomic category for argument-PPs.