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

Jakob Prange **Nathan Schneider** **Vivek Srikumar**

Georgetown University University of Utah
{jp1724, nathan.schneider}@georgetown.edu 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 (Nădejde 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\NP)/NP to indicate that they expect a subsequent object noun phrase (NP) and a preceding subject 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.

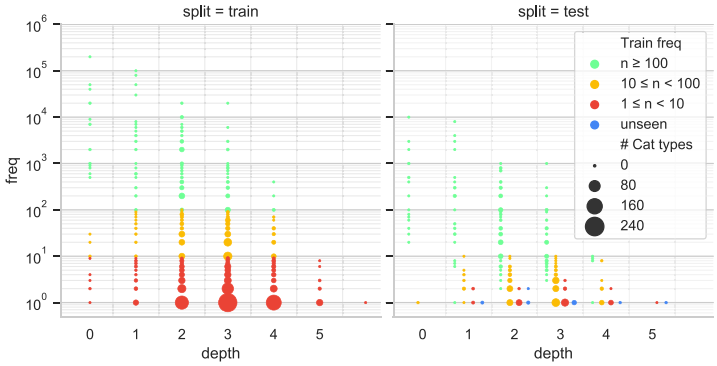
In this paper, we confront the long-tail problem head-on by proposing a *constructive* framework in which supertags are built from scratch rather than predicted as opaque labels (Kogkalidis et al., 2019). In contrast to prior constructive supertaggers (Kogkalidis et al., 2019; Bhargava and Penn, 2020), our model builds upon the observation that supertags are themselves tree-structured, and

Transactions of the Association for Computational Linguistics, vol. 9, pp. 243–260, 2021. https://doi.org/10.1162/tacl_a_00364
Action Editor: Reut Tsarfaty. Submission batch: 7/2020; Revision batch: 10/2020; Published 3/2021.
© 2021 Association for Computational Linguistics. Distributed under a CC-BY 4.0 license.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

Mary	saw	John	and	Bill
NP	(S\NP)/NP	NP	(NP\NP)/NP	NP

(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.

hence can be generated top–down.[1] Our experiments on the English CCGbank and its rebanked version show that constructing supertags as trees improves 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

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

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.

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

[1] Our models and code are available at https://github.com/jakpra/treeconstructive-supertagging.

[2] But see §7 for a discussion of how linguistic patterns limit the set of *observed* tags.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

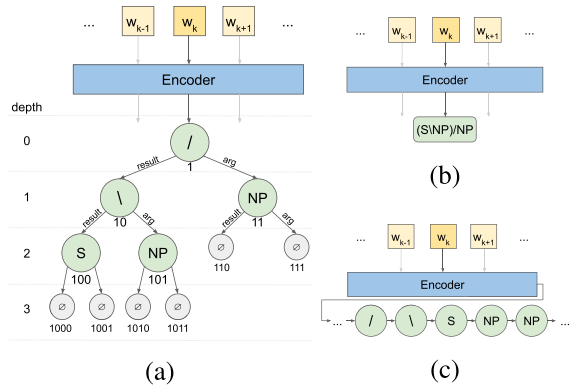


Figure 3: Schematic of our tree-structured supertagger (left) in contrast with unstructured (top right) and sequential (bottom right) models. Supertag depth also corresponds to decoding steps. Numbers below nodes denote positions or addresses.

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 FxnCats. Note in particular that the bulk of depth-4 categories and almost all categories with depth 5 or more fall below the 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 have been explored by Kogkalidis et al. (2019) (hereafter 'K+19') and Bhargava and Penn (2020) ('BP20'). K+19 used a sequence-to-sequence model, with a single target sequence consisting of all serialized supertags for a sentence (Figure 3c). They experimented with a type-logical grammar formalism similar to CCG, and a Dutch corpus. BP20 decoded CCG supertags as a separate sequence per token, and additionally conditioned each new supertag on the prediction history.

Here we go a step further and introduce methods for directly decoding supertags as *trees*, freeing the models from having to learn this fundamental property from sequential data. We hypothesize that this will produce better and more compact representations that generalize to the long tail.

3 Tree-Structured Constructive Supertagging

Given a sequence of words (a sentence), our goal is to predict each word's supertag. Constructing a

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

[4]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.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

supertag from its components requires a scoring function for the parts that is cognizant of both surrounding words and categories. Below we describe the decoding procedure (§3.1) and scoring functions (§3.2) we developed for this purpose, which, in line with §2, explicitly incorporate the categories' tree structure.

3.1 Predicting Tree-structured Supertags

According to the grammar in Figure 2, each category is a binary tree with the following properties: (1) Slashes are non-terminals with two children: the category's argument (the syntactic type it seeks to combine with), and its result (the type it yields after combining with its argument). (2) AtomCats are leaf nodes. (3) The root of the tree is either the category's sole AtomCat, or its outermost functor, whose argument it seeks to combine with first.

Our output supertags are trees, but there is a crucial difference between our work and constituency parsing of sentences. In the latter case, the yield of a predicted tree is constrained to be the input sentence, thereby restricting both its depth and width. But in the case of supertagging, each word is associated with a binary tree–structured supertag whose breadth and depth are unknown at inference time. We therefore grow supertags for each word from the top down (Figure 3a). 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 (Figure 3a). 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 CCG supertags are not very deep in practice, we impose an upper limit on the depth of predicted trees based on the most complex categories found in the training and development data, with the main advantage that memory allocation during training can be bounded.[5]

[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.

3.2 Modeling Supertags

All supertagging models we compare consist of (a) a sequence encoder, which generates a d-dimensional contextualized representation $\mathbf{h}_{k,0}$ for each word k in a sentence $\mathbf{x}$ (equation (1), together forming the $|\mathbf{x}| \times d$ matrix $\mathbf{H}_0$); (b) an output-positional encoder, which generates the hidden representation $\mathbf{h}_{k,i}$ for a position indexed by i within the k^{th} word's category tree; and (c) a fully-connected 2-layer perceptron (MLP) with a final softmax layer which maps such a representation to a probability distribution $\mathbf{o}_{k,i}$ over the inventory of possible labels L (atomic categories and slashes; equation (2)). We use the term *position* and the index i 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 Figure 3a and 3c, or the single output in Figure 3b.

$$\mathbf{H}_0 = \text{Encoder}\,(\mathbf{x}) \tag{1}$$

$$\mathbf{o}_{k,i} = \text{MLP}\,(\mathbf{h}_{k,i}) \tag{2}$$

The label $y_{k,i}$ is the most probable one per the MLP's prediction.

Contextualized Word Embeddings. In all conditions, we encode sentences using the pretrained RoBERTa-base encoder (Liu et al., 2019), fine-tuning it for our task.[6] Several recent studies have shown that such models can capture syntactic properties and relations (e.g., Jawahar et al., 2019; Clark et al., 2019; Hewitt and Manning, 2019).

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** (equation (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 $\mathbf{h}_{k,i}$. 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.

[6]We also experimented with a BiGRU encoder, but obtained consistently worse results.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

Our experiments use gated recurrent units (GRUs; Cho et al., 2014).

$$\mathbf{h}_{k,c(i)} = \mathrm{GRU}_c\left(\mathrm{Embed}\left(y_{k,i}\right), \mathbf{h}_{k,i}\right) \quad (3)$$

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 Slashes[7] in its ancestors (denoted by $\mathbf{Y}_{k,\mathrm{anc}(i)}$) as a single feature vector that augments the contextualized word embedding:

$$\mathbf{h}_{k,i} = \mathbf{h}_{k,0} + \mathrm{Linear}\left(\mathrm{Features}\left(i, \mathbf{Y}_{k,\mathrm{anc}(i)}\right)\right) \quad (4)$$

We use a binary addressing scheme to refer to individual nodes: Each node in a category's tree representation is addressed by a sequence of bits $a_0 a_1 a_2 \ldots a_T$, corresponding to a top-down traversal of the tree. The value $a_{t>0} = 0$ (or, 1) is interpreted as branching to the left (or, right) at depth t. The root a_0 has an arbitrary placeholder value (say, 1).[8] In the example in Figure 3, the inner NP argument (the argument of the top-level result) is addressed as 101. We represent the position of a node by a vector of elements in its address, mapping $a_{t>0} = 0$ to 1 and $a_{t>0} = 1$ to -1 and ignoring a_0. The slashes in node's ancestors are similarly mapped to a vector consisting of 1s for forward slashes and -1 for backward slashes. We use 0 to pad feature vectors to a fixed maximum length. 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.[9]

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 α as in equation (5) and then add the α-weighted context values to the hidden state, equation (6), replacing the simpler MLP from equation (2).[10]

$$\alpha = \mathrm{SoftMax}\left(\mathbf{h}_{k,i}\mathbf{H}_0^\top\right) \quad (5)$$

$$\mathbf{o}_{k,i} = \mathrm{MLP}\left(\mathbf{h}_{k,i} + \alpha\mathbf{H}_0\right) \quad (6)$$

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., NP when it occurs inside a complex category (S\NP)/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. (Because 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.

[7] Only Slash operators can have children (Figure 2).

[8] Prepending all addresses with 1 has several representational advantages, the most straightforward of which is that addresses can alternatively be read as binary numbers enumerating category pieces in breadth-first traversal.

[9] The featurized encoder is, to a large extent, made possible by fixing the arity and maximum depth of categories. The TreeRNN will likely better admit more general setups, where outputs of unbounded depth and/or variable arity are allowed.

[10] We also tried self-attention over previously predicted partial outputs but did not find an increase in performance.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

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 supertags, 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 development set performance.[11] 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 automatically obtained using Stanza (Qi et al., 2020).

[11] Preliminary experiments showed that best dev performance is usually reached within 10 epochs; batches larger than 8 make our (single) GPU run out of memory.

Hidden dim d	768	Weight decay	.01
Activation	gelu	LRs	1e-4, 1e-5 (ft)
Dropout	.2	Seeds	14112, 36125, 92225
AdamW β's	.9, .999		
AdamW ϵ	1e-6	Max cat depth	6

Table 1: Hyperparameters used in our experiments. We use separate learning rates (*LR*) for fine-tuning (*ft*) the RoBERTa-base model.

	CCGbank	Rebank
cat types	1,285	1,574
≥ 100	172	199
10–99 (medium rare)	253	312
< 10 (very rare)	860	1,063
atomic	34	37
sentences	39,604	39,604
tokens	929,552	943,204
medium rare cat	7,549	9,640
very rare cat	2,055	2,527

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

4.1 Model Details and Hyperparameters

In Table 1 we report the model and training hyperparameters 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 values for the AdamW hyperparameters. We follow Kogkalidis et al. (2019) in setting up the sequential Transformer model with 8 decoder heads and 2 decoder layers, but swap out the from-scratch encoder 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. (2010) ''rebanked'', i.e., corrected 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). Rebank has a larger and more diverse category space, due in large part to a more principled treatment of NP argument structure. Hence, we conduct our main experiments with Rebank and

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

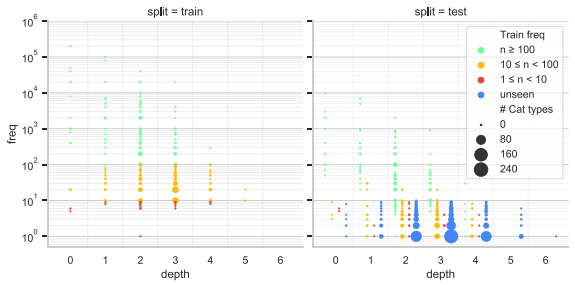


Figure 4: Shifting the tail to evaluation. The new test set (right) consists of those sentences in sections 02–21 that contain a category type occurring less than 10 times, and the new training set of the remaining sentences (left). As a result, we evaluate on many more category types that are not seen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

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 categories seen 1–9 times in training, and only 27 tokens 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 ($\geq$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 way—the distribution follows Figure 4.[12] In comparison with the default data splits (Figure 1b), we see that this sampling method captures precisely the long tail of categories, while leaving the rest of the category distribution largely unchanged.

For out-of-domain evaluation we use Honnibal et al. (2009) (English) Wikipedia gold standard and the (English) gold section of the Parallel Meaning Bank, v3.0, which comprises multiple text types, including literary and biblical texts (PMB; Abzianidze et al., 2017). The Wikipedia dataset follows CCGbank in terms of category conventions, while PMB is more similar to Rebank; we evaluate models trained on one style only on in- and out-of-domain test sets matching that style. That said, PMB contains an unusually large number of unseen categories following idiosyncratic conventions that even Rebank-trained models are unlikely to pick up on without additional training data.

[12]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.

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 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 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.

For comparison with existing work (Table 5), we also report results on the original CCGbank

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

Model	Acc	Acc by cat frequency in training				Acc by cat depth		
	All	≥100	10–99	1–9	OOV	0	1–2	3–6
	n=56,395	n=55,698	n=563	n=107	n=27	n=19,67	n=33,409	n=3,315
	N=538	N=199	N=222	N=91	N=26	N=18	N=253	N=267
Nonconstructive Classification								
MLP_10	**94.77** ± .07	**95.26** ± .07	**68.32** ± 1.42	–	–	**97.80** ± .14	**94.25** ± .08	82.01 ± .18
MLP_10@	**94.76** ± .17	**95.25** ± .18	**68.98** ± 0.89	–	–	**97.73** ± .16	**94.29** ± .23	81.88 ± .29
MLP_1	**94.83** ± .09	**95.27** ± .10	**68.68** ± 1.09	23.99 ± 1.08	–	**97.71** ± .16	**94.37** ± .14	**82.39** ± .11
MLP_1@	**94.75** ± .18	**95.18** ± .17	**70.16** ± 0.81	27.10 ± 1.62	–	**97.84** ± .18	**94.26** ± .52	**82.33** ± .51
Constructive: Sequential								
K+19	90.68 ± .15	91.10 ± .16	63.65 ± 0.21	**34.58** ± 1.62	**7.41** ± 0.00	91.71 ± .29	91.28 ± .02	78.43 ± .77
RNN	93.92 ± .01	94.39 ± .02	65.48 ± 0.62	19.00 ± 2.35	0.00 ± 0.00	95.25 ± .09	**94.33** ± .04	**81.77** ± .78
RNN@	94.48 ± .08	94.93 ± .04	66.90 ± 2.32	27.41 ± 5.31	1.23 ± 2.14	**97.72** ± .11	93.88 ± .09	81.33 ± .16
Constructive: Tree-structured								
TreeRNN	94.62 ± .12	**95.10** ± .11	64.24 ± 2.60	25.55 ± 0.54	2.47 ± 2.14	**97.70** ± .21	94.14 ± .08	81.14 ± .90
TreeRNN@	94.44 ± .22	94.95 ± .20	62.17 ± 3.03	22.43 ± 1.87	0.00 ± 0.00	97.61 ± .05	**93.95** ± .33	80.61 ± .63
AddrMLP	94.58 ± .16	95.01 ± .16	67.44 ± 1.45	**34.89** ± 2.35	3.70 ± 0.00	**97.73** ± .13	94.02 ± .17	81.47 ± .24
AddrMLP@	**94.70** ± .05	**95.11** ± .06	**68.86** ± 0.57	**36.76** ± 2.86	4.94 ± 2.14	**97.85** ± .16	94.11 ± .03	81.92 ± .26

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 $r + \text{stdev}(r) > b - \text{stdev}(b)$. For comparison, the overall tagging accuracy reported in Honnibal et al. (2010) is 92.2%.

Model	Acc	Acc by cat freq in training				Parsing	
	All	≥100	10–99	1–9	OOV	LF	Parseability
	n=55,371	n=54,825	n=442	n=82	n=22		n=2,407
	N=435	N=171	N=176	N=67	N=21		
Nonconstructive							
MLP_10@	96.09 ± .07	**96.50** ± .08	**67.27** ± 1.02	–	–	**90.78** ± .09	86.95 ± 0.75
MLP_1	**96.22** ± .06	**96.58** ± .07	**70.29** ± 2.35	23.17 ± 3.23	–	**90.91** ± .09	88.26 ± 0.39
Constructive							
K+19	92.12 ± .21	92.46 ± .20	65.38 ± 0.99	**34.55** ± 4.28	1.52 ± 2.62	87.66 ± .19	91.14 ± 0.13
RNN@	95.10 ± .07	95.48 ± .07	65.76 ± 1.71	26.02 ± 0.70	0.00 ± 0.00	90.63 ± .04	89.53 ± 0.18
AddrMLP@	**96.09** ± .07	**96.44** ± .08	**68.10** ± 1.38	**37.40** ± 1.41	**3.03** ± 2.62	**90.79** ± .08	86.03 ± 1.72

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 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 from Table 3 to Table 4 is consistent with Honnibal et al. (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

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

	Acc			Parsing	
Model	All	≥ 10	OOV	LF	P/ability
Nonconstructive					
V+16	94.24	–	–	88.32	–
C+18	96.05	–	–	–	–
T+20	–	96.39	–	90.68	–
Constructive					
BP20	96.00	–	5	90.9	96.2

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).

	Acc	Acc by cat freq			
	All	≥100	10–99	1–9	OOV
	n=53,765	*n*=50,754	*n*=989	*n*=292	*n*=1,730
Model	*N*=1,351	*N*=188	*N*=240	*N*=118	*N*=805
Nonconstructive					
MLP_10	88.76	92.86	**55.71**	13.24	–
MLP_1	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 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.

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 set (Table 7). These experiments largely confirm our findings from the standard Rebank evaluation set, while the change in category distribution has several important effects on our ability to evaluate model generalization: First, OOV performance is much higher on the redistributed data (Table 6) than on the standard test splits in Tables 3, 4, and 7, highlighting all of the constructive models' generalization capability, and in turn suggesting that the OOV categories in WSJ section 23 and PMB are truly difficult, noisy, or otherwise inconsistent with the training data. Second, the proportion of evaluation tokens of categories less than 10 times in training is 1.6% in PMB and 3.8% in our

	Wiki	PMB			
	Acc	All	≥100	10–99	1–9	OOV
	n=4,151	*n*=53,739	*n*=52,010	*n*=870	*n*=191	*n*=668
Model	*N*=138	*N*=243	*N*=129	*N*=47	*N*=14	*N*=53
Nonconstructive						
MLP_10	**92.54**	90.11	**92.10**	57.05	–	–
MLP_1	92.31	**90.27**	**92.10**	**63.41**	29.14	–
Constructive						
K+19	87.29	84.39	86.13	55.86	32.64	0.20
RNN	92.00	89.52	91.38	61.42	24.26	0.25
AddrMLP	92.46	90.16	92.02	59.00	**36.30**	**1.55**

Table 7: Performance of the best systems (the variants *with attention* for each paradigm) on the Wikipedia and PMB[13] datasets. The state of the art on the Wikipedia data is 90.00% (Xu et al., 2015).

redistributed Rebank evaluation data, compared to only ≈0.2% in the standard CCGbank and Rebank test sets. This 7x–16x increase in relative size renders the tail much more consequential for overall performance. And indeed we observe slightly smaller gaps in overall accuracy between the best-performing nonconstructive and the best-performing constructive systems in Table 7 (0.08 on Wiki, 0.11 on PMB) compared to 0.13 in Tables 3 and 4, while in Table 6 AddrMLP even clearly outperforms the nonconstructive models. Third, performance on rare and unseen categories can now be measured much more reliably due to the larger *absolute* counts of rare and unseen categories. We provide in-depth analyses of this subset of tags in § 6.2.

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 to the long tail may well outweigh any minor improvement on the most frequent categories when applied to even more diverse data, within other languages, and across languages.

[13]Because we did not train any models on PMB itself, we analyze performance on all of PMB-gold, but for future comparisons, we also report accuracy on the suggested evaluation split: K+19: 85.43%; RNN@: 90.24%; AddrMLP@: 90.78%; MLP_1@: 90.88%; MLP_10@: 90.91%.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

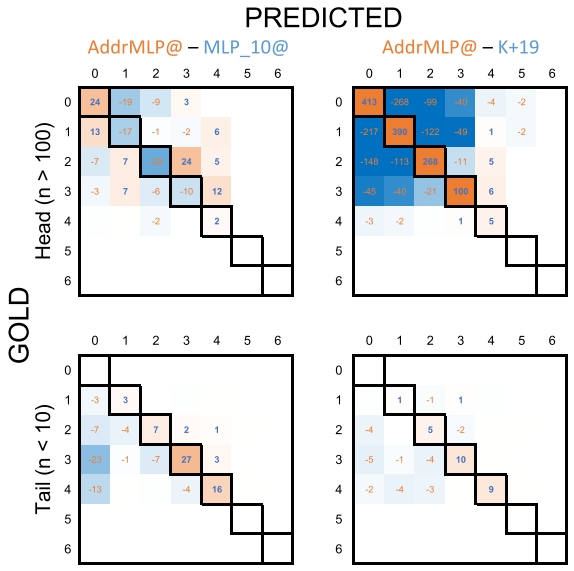


Figure 5: Confusion matrices by category depth, based on the standard Rebank evaluation set. Rows (columns) correspond to gold (predicted) categories with the respective depth. Thus, cells above (below) the diagonal refer to categories predicted too deep (shallow). All numbers are absolute differences between confusions made by AddrMLP@ and MLP_10@ / K+19, respectively. Thus, 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

Whereas 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 rightmost columns of Table 3 that it is not dramatically more difficult for constructive systems to generate complex categories of depth ≥ 1 than it is for nonconstructive 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 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

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

	garnered	from	1984 to 1986
Gold	**(S[pss]\NP)**	**(ADV/ADV)/NP**	
MLP_10	✓	✓	
MLP_1	✓	✓	
K+19	✓	✓	
RNN	✓	✓	
AddrMLP	(S[pss]\NP)/PP	(PP/ADV)/NP	

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.[14] 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_10		S[ng]\NP	ADV
MLP_1		S[ng]\NP	ADV
K+19		S[ng]\NP	ADV
RNN		(S[ng]\NP)/PP	S[adj]\NP
AddrMLP		✓	✓

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).

structure (✓ struct).[15] 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.

[14] 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.

[15] E.g., for ‘‘piling’’ in Table 9 the RNN predicts (S[ng]\NP)/PP, which exhibits the correct structure (X\X)/X with an incorrect atomic label (PP instead of PR).

	Why	constructive	?
Gold	**S[wq]/(S[adj]\NP)**	**S[adj]\NP**	
MLP_10	<UNKNOWN>	✓	
MLP_1	(S/S)/(S[adj]\NP)	✓	
K+19	✓	✓	
RNN	✓	✓	
AddrMLP	✓	✓	

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.

			Incorrect		
	Model	**Correct**	✓struct	✓formed	✗formed
All	MLP_10@	47,542	1,345	4,746	–
	MLP_1@	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
Invented	K+19	201	96	160	127
	RNN@	93	26	71	5
	TreeRNN@	162	83	213	1
	AddrMLP@	190	89	240	1

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).

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

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

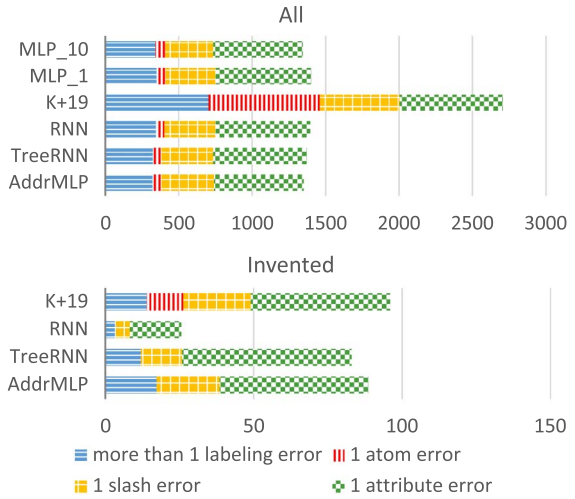


Figure 6: Fine-grained analysis of correctly structured but incorrectly labeled predictions (‘✓ struct’ in Table 11). ‘Attribute error’ means that the predicted atomic category is correct except for a wrong or missing linguistic attribute (e.g., S vs. S[dcl]); ‘atom error’ means that an entirely wrong atomic category has been chosen (e.g., PP vs. NP); and ‘slash error’ means confusing / and \.

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 a 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,[16] 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 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

[16]That TreeRNN and AddrMLP still produced one malformed category can be considered a bug: They attempted to generate a category deeper than the maximally allowed depth and were unable to complete it. This is avoidable in practice.

	bring
Predicted sequence	/ / \S[b] NP \NP
Predicted supertag	((S[b]\NP)/(NP_))/_
Gold sequence	/ \S[b] NP NP
Gold supertag	(S[b]\NP)/NP

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.

(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)\(NP\NP) (68 times in training), ((N\N)\(N\N))/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.

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

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

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

Model	Nouns n=16,946 N=83 f=1,158	Verbs n=7,915 N=296 f=129	Wh n=542 N=54 f=38	Other n=29,968 N=436 f=358
MLP_10@	98.58	93.18	92.25	95.51
MLP_1	**98.62**	93.49	92.68	**95.65**
K+19	95.58	90.54	90.04	90.62
RNN@	98.60	93.17	91.88	93.68
AddrMLP@	98.56	**93.62**	**93.11**	95.43

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.

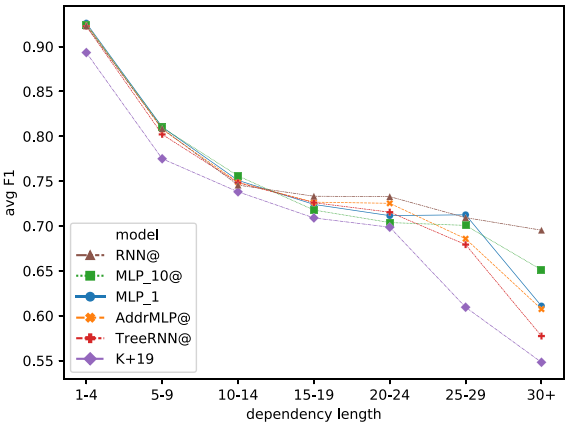


Figure 7: Parsing F1-score for varying dependency lengths, measured in terms of linear distance of the two words involved in the dependency.

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: 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]\S[dcl])\NP. These supertags have in common that they all contain sentential result/argument pairs of the form S[x]|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

[18] The C&C parser also reports coverage, the proportion of sentences for which at least one dependency relation can be recovered. Coverage is 100% in all our conditions.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

Model	Params *millions*	Train time *hours*	Infer speed *sents/s*
Nonconstructive Classification			
MLP_10	2.0	9	191
MLP_1	2.4	11	195
Constructive: Sequential			
K+19	11.8	120	0.3
RNN	4.8	68	135
Constructive: Tree-structured			
TreeRNN	8.3	10	125
AddrMLP	1.3	10	126

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.

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 and 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 concerned 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.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

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 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, (NP|NP)|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 (NP\NP)/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.

Acknowledgments

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 Hockenmaier, 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.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

References

Lasha Abzianidze, Johannes Bjerva, Kilian Evang, Hessel Haagsma, Rik van Noord, Pierre Ludmann, Duc-Duy Nguyen, and Johan Bos. 2017. The Parallel Meaning Bank: Towards a multilingual corpus of translations annotated with compositional meaning representations. In *Proceedings of EACL*, pages 242–247, Valencia, Spain. **DOI:** https://doi.org/10.18653/v1/E17-2039

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proceedings of EMNLP*, pages 1699–1710, Lisbon, Portugal.

Michael Auli and Adam Lopez. 2011. A comparison of Loopy Belief Propagation and Dual Decomposition for integrated CCG supertagging and parsing. In *Proceedings of ACL-HLT*, pages 470–480, Portland, OR, USA.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by jointly learning to align and translate. In *Proceedings of ICLR*. San Diego, CA, USA.

Jason Baldridge. 2008. Weakly supervised supertagging with grammar-informed initialization. In *Proceedings of COLING*, pages 57–64, Manchester, UK.

Jason Baldridge and Geert-Jan M. Kruijff. 2003. Multi-modal Combinatory Categorial Grammar. In *Proceedings of EACL*. Budapest, Hungary.

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

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of Neural Machine Translation: Encoder–decoder approaches. In *Proceedings of SSST-8*, pages 103–111, Doha, Qatar.

Noam Chomsky and Howard Lasnik. 1993. The theory of principles and parameters, Joachim Jacobs, Arnim von Stechow, Wolfgang Sternefeld, and Theo Vennemann, editors, *Syntax: An International Handbook of Contemporary Research*, 1, pages 506–569, Walter de Gruyter, Berlin, Germany.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? An analysis of BERT's attention. In *Proceedings of BlackboxNLP*, pages 276–286, Florence, Italy. **DOI:** https://doi.org/10.18653/v1/W19-4828, **PMID:** 31709923

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of EMNLP*, pages 1914–1925, Brussels, Belgium. **DOI:** https://doi.org/10.18653/v1/D18-1217

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

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING*, pages 282–288, Geneva, Switzerland.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4):493–552. **DOI:** https://doi.org/10.1162/coli.2007.33.4.493

Stephen Clark, Darren Foong, Luana Bulat, and Wenduan Xu. 2015. The Java version of the C&C Parser: Version 0.95. *Technical report, University of Cambridge Computer Laboratory*.

Tejaswini Deoskar, Christos Christodoulopoulos, Alexandra Birch, and Mark Steedman. 2014. Generalizing a strongly lexicalized parser using unlabeled data. In *Proceedings of EACL*, pages 126–134, Gothenburg, Sweden. **DOI:** https://doi.org/10.3115/v1/E14-1014

Tejaswini Deoskar, Markos Mylonakis, and Khalil Sima'an. 2013. Learning structural dependencies of words in the Zipfian tail. *Journal of Logic and Computation*, 24(2):433–453. **DOI:** https://doi.org/10.1093/logcom/exs062

Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. 2015. A supertag-context model

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

for weakly-supervised CCG parser learning. In *Proceedings of CoNLL*, pages 22–31, Beijing, China. **DOI:** https://doi.org/10.18653/v1/K15-1003

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of NAACL-HLT*, pages 4129–4138, Minneapolis, MN, USA.

Julia Hockenmaier and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396. **DOI:** https://doi.org/10.1162/coli.2007.33.3.355

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

Matthew Honnibal, Joel Nothman, and James R. Curran. 2009. Evaluating a statistical CCG parser on Wikipedia. In *Proceedings of People's Web*, pages 38–41, Suntec, Singapore. **DOI:** https://doi.org/10.3115/1699765.1699771

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of ACL*, pages 3651–3657, Florence, Italy. **DOI:** https://doi.org/10.18653/v1/P19-1356

Rekia Kadari, Yu Zhang, Weinan Zhang, and Ting Liu. 2018. CCG supertagging via Bidirectional LSTM-CRF neural architecture. *Neurocomputing*, 283:31–37. **DOI:** https://doi.org/10.1016/j.neucom.2017.12.050

Jungo Kasai, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr. 2017. TAG parsing with neural networks and vector representations of supertags. In *Proceedings of EMNLP*, pages 1712–1722, Copenhagen, Denmark. **DOI:** https://doi.org/10.18653/v1/D17-1180

Jungo Kasai, Robert Frank, Pauli Xu, William Merrill, and Owen Rambow. 2018. End-to-end graph-based TAG parsing with neural networks. In *Proceedings of NAACL-HLT*, pages 1181–1194, New Orleans, LA, USA. **DOI:** https://doi.org/10.18653/v1/N18-1107

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *Proceedings of ICLR*, Toulon, France.

Konstantinos Kogkalidis, Michael Moortgat, and Tejaswini Deoskar. 2019. Constructive type-logical supertagging with self-attention networks. In *Proceedings of RepL4NLP*, pages 113–123, Florence, Italy. **DOI:** https://doi.org/10.18653/v1/W19-4314

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of EMNLP*, pages 1512–1523, Edinburgh, Scotland, UK.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG parsing. In *Proceedings of NAACL-HLT*, pages 221–231, San Diego, CA, USA. **DOI:** https://doi.org/10.18653/v1/N16-1026 **PMCID:** PMC5024747

Tao Li, Vivek Gupta, Maitrey Mehta, and Vivek Srikumar. 2019. A logic-driven framework for consistency of neural models. In *Proceedings of EMNLP-IJCNLP*, pages 3924–3935, Hong Kong, China. **DOI:** https://doi.org/10.18653/v1/D19-1405, **PMID:** 31251625, **PMCID:** PMC6767096

Tao Li and Vivek Srikumar. 2019. Augmenting neural networks with first-order logic. In *Proceedings of ACL*, pages 292–302, Florence, Italy. **DOI:** https://doi.org/10.18653/v1/P19-1028

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. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic gradient descent with warm restarts. *Proceedings of ICLR*.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*. New Orleans, LA, USA.

Maria Nădejde, Siva Reddy, Rico Sennrich, Tomasz Dwojak, Marcin Junczys-Dowmunt, Philipp Koehn, and Alexandra Birch. 2017. Predicting target language CCG supertags improves Neural Machine Translation. In *Proceedings of WMT*, pages 68–79, Copenhagen, Denmark. **DOI:** https://doi.org/10.18653/v1/W17-4707

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of ACL*, pages 101–108, Online.

Sujith Ravi, Jason Baldridge, and Kevin Knight. 2010. Minimized models and grammar-informed initialization for supertagging with highly ambiguous lexicons. In *Proceedings of ACL*, pages 495–503, Uppsala, Sweden.

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

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured Long Short-Term Memory networks. In *Proceedings of ACL-IJCNLP*, pages 1556–1566, Beijing, China.

Emily Thomforde and Mark Steedman. 2011. Semi-supervised CCG lexicon extension. In *Proceedings of EMNLP*, pages 1246–1256, Edinburgh, Scotland, UK.

Yuanhe Tian, Yan Song, and Fei Xia. 2020. Supertagging Combinatory Categorial Grammar with Attentive Graph Convolutional Networks. In *Proceedings of EMNLP*, pages 6037–6044, Online. **DOI:** https://doi.org/10.18653/v1/2020.emnlp-main.487, **PMID:** 32060988

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with LSTMs. In *Proceedings of NAACL-HLT*, pages 232–237, San Diego, CA, USA. **DOI:** https://doi.org/10.18653/v1/N16-1027

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 *Proceedings of NeurIPS*, pages 5998–6008, Long Beach, CA, USA.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280. **DOI:** https://doi.org/10.1162/neco.1989.1.2.270

Huijia Wu, Jiajun Zhang, and Chengqing Zong. 2017. A dynamic window neural network for CCG supertagging. In *Proceedings of AAAI*. San Francisco, CA, USA.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. In *Proceedings of ACL-IJCNLP*, pages 250–255, Beijing, China.

Xingxing Zhang, Liang Lu, and Mirella Lapata. 2016. Top-down Tree Long Short-Term Memory networks. In *Proceedings of NAACL-HLT*, pages 310–320, San Diego, CA, USA. **DOI:** https://doi.org/10.18653/v1/N16-1035

Zhenqi Zhu and Anoop Sarkar. 2019. Deconstructing supertagging into multi-task sequence prediction. In *Proceedings of CoNLL*, pages 12–21, Hong Kong, China.

Downloaded from http://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00364/1924126/tacl_a_00364.pdf by guest on 26 September 2022