Supertagging the Long Tail
with Tree-Structured Decoding of Complex Categories

Jakob Prange Nathan Schneider Vivek Srikumar
Georgetown University
{jp1724, nathan.schneider}@georgetown.edu
University of Utah svivek@cs.utah.edu

Abstract

Although current CCG supertaggers achieve high accuracy on the standard WSJ test set, few systems make use of the categories’ internal structure that will drive the syntactic derivation during parsing. The tagset is traditionally truncated, discarding the many rare and complex category types in the long tail. However, supertags are themselves trees. Rather than give up on rare tags, we investigate constructive models that account for their internal structure, including novel methods for tree-structured prediction. Our best tagger is capable of recovering a sizeable fraction of the long-tail supertags and even generates CCG categories that have never been seen in training, while approximating the prior state of the art in overall tag accuracy with fewer parameters. We further investigate how well different approaches generalize to out-of-domain evaluation sets.

1 Introduction

Combinatory Categorial Grammar (CCG; Steedman, 2000) is a strongly-lexicalized grammar formalism in which rich syntactic categories at the lexical level impose tight constraints on the constituents that can be formed. Its syntax-semantics interface has been attractive for downstream tasks such as semantic parsing (Artzi et al., 2015) and machine translation (Nádeje et al., 2017).

Most CCG parsers operate as a pipeline whose first task is ‘supertagging’, i.e., sequence labeling with a large search space of complex ‘supertags’ (Clark and Curran, 2004; Xu et al., 2015; Vaswani et al., 2016, inter alia). The complex categories specify valency information: expected arguments to the right are signaled with forward slashes, and expected arguments to the left with backward slashes. For example, transitive verbs in English (like “saw” in Figure 1a) are tagged \((S/\text{NP})/\text{NP}\) to indicate that they expect a subsequent object noun phrase (\(\text{NP}\)) and a preceding subject \(\text{NP}\) to form a clause (\(S\)). Given the 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. In contrast to the simpler task of part-of-speech tagging, supertaggers are required to resolve most of the syntactic ambiguity in the input.

One key challenge of CCG supertagging is that the tagset is large and open-ended to account for combinatorial possibilities of syntactic constructions. This results in a heavy-tailed distribution of supertags, which is visualized in Figure 1b; a large proportion of unique supertags are rare or unseen (out-of-vocabulary, OOV) even in a training set as large as the Penn Treebank’s. Previous CCG supertaggers have surrendered in the face of this challenge: they treat categories as a fixed set of opaque labels, rather than modeling their compositional structure. Following Clark (2002), the standard approach is to consider only supertags appearing at least 10 times in the training data, sacrificing the possibility of predicting two thirds of the supertag types in CCGbank. Rare supertags may have little impact on overall token accuracy—but the cost of this compromise is a fundamental incapability in truly generalizing to the task.

In this paper, we confront the long-tail problem head-on by proposing a constructive framework in which supertags are built from scratch rather than predicted as opaque labels (Kogkalidis et al., 2019). In contrast to prior constructive supertaggers (Kogkalidis et al., 2019; Bhargava and Penn, 2020), our model builds upon the observation that supertags are themselves tree-structured, and hence can be generated top-down.\(^1\) Our experiments on the English CCGBank and its rebanked version show that constructing supertags as trees improves

---

\(^1\)Our models and code are available at https://github.com/jakpra/treeconstructive-supertagging.
Mary saw John and Bill

(a) A CCG-supertagged sentence. Colors indicate functors and atomic categories that will unify in parsing.

(b) Number of supertag types (circle sizes) in relation to token log-frequency (y-axis) and supertag depth (x-axis) for the Rebank training set (left) and test set (right). Colors and horizontal offsets indicate supertags’ training-data frequency band (decreasing frequency from left to right for each depth value).

Figure 1: CCG supertags.

Cat := FxnCat | AtomCat
FxnCat := Cat Slash Cat
AtomCat := N | NP | S | PP | ... 
Split := / \ / 

Figure 2: The ‘syntax’ of CCG categories, using infix notation for complex categories (FxnCat). Our model generates supertags of type Cat top-down from this grammar.

our ability to predict rare and even unseen tags, without sacrificing performance on the more common ones.

Our contributions are threefold:
1. We introduce a general constructive supertagger that generates each lexical category recursively as a tree. To our knowledge, this is the first tree-structured predictor of its kind.
2. We apply this model to English CCG supertagging. On frequent supertags, it matches the more traditional approach of using a fixed label set, while on the rare and unseen ones, we see substantial improvements in predictive performance.
3. We perform an array of in-depth analyses that highlight the impact of different modeling and inference choices for the task of predicting supertags.

2 Motivation

2.1 Anatomy of a Supertag

The internal structure of any CCG supertag is a tree licensed by the CFG in Figure 2. Atomic categories like S and NP are related by slashes to form functional categories, which can in turn participate in larger functional categories. By convention, the infix-notation supertag (S(NP)/NP) is equivalent to the tree in Figure 3a, with prefix notation (/ (\ S NP) NP), where the slash signals the direction in which the category can combine, the right child of any slash is the argument, and the left child is the result of combining the category with its argument. These hierarchical supertags constrain lexical item combination, e.g., specifying subcategorization of verbs for an object NP to the right (\). This flexibility leads to infinite\(^2\) possible supertags; in practice, they follow a power law distribution. CCGbank (comprising the WSJ portion of the Penn Treebank) contains numerous rare supertags, including several that occur only in the test set. Still others can be expected to occur in a much larger English corpus.

In previous work, CCG supertaggers have skirted this problem by ignoring the long tail of supertags: specifically, the ones occurring fewer than 10 times in the training set. The consequences of such a threshold can be seen from Figure 1b, which visualizes the distribution of supertag types in terms of depth (representing supertag complexity) and token frequency. The supertags seen in training that would be ignored under a threshold of 10 appear in red, and the test set supertags never seen in training in dark blue. Though these only account for 0.2% of tokens in the test set, they are present in nearly 4% of sentences and represent fully two thirds of supertag types in CCGbank. Further, we see that rarer categories are increasingly more complex, i.e., their argument and result types are in turn composed of FxnCat's. Note in particular that the bulk of depth-4 categories and almost all categories with depth 5 or more fall below the

\(^2\)But see §7 for a discussion of how linguistic patterns limit the set of observed tags.
10-count threshold.

Inspired by the recent proposals of Kogkalidis et al. (2019) and Bhargava and Penn (2020), we hypothesize that modeling the structure of supertags, rather than treating them holistically and thresholding by frequency, can successfully generalize to rare and unseen tags. For example, a good model should draw connections between words that are NPs themselves, words that take NPs as arguments (e.g., verbs), and words that yield NPs as their result (e.g., determiners). We examine whether such linguistically-informed generalizations can benefit supertags of various frequency and structures, focusing on the rare and complex ones.

2.2 Constructivity in Supertagging

We contrast two general paradigms for supertagging below. (Our experiments will explore multiple specific modeling strategies within each.)

Most previous supervised CCG supertaggers assume a closed tagset and nonconstructively assign one complete category per word (Figure 3b). This paradigm is oblivious to the internal structure of the supertag and incapable of predicting unseen supertags. This is often combined with a frequency cutoff: only the $k$ supertags seen at least $n$ times in the training data are considered by the model, making each tag decision a $k$-way classification task. Traditionally (Clark, 2002), systems use a threshold of $n = 10$ yielding $k = 425$ in CCGbank and $k = 511$ in CCGrebank). The main motivation for this is to sidestep the most sparse and possibly noisy region of the output space without dramatically decreasing token coverage. Below we experiment with both thresholded and non-thresholded models.

In contrast, a constructive tagger models the internal structure of supertags (Kogkalidis et al., 2019). Supertags are constructed from minimal pieces (which for CCG are slashes and atomic categories).\(^3\) There is no frequency cutoff at training time.\(^4\) At test time, supertags are predicted piece by piece, and there is no constraint that predicted supertags must have been seen before. This can be done sequentially or recursively, taking the categories’ internal tree structure into account.

Two different methods of sequential decoding

\[^3\text{For simplicity, we consider linguistic attributes like dcl (declarative) to be part of the atomic category.}\]

\[^4\text{In principle, a constructive model could be trained with frequency-thresholded training data, but we do not see any value in pursuing this option, as constructivity in itself already mitigates noise and sparsity.}\]
We therefore grow supertags for each word from all supertagging models we compare consist of well-formed categories. As CCG supertags are not which may be atomic categories or functors them-

(a) a sequence encoder, which generates a \(d\)th in practice, we impose an upper limit this measure has virtually no impact on tagging performance.

(b) an output-positional encoding. We experiment with two alternative ways of deriving hidden states for category-internal positions \((k,i)\), where \(i > 0\): a tree-structured recursive neural network (TreeRNN; Tai et al., 2015, inter alia), and a deterministic addressing function that accesses each node directly (AddrMLP). Both variants, described below, also take into account the current node’s ancestors.

Output-positional encoding. We experiment with two alternative ways of deriving hidden states for category-internal positions \((k, i)\), where \(i > 0\): a tree-structured recursive neural network (TreeRNN; Tai et al., 2015, inter alia), and a deterministic addressing function that accesses each node directly (AddrMLP). Both variants, described below, also take into account the current node’s ancestors.

The TreeRNN (eq. (3)) computes the hidden representation for a child node \(c(i)\) from a vector embedding of its parent’s label \(y_{k,i}\) and the hidden representation \(h_{k,i}\). The encodings are separately computed for child nodes representing the result \(c = ‘left’\) and argument \(c = ‘right’\) of the parent. Following K+19, we use the transpose of the last layer of the MLP to embed labels. Our experiments use gated recurrent units (GRUs; Cho et al., 2014).

\[
\begin{align*}
    h_{k,c(i)} &= \text{GRU}_c \left( \text{Embed} \left( y_{k,i} \right), h_{k,i} \right) \\
    h_{k,i} &= \text{GRU}_i \left( \text{Embed} \left( y_{k,i} \right), h_{k,i} \right)
\end{align*}
\] (3)

Using tree-structured RNNs for top-down generation is reminiscent of Zhang et al. (2016).

For the AddrMLP, we represent the position \(i\) of a node and the \(\text{S}1\)lash operators\(^7\) in its ancestors (denoted by \(Y_{k,\text{anc}(i)}\)) as a single feature vector that augments the contextualized word embedding:

\[
\begin{align*}
    h_{k,i} &= h_{k,0} + \text{Linear} \left( \text{Features} \left( i, Y_{k,\text{anc}(i)} \right) \right)
\end{align*}
\] (4)

We employ a binary addressing scheme to refer to individual nodes: each node in a category’s tree representation is addressed by a sequence of bits

\(5\) The limits on depth and arity are practical simplifications that follow from our task (supertags are always binary trees) and data distribution (there are no categories with depth > 6 in any of the training or development sets we use). However, our model can be generalized to trees of arbitrary depth, and not as easily, but conceivably, to a different or even variable arity. It turns out that none of the evaluation sets contain categories that are deeper that what is seen in training (except the redistributed test set in Figure 4, which contains one), so this measure has virtually no impact on tagging performance.

\(6\) We also experimented with a BiGRU encoder, but got consistently worse results.

\(7\) Only \(\text{S}1\)lash operators can have children (Figure 2).
We train the model using the AdamW optimizer (Loshchilov and Hutter, 2019) and apply teacher forcing (Williams and Zipser, 1989) to avoid a noisy feedback loop during learning.

**Attention.** While each word’s contextualized encoding contains some information about all other words in the sentence, we hope to increase the model’s output consistency using attention (Bahdanau et al., 2015; Kim et al., 2017; Wu et al., 2017) over the encoder’s hidden state. We compute attention weights $\alpha$ as in eq. (5) and then add the $\alpha$-weighted context values to the hidden state, eq. (6), replacing the simpler MLP from eq. (2).

\[
\alpha = \text{SoftMax} \left( \mathbf{h}_o, \mathbf{H}_i \right) \quad \text{(5)}
\]
\[
o_{t,i} = \text{MLP} \left( \mathbf{h}_{t,i} + \alpha \mathbf{H}_0 \right) \quad \text{(6)}
\]

### 3.3 Learning

We train the model using the AdamW optimizer (Loshchilov and Hutter, 2019) and apply teacher forcing (Williams and Zipser, 1989) to avoid a noisy feedback loop during learning.

**Loss function.** To achieve our goal of constructing correct and complete categories, we need our models to be correct in each atomic decision, even and especially for more complex categories. We make the loss function sensitive to this by normalizing the cross-entropy between the predictions and the ground-truth only over the number of words in a batch and retaining the unnormalized sum over individual atomic category decisions. This naturally scales with category complexity.

If instead we were normalizing over atomic decisions, too, the loss contribution of, e.g., $\text{NP}$ when it occurs inside a complex category $(S, \text{NP})/\text{NP}$ with size 5, would be 5 times smaller than when it occurs as a complete category on its own. The disadvantage that complex categories already have as they tend to be rarer than simpler ones (Figure 1b) would be reinforced. By keeping the atomic losses unnormalized, we therefore essentially put higher weight on the long tail in order to counterbalance this trend and improve generalizability.

### 4 Experimental Setup

Per our quest to supertag the long tail, we compare our TreeRNN and AddrMLP models to the following baselines:

1) **Thresholded classification** (MLP_10): We compute the output probabilities directly from the encoder’s hidden state. (Since there is always exactly one output position for each input word, no additional encoding function is needed.) Only categories that are seen 10 times or more in training are considered. Supertags that fall below the threshold are replaced with an $<$UNKNOWN$>$ symbol in training.

2) **Non-thresholded classification** (MLP_1): Like MLP_10, except that all tags seen in training may be predicted no matter their frequency.

3) **Per-sentence sequential** (K+19): Kogkalidis et al. (2019) construct type-logical supertags by generating for each sentence a single sequence of atomic types and functors (Figure 3c). Trees are unwrapped in prefix notation and complete tags are separated from one another by a special token. We adapt K+19’s implementation of the sequence-to-sequence Transformer model (Vaswani et al., 2017), accommodating its decoding procedure and memory requirements by training with a batch size of 32 for up to 256 epochs. We achieve the best performance using a cosine-annealed learning rate schedule that is warmed up over 10% of the total training steps and with a warm restart after 128 epochs (Loshchilov and Hutter, 2017).

4) **Per-tag sequential** (RNN): Instead of generating a single sequence for each sentence, Bhargava and Penn (2020) generate each word’s supertag separately with an RNN. We implement a simplified version of Bhargava and Penn’s model, omitting their prediction history connections between su-
We use two versions of the English CCGbank as
pertags, and using GRUs for decoding. We train
this model for up to 50 epochs (batch sizes and
learning rates are as with the tree-structured and
nonconstructive models).

If not indicated otherwise, we train the models
with a batch size of 8 for a maximum of 10 epochs,
and use early stopping based on the best develop-
ment set performance. All reported results are
averaged over 3 random restarts.

For downstream parsing evaluation (§6.3), we
run the C&C parser (Clark and Curran, 2007; Clark
et al., 2015) with the pretrained CCGbank model
and default hyperparameters, providing as input our
supertaggers’ 1-best predictions and POS tags au-
tomatically obtained using Stanza (Qi et al., 2020).

### 4.1 Model Details and Hyperparameters

In Table 1 we report the model and training hyper-
parameters we use to facilitate replication of our
results. We performed manual grid-search based
on the development data to find workable learning
rates. We chose a hidden dimensionality of
768 to match RoBERTa’s. We kept the default val-
ues for the AdamW hyperparameters. We follow
Kogkalidis et al. (2019) in setting up the sequen-
tial Transformer model with 8 decoder heads and
2 decoder layers, but swap out the from-scratch
coder with RoBERTa-base.

### 4.2 Datasets

We use two versions of the English CCGbank as
in-domain (financial news) training and test sets:
the original (Hockenmaier and Steedman, 2007)
and Honnibal et al.’s (2010) ‘rebanked’, i.e., cor-
rected and enriched version (training sets reported
in Table 2; the results tables show test set counts).

The original CCGbank and Rebank differ in a
number of conventions for atomic categories and
category construction (Honnibal et al., 2010). Re-
bank has a larger and more diverse category space,

| Hidden dim $d$ | 768 | Weight decay $\cdot .01$
|----------------|-----|------------------------|
| Activation     | gelu| Dropout $\cdot .2$
|                |     | AdamW $\beta$’s $\cdot .9, .999$
|                |     | AdamW $\epsilon$ $\cdot 1e-6$
|                |     | Max cat depth $\cdot 6$
|                |     | LRs $\cdot 1e-4, 1e-5 (ft)$
|                |     | Seeds $\cdot 14112, 36125, 92225$

Table 1: Hyperparameters used in our experiments.

<table>
<thead>
<tr>
<th>CCGbank</th>
<th>Rebank</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat types</td>
<td>1,285</td>
</tr>
<tr>
<td>$\geq 100$</td>
<td>172</td>
</tr>
<tr>
<td>10–99 (medium rare)</td>
<td>253</td>
</tr>
<tr>
<td>$&lt; 10$ (very rare)</td>
<td>860</td>
</tr>
<tr>
<td>atomic</td>
<td>34</td>
</tr>
<tr>
<td>sentences</td>
<td>39,604</td>
</tr>
<tr>
<td>tokens</td>
<td>929,552</td>
</tr>
<tr>
<td>medium rare cat</td>
<td>7,549</td>
</tr>
<tr>
<td>very rare cat</td>
<td>2,055</td>
</tr>
</tbody>
</table>

Table 2: Statistics of the CCG training corpora we
use in our experiments.

Figure 4: Shifting the tail to evaluation. The new
test set (right) consists of those sentences in sec-
tions 02–21 that contain a category type occurring
less than 10 times, and the new training set of the
remaining sentences (left). As a result, we evaluate
on many more category types that are not seen at all
in training (dark blue circles/right-most horizontal
offset for each depth) than before (Figure 1b).

due in large part to a more principled treatment
of NP argument structure. Hence, we conduct our
main experiments with Rebank and use the original
CCGbank for comparisons with prior work.

A limitation of standard test sets for studying the
long tail is that category types appearing rarely in
training are even less frequent in evaluation (the
Rebank test set contains just 107 tokens of cate-
gories seen 1–9 times in training, and only 27 to-
kens of OOV categories). Scores computed over
these small samples may thus not reliably estimate
the models’ generalization capacity. We counteract
this in two ways: 1. by explicitly redistributing the
training and test splits; and 2. by evaluating on out-
of-domain data, with the assumption that a shift in
domains means a shift in category distribution.

In the first case, we train the models on sentences
containing exclusively the higher-frequency (≥10)
categories, and evaluate them only on sentences with
at least one rare category. We split the usual
Rebank training set (WSJ sections 02–21) in this
while PMB is more similar to Rebank; we evaluate
way—the distribution follows Figure 4.

Table 3: Main results on Rebank evaluation set (WSJ section 23). Accuracy scores are computed for bins based on the order of magnitude of category occurrences in training, and complexity of categories in depth, with depth=0 corresponding to atomic categories like NP (Figure 3a has depth 2). Token (n) and type (N) counts for each bin are given in the first two rows. ‘@’ refers to model variants that use an attention mechanism over the encoder’s hidden states. (As a Transformer model, the K+19 model attends to both the encoder and previously predicted outputs by default.) In each column, we highlight all results r that fall within the standard deviation of the best result b, i.e., when r + stdv(r) > b − stdv(b). For comparison, the overall tagging accuracy reported in Honnibal et al. (2010) is 92.2%.

5 Results

We report our main results on Rebank in Table 3. In terms of overall accuracy, the tree-structured
constructive supertaggers (best: 94.70%) outperform
the sequential ones (90.68%, 93.92%) and are roughly on par with the nonconstructive classifiers (best: 94.83). Performance is generally very similar across all systems, except K+19. We conjecture that the main disparities between K+19 and the other models lie in the increased ‘cognitive load’ of having to learn the correct structure of categories, as well as the missing hard alignment between words and supertags at test time.

Regarding the long tail, we ask: Can constructive models accurately predict rare and complex categories without sacrificing performance on the head of the distribution? To answer this question, we break down performance by the frequency of category types in the training data. The baseline is the thresholded classifier MLP_10, which performs well on frequent categories but cannot access rare categories occurring less than 10 times in training. The simplest way of resolving this main hurdle is

\[2 \text{The few supertags in the 1–9 range of the new training set are those which occurred slightly above 9 times in the original training set, but some of their tokens were moved due to occurring in the same sentence as a low-frequency tag.} \]
Table 4: Results on the original CCGbank evaluation set (WSJ section 23). The population \(n\) for computing Parseability is the number of sentences in the test set. In each column, we highlight all results that fall within the standard deviation of the best result.

Table 5: Relevant baselines reported in previous work (on the original CCGbank): Vaswani et al. (2016), Clark et al. (2018), Tian et al. (2020), and Bhargava and Penn (2020).

Table 6: Performance of the best systems (the variants with attention for each paradigm) on redistributed Rebank train/test splits. Frequency bins are based on the new training set.

For comparison with existing work (Table 5), we also report results on the original CCGBank (Table 4). Our best constructive and nonconstructive models are on par with the previously reported state of the art in terms of overall accuracy. Tian et al. (2020) only report performance on categories seen at least 10 times in training, i.e., the union of our ‘\(\geq 100\)’ and ‘10-99’ bins; our top-3 results on this subset are MLP_1: 96.37\%, MLP_10@: 96.27\%, AddrMLP@: 96.22\%. The rise in absolute scores to remove the threshold, and indeed we find that MLP_1 is able to predict about a quarter of long-tail categories correctly. Can we do better? The sequence-to-sequence model by K+19 does a lot better on the tail and even retrieves some unseen categories, but at the cost of frequent ones. The per-tag 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, outperforms all others on the rare tail while matching nonconstructive taggers on frequent and simple ones.

from Table 3 to Table 4 is consistent with Honnibal et al.’s (2010) finding that Rebank is more difficult to supertag and parse than CCGbank due to its sparser category space. We therefore encourage future researchers to conduct experiments on Rebank and report detailed results for frequency- and complexity-binned subsets of the output space to facilitate more in-depth comparisons.

Evaluating generalizability. One of the inherent problems of the supertagging task is the sparsity of the output space. This is, however, not sufficiently captured by standard evaluation sets, as illustrated in Figure 1b. To test how well the models really generalize to the long tail, we evaluate them on alternatively sampled training and evaluation splits of the WSJ data (Table 6) as well as in domains diverging from the WSJ training
In both in-domain and out-of-domain data, the performance gap between the nonconstructive MLPs and AddrMLP on the most frequent categories is minimal and in fact lies within the standard deviation. Given the trend we observe from Tables 3 and 4 to Tables 6 and 7, the ability to generalize and positive numbers (red) are more typical for AddrMLP@ and negative numbers (blue) are more typical for one of the other systems.

6 Detailed Analysis

6.1 Constructing Complex Categories

While nonconstructive taggers do not distinguish between categories of varying complexity (each supertag prediction is a single k-way decision), 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. This raises the question: How difficult are categories of varying complexity for each of the systems?

As Figure 1b shows, deeper, i.e., more complex categories tend to be rarer and thus are more difficult than simple ones in general, for all models. Surprisingly however, we can see in the three right-
most columns of Table 3 that it is not dramatically more difficult for constructive systems to generate complex categories of depth $\geq 1$ than it is for non-constructive systems to simply assign them (apart from K+19, which underperforms on frequent categories regardless of their complexity).

In Figure 5 we take a closer look at the models’ ability to predict categories of the appropriate depth. For the sake of brevity, we only consider three extreme cases: MLP_10, K+19, and AddrMLP. Compared with MLP_10, which tends to choose one of the very frequent but relatively shallow categories of depth 1 or 2, AddrMLP prefers both standalone atomic (depth-0) categories and those of depth 3 and 4 (column totals in the top left matrix). On the head, AddrMLP confuses depth-1 for depth-0 categories and overpredicts the depth of depth-2 and depth-3 categories more frequently than the baseline. On the subset of rare categories (which are deeper than more frequent categories on average), AddrMLP is consistently better at predicting categories of the correct depth (diagonal in the bottom left matrix); the thresholded model consistently chooses categories that are too shallow here. The sequential tagger by K+19 struggles with predicting the correct depth for frequent categories much more than the tree-structured model (top right matrix), which is almost certainly a result of its lack of an inductive bias for the tree structure of categories. On the rare tail, however, its ability to guess the right depth is almost as good as that of AddrMLP (bottom right).

6.2 Generation Behavior and Unseen Tags

Are there any distinct patterns in the output of the different models? 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 8 and 9.

The thresholded tagger has the option to output an <UNKNOWN> label when it believes the correct category is not in the tagset. It makes use of this option for 0.25% of tokens on average (0.11% with standard train/test splits); when it does, the correct category is indeed missing from the tagset about

```
<table>
<thead>
<tr>
<th></th>
<th>Garnered</th>
<th>Piling</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gold</strong></td>
<td>$(S[pss]\NP)/(ADV/ADV)/NP</td>
<td>$(S[adj]\NP)/PP $(PP/ADV)/NP</td>
</tr>
<tr>
<td>MLP_10</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MLP_1</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>K+19</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RNN</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AddrMLP</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
```

Table 8: 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. AddrMLP’s predictions for “garnered” and “from” are consistent in treating the entire construction “from 1984 to 1986” as an argument of the verb.

```
<table>
<thead>
<tr>
<th></th>
<th>Orders began</th>
<th>Piling up</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gold</strong></td>
<td>$(S[ng]\NP)/PR PR</td>
<td>$(S[adj]\NP)/(PP/ADV)/NP</td>
</tr>
<tr>
<td>MLP_10</td>
<td>(S[ng]\NP)/PR PR</td>
<td>✓</td>
</tr>
<tr>
<td>MLP_1</td>
<td>(S[ng]\NP)/PR PR</td>
<td>✓</td>
</tr>
<tr>
<td>K+19</td>
<td>(S[ng]\NP)/PR PR</td>
<td>✓</td>
</tr>
<tr>
<td>RNN</td>
<td>(S[ng]\NP)/(S[adj]\NP)/PP $[adj]\NP</td>
<td>✓</td>
</tr>
<tr>
<td>AddrMLP</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
```

Table 9: 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 nonconstructive 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).

```
<table>
<thead>
<tr>
<th></th>
<th>Why</th>
<th>Constructive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gold</strong></td>
<td>$(S[ng]\NP)/(S[adj]\NP) $[adj]\NP</td>
<td>✓</td>
</tr>
<tr>
<td>MLP_10</td>
<td>$&lt;UNKNOWN&gt;$</td>
<td>✓</td>
</tr>
<tr>
<td>MLP_1</td>
<td>$(S/S)/(S[adj]\NP) $[adj]\NP</td>
<td>✓</td>
</tr>
<tr>
<td>K+19</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>RNN</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AddrMLP</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
```

Table 10: Supertags for Wh-words tend to be rare or unseen in training. Here, MLP_10 correctly identifies that it cannot predict the true category for “why” and instead outputs <UNKNOWN>, while MLP_1 chooses an incorrect tag. The constructive taggers are able to generate the correct category.
Table 11: 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 still obeying the grammar in Figure 2), or an invalid structure (✗ formed, e.g., missing arguments to slashes).

2/3 of the time. This happens, e.g., with WH-words in elliptical questions, as in Table 10.

In Table 11 we quantify the structural and labeling errors more generally, based on the redistributed evaluation set to ensure reliable estimates on rare phenomena. A substantial portion of erroneous categories actually do have the correct structure (✓ struct). For these cases, we perform a detailed error analysis, whose results we present in Figure 6. 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.

To what extent can the constructive models generate categories that were unseen during training? We take a closer look at categories the constructive taggers invented in the bottom halves of Table 11 and Figure 6. K+19 is the most willing to invent categories, closely followed by the tree-structured models and finally RNN, which is rather conservative in this respect (see sums of the last four rows in Table 11). Merely generating more new categories irrespective of their correctness is of course not necessarily an advantage, but it is encouraging to see the models make use of their freedom to do so at an adequate rate, rather than only reproducing known categories or vastly overgenerating invented ones. Interestingly, given that an incorrect invented category has the same structure as the gold category, we again see that the majority of errors are due to only a single attribute or slash, suggesting that in these cases the models get the general idea of the category right and only err in fine-grained and context-sensitive subcategorization. In the case of a slash mistake, they are notably also able to recover from it in later predictions.

While the tree-structured taggers are guaranteed to produce valid categories, it is possible for the sequential taggers to generate structurally invalid categories, i.e., sequences of atomic categories and slashes that are not licensed by the grammar in the models make use of their freedom to do so at an adequate rate, rather than only reproducing known categories or vastly overgenerating invented ones. Interestingly, given that an incorrect invented category has the same structure as the gold category, we again see that the majority of errors are due to only a single attribute or slash, suggesting that in these cases the models get the general idea of the category right and only err in fine-grained and context-sensitive subcategorization. In the case of a slash mistake, they are notably also able to recover from it in later predictions.

Table 12: A malformed supertag extracted from a sequence predicted by K+19. Underscores ‘_’ indicate gaps in the tree structure resulting from predicted surplus slashes.

<table>
<thead>
<tr>
<th>Predicted sequence</th>
<th>Predicted supertag</th>
<th>Gold sequence</th>
<th>Gold supertag</th>
</tr>
</thead>
<tbody>
<tr>
<td>/ / \ S[b] NP \ NP</td>
<td>((S[b]\NP)/(NP_)) / _</td>
<td>/ \ S[b] NP NP</td>
<td>(S[b]\NP)/NP</td>
</tr>
</tbody>
</table>

Figure 6: Fine-grained analysis of correctly-structured but incorrectly labeled predictions (✓ struct’ in Table 11). ‘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 \.
Table 13: Performance by part-of-speech, based on the original CCGbank test set. \( n \) and \( N \) refer to token and type counts in the test set, as before; \( f \) refers to the average frequency with which a supertag belonging to the respective POS class is seen in training.

Figure 2. With the tag-wise RNN generator, which generally refrains from inventing new categories, this only happens extremely rarely, but in the case of K+19, every 14th sentence is affected by an ill-formed supertag on average (every 66th sentence in the standard Rebank test set). A common source of errors is that too many slashes are predicted, whose argument and result slots can then not be filled by the predicted atomic categories. We show an example in Table 12.

And vice versa, are there any categories that are not generated despite being seen in training? There are 80 category types in the standard Rebank test set that none of the tree-structured taggers ever predict correctly, although they are attested in the training data, and there are 93 types that are never retrieved by K+19, 73 of which overlap. Out of these 73, no one occurs more than three times in the test set and almost all appear fewer than 50 times in training, with three exceptions: (\( NP|NP \)) \( \langle NP|NP \) \( (68 \) times in training), (\( (N|N) \) \( (N|N) \) \langle NP \) (50 times), and (\( NP \) \( NP \) \( N \) (50 times). The first one is usually used for the last part of complex numerical expressions (such as dates and ranges), but the one token bearing this category in the test set is “not” in “they might not miss one at all”, which is likely an annotation error.\(^{17}\)

The second one encodes prepositions modifying an appositive bare noun, typically an appellation or postposed proper noun. The third one is for determiners of appositions or parentheticals. 67 of the 73 types that are problematic for the constructive models are never accurately predicted by the nonconstructive models either.

\[^{17}\]There are a few more instances of such implausible lexical categories in the training data, like S or (\( \langle NP|PP \rangle \) \( /NP \).

6.3 Parts of Speech and Sentence Parsing

Parsing performance is computed using labeled F1-score (LF) over CCG dependencies in all sentences, following Clark and Curran (2007), and Parseability, i.e., the proportion of sentences for which a complete CCG derivation can be constructed.\(^{18}\) Nearly all the models we compare outperform the state of the art in labeled dependency F1-score (right-most columns in Table 4). Interestingly, the K+19 model produces more parseable supertag sequences than others, despite consistently lagging behind in terms of category accuracy. Apparently this tagger prefers to be self-consistent over producing the actual correct categories, either due to its multihead attention mechanism, the fact that decisions towards the end of the sequence have access to all previously predicted categories in their entirety (rather than just parts of them), or both.

Long-range dependencies. We examine supertagging performance by POS class (a few are shown in Table 13) and find that constructive and nonconstructive taggers perform similarly across classes, with one notable exception: WH-words, whose supertags are rarely seen in training and have a high type/token ratio at test time. Their special syntactic status raises the question: How important are constructivity, tree structure, and long-tail recall for recovering categories involved in long-range dependencies?

Somewhat surprisingly, we find that the RNN is best for these dependencies (Figure 7), which might be related to the two parsing metrics in Table 4:

- There are a few more instances of such implausible lexical categories in the training data, like S or (\( \langle NP|PP \rangle \) \( /NP \).
RNN@ strikes a good balance between LF and Parseability. We further examine the average dependency length per category, and contrary to our expectation, dependencies involving WH-categories are relatively short (usually 3–4 intervening words). We find that the supertags with the longest dependencies on average largely are functioning as subordinators, sentence adverbials, and inverted speech verbs such as $S[dcl] \neg S[dcl] / NP$. These supertags have in common that they all contain sentential result/argument pairs of the form $S[x] \neg S[x]$ (where $x$ is an optional attribute). The autoregressive nature of the RNN may be conducive to modeling the matching atomic categories of argument and result. Exploring various decoding orders for both sequential and tree-structured constructive taggers in order to more explicitly take advantage of these intra-category relations is an interesting avenue for future work. We also expect a major boost in Parseability from incorporating inter-category prediction history into our models (Bhargava and Penn, 2020). But this is nontrivial for tree-structured decoding and goes beyond our scope here.

### 6.4 Runtime and Model Size

While the constructive taggers need to make more individual decisions for each supertag than nonconstructive ones, they only have to consider a much smaller and denser output space. This trade-off between time and space complexity should be considered in addition to tagging accuracy when evaluating each model. Thus we ask: How do the constructive supertaggers compare to nonconstructive ones in terms of efficiency? In Table 14 we report model sizes (i.e., the number of learned parameters), training time until development performance plateaus, and inference speed. As model size and runtime vary greatly between different constructive taggers, the answer to our question depends on how supertags are modeled and inferred.

The K+19 sequential Transformer model has low efficiency for two reasons: The Transformer architecture itself has a large number of parameters; and sequential inference is slow because individual predictions for the same sentence cannot be parallelized and the number of inference steps per input sentence is linear in the sum of all category sizes (the number of atomic pieces) for that sentence. The GRUs in the RNN and TreeRNN models are much smaller than the Transformer of K+19, but the TreeRNN with its two GRUs for argument and result transitions ends up having almost as many parameters as the Transformer in total. The nonconstructive models map hidden representations into a much larger and sparser output space than the constructive models (and the output space of MLP_1, in turn, is larger and sparser than that of MLP_10). AddrMLP, on the other hand, consists exclusively of feed-forward layers, resulting in the smallest model size among the ones we compare.

The sequential models require relatively many training epochs to converge. The reason total training time is still comparable between K+19 and RNN despite the extreme disparity in inference speed is that the Transformer is trained non-autoregressively and thus performs inference only between epochs, for evaluation on the development set, whereas RNN training inherently relies on inference. The nonconstructive and tree-structured models converge within the first 10 epochs.

For the per-tag constructive models RNN, TreeRNN, and AddrMLP we parallelize inference across all supertags in a batch, and for the tree-structured ones, we further parallelize the prediction of the children of slash functors, making their inference time logarithmic in the size of the largest predicted category in a sentence.

AddrMLP is both time- and space-efficient overall. Its parameter count is only $\approx 1/10$ of the K+19 model and $\approx 1/2$ of the nonconstructive ones.

### 7 Discussion & Related Work

For a long time, researchers have addressed the large search space of CCG supertags. Baldridge (2008) and Ravi et al. (2010) were particularly con-

<table>
<thead>
<tr>
<th>Model</th>
<th>Params</th>
<th>Train time</th>
<th>Infer speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonconstructive Classification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MLP_10</td>
<td>2.0</td>
<td>9</td>
<td>191</td>
</tr>
<tr>
<td>MLP_1</td>
<td>2.4</td>
<td>11</td>
<td>195</td>
</tr>
<tr>
<td>Constructive: Sequential</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K+19</td>
<td>11.8</td>
<td>120</td>
<td>0.3</td>
</tr>
<tr>
<td>RNN</td>
<td>4.8</td>
<td>68</td>
<td>135</td>
</tr>
<tr>
<td>Constructive: Tree-structured</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TreeRNN</td>
<td>8.3</td>
<td>10</td>
<td>125</td>
</tr>
<tr>
<td>AddrMLP</td>
<td>1.3</td>
<td>10</td>
<td>126</td>
</tr>
</tbody>
</table>

Table 14: Model size and time required for training and inference. All models use the RoBERTa encoder, whose 124.6 million parameters are not included here. Training times are approximate and include development set evaluation at every epoch.
cerned with high lexical ambiguity and counteracted this, respectively, by improving lexicon initialization using linguistic principles, and explicitly minimizing model sizes. Deoskar et al. (2013), working with lexico-syntactic dependencies similar to supertags, addressed difficulties arising from the long tail of rare and unseen words; and Deoskar et al. (2014) addressed a similar issue specifically for generalizing a CCG parser. The problem of out-of-vocabulary words has gotten much less severe with the advent of deep contextualized sentence encoders operating on subword units.

An alternative way of reducing the burden on the supertagger is to couple it with the parser and jointly optimize lexical and phrasal categories, subject to the combinatory rules of CCG (Auli and Lopez, 2011; Garrette et al., 2015). Garrette et al. (2015) notably included a fully constructive probabilistic model of categories in a weakly-supervised grammar-induction scenario. In the context of grammar induction for semantic parsing specifically, Kwiatkowski et al. (2011) and Artzi et al. (2015) have explored template-based methods to generalize a limited initial lexicon to likely alternative syntactic usages of observed words.

In the special case that all categories in a sentence but one are known, the combinatory rules of CCG can be reverse-engineered to infer the missing category. As an efficient and scalable example of this, Thomforde and Steedman (2011) have proposed Chart Inference.

Since the beginning of the neural era, virtually all advances in CCG supertagging have involved different means of deep sequence encoding, typically in the form of (Bi)LSTMs, with techniques including: predicting categories directly from the word-level encoder (Xu et al., 2015; Lewis et al., 2016); giving credit to likely category sequences (Vaswani et al., 2016; Kadari et al., 2018); forcing the model to distribute its attention over a fixed-size window of neighboring words (Wu et al., 2017); training the encoder specifically to be aware of each word’s neighboring categories (‘cross-view training’; Clark et al., 2018); and latently modeling parse chunks with a graph-convolutional network over word n-grams (Tian et al., 2020).

Similar techniques have been applied to supertagging in the related formalism Tree-Adjoining Grammar (TAG) (Kasai et al., 2017, 2018). Zhu and Sarkar (2019) have formulated TAG supertagging as multitask learning with respect to certain aspects of the elementary trees’ internal structure. Their system predicts the category that optimizes the weighted sum of the scores for each subtask.

A possible objection to generating categories entirely productively is that universal linguistic patterns constrain the shape of categories and the syntactic relations they may engage in (Chomsky and Lasnik, 1993; Baldridge and Kruijff, 2003), and for any given language, word order and other language-specific properties further restrict the underlying grammar. Note that for a $\text{FxnCat}$ shape with given argument and result types, the direction of its slash functors is largely determined by global word order properties of the respective language. Consider the prototypical category shape for adpositions, $(\text{NP|NP})|\text{NP}$, where ‘|’ stands for either forward or backward direction. In English, a predominantly prepositional (as opposed to postpositional) language with postnominal-PP modifiers, this shape is most commonly instantiated as $(\text{NP|NP})/\text{NP}$, but different ordering patterns may dominate in other languages. Languages with more flexible word orders will show a greater variance in slash directionality than those with fixed word orders. While our approach is in principle equipped to pick up on such patterns from data, we do not explicitly prohibit unlikely category types. One potential way of incorporating such information is via logical constraints at training and/or inference time in the style of Li and Srikumar (2019); Li et al. (2019). Another approach could be a hybrid one, bridging between constructive and nonconstructive tagging in a more fluid way. We plan to explore these avenues in future work.

8 Conclusion

We introduced a novel, explicitly tree-structured CCG supertagging method, advancing the nascent paradigm of constructive supertagging. Our analysis of complex and long-tail categories highlights the positive impact of different modeling and inference choices within this paradigm: structural inductive bias as well as adequate contextualization via, e.g., attention contribute to more efficient, robust, and self-consistent models. We hope that our proposed method can be instrumental in researching and applying not only CCG and related syntactic formalisms, but also other paradigms like morphological (de)composition of complex words in morphologically rich languages, or compositional semantic parsing.
Acknowledgements

We would like to thank Aditya Bhargava and Konstantinos Kogkalidis for assistance with replicating their experiments and extended discussions of constructive models; Mark Steedman, Julia Hockemaier, and Noah Smith for their deep insight into CCG; Kilian Evang and Lasha Abzianidze for explanations of data formats and conventions; Tao Li, Yichu Zhou, and Sean MacAvaney for help with implementing our models in PyTorch; and members of the NERT lab at Georgetown for feedback on an early abstract. We are indebted to our TACL action editor Reut Tsarfaty and editor-in-chief Ani Nenkova, as well as the anonymous reviewers for their diligent assessment and handling of logistics.

This research was supported in part by NSF award IIS-1812778 and a generous gift from Google.

References


