Supertagging with CCG primitives

Aditya Bhargava **Gerald Penn**
Department of Computer Science
University of Toronto
Toronto, ON, Canada M5S 3G4
{aditya,gpenn}@cs.toronto.edu

Abstract

In CCG and other highly lexicalized grammars, supertagging a sentence's words with their lexical categories is a critical step for efficient parsing. Because of the high degree of lexicalization in these grammars, the lexical categories can be very complex. Existing approaches to supervised CCG supertagging treat the categories as atomic units, even when the categories are not simple; when they encounter words with categories unseen during training, their guesses are accordingly unsophisticated.

In this paper, we make use of the primitives and operators that constitute the lexical categories of categorial grammars. Instead of opaque labels, we treat lexical categories themselves as linear sequences. We present an LSTM-based model that replaces standard word-level classification with prediction of a sequence of primitives, similarly to LSTM decoders. Our model obtains state-of-the-art word accuracy for single-task English CCG supertagging, increases parser coverage and F_1, and is able to produce novel categories. Analysis shows a synergistic effect between this decomposed view and incorporation of prediction history.

1 Introduction

Highly lexicalized grammars, such as lexicalized tree-adjoining grammar (LTAG) and combinatory categorial grammar (CCG), have very large sets of possible lexical categories. Where most phrase-structure and dependency grammars have lexical category sets numbering in the tens for English (Taylor et al., 2003), LTAG and CCG have sets numbering in the hundreds or thousands (Joshi and Srinivas, 1994; Clark, 2002). The large number of possible labels for each word can make the search space for the syntactic tree of the sentence intractably large; narrowing the set of viable lexical categories per word is therefore an important step in efficient parsing for such grammars (Clark and Curran, 2007; Lewis et al., 2016). As the tags are much more complex and informative than part-of-speech (POS) tags, tagging the words with these more complex categories is called **supertagging**.

Category	Count
N	206,312
N/N	152,508
NP_{nb}/N	83,377
$(NP\backslash NP)/NP$	43,700
$((S\backslash NP)\backslash(S\backslash NP))/NP$	22,189
conj	20,170
NP	19,749
PP/NP	17,199
$(S\backslash NP)\backslash(S\backslash NP)$	16,146
$((S\backslash NP)\backslash(S\backslash NP))/((S\backslash NP)\backslash(S\backslash NP))$	3,820
$(((S\backslash NP)\backslash(S\backslash NP))\backslash((S\backslash NP)\backslash(S\backslash NP)))/NP$	325

Figure 1: Some sample CCG lexical categories from the CCGbank training set. The first nine are the most frequent non-punctuation categories. The final two are in the top 100 (out of 1285) and illustrate the capacity for syntactic richness and variety in complexity.

The large number of lexical categories comes from the high degree of complexity that the categories can have. When grammars have small tag sets, the bulk of the work in developing or learning a grammar comes from deciding how to combine the tags and their words. Categorial grammars instead have fewer combination rules, requiring the lexical categories to support much greater syntactic richness; see Figure 1 for some sample categories.

Existing approaches to supervised supertagging operate in the same manner as POS taggers: as a word classifiers, predicting the correct tag from a fixed set. This is relatively straightforward for POS tags: there are relatively few possibilities as the tags are simple–*e.g.*, it is not immediately apparent if or how *VBD* is more complex than *NNP*. By con-

Proceedings of the 5th Workshop on Representation Learning for NLP (RepL4NLP-2020), pages 194–204

trast, CCG categories have varying complexities and are clearly not atomic units; they are composed from a much smaller vocabulary of primitives.

In this paper, we challenge the usual treatment of CCG supertagging as large-tagset POS tagging, instead treating lexical categories as the complex units that they are. We present a model for CCG supertagging that replaces traditional whole-category prediction with the prediction of their composing primitives. In addition to addressing the incongruity between POS tags and CCG categories, this allows for the generation of new categories that do not occur in the training set, a necessary property for handling the long tail of syntactic phenomena.

We treat supertags as linear sequences, enabling us to employ LSTM decoders to autoregressively predict CCG primitives in sequence. On CCGbank, our model outperforms a bidirectional LSTM classification baseline on word accuracy, parser F_1, and parser coverage, establishing a new state-of-the-art for single-task English CCG supertagging.

Analysis of our model and results shows that our non-atomic view of CCG lexical categories enables more effective incorporation of model prediction history than is the case with atomic category classification. Our model can also generate new categories that it has not seen during training, and even manages to correctly label some words with such out-of-vocabulary (OOV) categories. To the best of our knowledge, our model is the first fully-supervised CCG supertagger that constructs lexical categories from primitive types, and the first to be able to produce OOV categories. Our work presents both a more appropriate view of the problem and establishes a strong baseline for CCG supertagging according to this view.

2 Background and motivation

Supertagging is quite different from POS tagging. CCG lexical categories are composed from a fixed set of more **primitive** units; as a result, CCG supertagging has a much larger set of possible tags than does POS tagging—an open set, in fact. Where the Penn Treebank (PTB) has 48[1] POS tags (Taylor et al., 2003), CCGbank has 1322 lexical categories (Hockenmaier and Steedman, 2007). Selecting from a much larger set is more difficult, of course, and therefore, POS tagging accuracy is substantially higher than for CCG supertagging. Recent POS tagging work has reached up to 98% accuracy and above, depending on the language and corpus, *without* the use of pre-trained embeddings or other incorporation of external corpora (Plank et al., 2016). English CCG supertagging, meanwhile, has only recently broken past 96%, accuracy and that too with a heavy dependence on pre-trained embeddings, external corpora, and/or multi-task training (Clark et al., 2018).

Given these substantial differences, the complex, structured nature of CCG lexical categories warrants further investigation for supertagging. We see two primary advantages in doing so. First, as noted by Baldridge (2008) and Garrette et al. (2014), a compositional view of lexical categories can provide strong information about surrounding categories. For example, if a word has category S/NP, then it is likely that there is a primitive NP type *somewhere* else in the sentence, whether as a simple category or as part of a complex one.

Second, treating CCG categories as atomic makes it impossible to fully tag all new data, since new, rare categories may be encountered during inference. Such rare categories are not necessarily spurious; over the whole CCGbank, Hockenmaier and Steedman (2007) note that while some of the once-occurring categories "are due to noise or annotation errors, most are in fact required for certain constructions."[2] Admittedly, novel categories (*i.e.*, those not occurring in the training set) are rare in CCGbank: in the standard splits, 0.06% of word tokens in the development set and 0.04% in the test set are tagged with a category that does not occur in the training set. But since an incorrect lexical category can impair the parsability of a full sentence, it is more appropriate to consider the number of affected *sentences*, which is 0.9% for both the development and test sets.[3] Work on CCG parsers has noted their high sensitivity to supertagging accuracy (*e.g.*, Clark and Curran, 2004; Lewis et al., 2016), so such cases should not be ignored. And unlike typical classification scenarios, out-of-vocabulary lexical categories are not different in kind from in-vocabulary ones; they are composed from the same units using the same rules, suggesting that OOV categories can, in principle, be treated in a concordant manner.

[1] Twelve of which are for punctuation.

[2] They provide relative pronouns in pied-piping constructions and verbs which take expletive subjects as examples; we found lengthy adjunction chains to contribute as well.

[3] These proportions are even higher for out-of-domain data (Rimell and Clark, 2008).

3 Related work

While our focus in this paper is on CCG supertagging, it is worth noting that the supertagging task originates in the context of LTAG (Joshi and Srinivas, 1994; Bangalore and Joshi, 1999), which also has highly complex lexical categories. Supertagging is important for efficient parsing in such grammars as it helps narrow the search space for the parse (Clark and Curran, 2004).

Despite the complexity captured in supertags, the vast majority of existing approaches treat CCG lexical categories as atomic units for prediction, ignoring their varying complexities and structured nature. Effectively, at each time step of the input sentence, the model must decide which of a fixed set of CCG categories is the best choice. This category-classification approach is the same as for POS tagging, and indeed, existing supertagging models are very similar to (if not the same as) POS tagging models in structure.

Early work for CCG supertagging relied on maximum-entropy models with hand-specified features, a limited set of possible categories, and tag dictionaries that tracked allowed categories for frequent words based on the training data (Clark, 2002; Clark and Curran, 2004, 2007). Recent work relies heavily on word embeddings: they allow better handling of out-of-vocabulary words and decrease reliance on part-of-speech tags, where imperfect accuracy can be a detriment for the supertagger. Lewis and Steedman (2014) used externally-trained embeddings (Turian et al., 2010) combined with suffix and capitalization features in a simple feed-forward neural network as well as a CRF; they also allowed words to be tagged with categories with which they did not co-occur in the training data. Xu et al. (2015) applied the same embeddings and features in a standard Elman RNN (Elman, 1990); they later improved their model by making it bidirectional (Xu et al., 2016). Lewis et al. (2016) replaced the Elman RNNs with two-layer bidirectional LSTMs, taking advantage of the LSTM units' ability to retain information over time (Hochreiter and Schmidhuber, 1997), and incorporated a data-augmentation technique as well.

Vaswani et al. (2016) used a single-layer bidirectional LSTM but dropped all hand-specified features, removed the limit on the categories that the model could produce, used custom in-domain word embeddings, and included a language model–style LSTM over the output lexical categories that allowed the model to condition its predicted tag at time t on the previously-predicted lexical category at time $t-1$. In addition to improving word accuracy, this latter addition drastically increased the number of tagged sentences that were parsable, even in variations that hurt word accuracy.

The current state-of-the-art result in CCG supertagging was achieved by Clark et al. (2018). Their model consisted of a two-layer bidirectional LSTM with GloVe word embeddings (Pennington et al., 2014) supplemented by the output of a character-level convolutional neural network. Their approach involved training additional "auxiliary" prediction modules on top of the same LSTM on an additional, unlabelled corpus (Chelba et al., 2014). These auxiliary modules were given an incomplete view of the input (*e.g.*, only words to the left) and trained to predict the same label that the primary prediction module predicted.

If we consider other grammars, (Kogkalidis et al., 2019) presented a type-logical grammar for Dutch and a supertagging approach that relied on primitive units. While their approach yielded improvement in word accuracy, the overall accuracy was substantially lower than with CCG supertagging; furthermore, the grammar's type system was so different that it is difficult to draw conclusions about applicability to other grammars.[4] In order to see some consideration of the composed structure of *CCG* lexical categories, we must alter our task scope somewhat. Garrette et al. (2014), following earlier work (Baldridge, 2008), applied a Bayesian model with grammar-informed priors for supertagging where only a tag dictionary and raw, unlabelled text was made available. Their model included a generative model for categories as well as the notion of *combinability*, preferring tag sequences where adjacent words could be combined via CCG rules. Similarly, work in CCG grammar induction has involved some basic consideration of how CCG categories are constructed so that that a grammar could be built using EM (Bisk and Hockenmaier, 2012) or hierarchical Dirichlet processes (Bisk and Hockenmaier, 2013). Despite these applications, consideration of CCG primitives has yet to make its way to supervised supertagging approaches; we aim to fill that gap.

[4] Their grammar had 5700 unique types for a corpus of 65k sentences; categories were constructed from 30 atomic types, corresponding to POS tags or phrasal categories, and 22 *non-directional* binary connectives, corresponding to dependency labels.

4 Method and model

Despite the potential advantages discussed above in Section 2, it is unclear *a priori* whether supertagging with primitive units is more or less difficult than standard, whole-category classification. While the output vocabulary becomes drastically smaller, the output sequences are longer and must be arranged correctly. One of our aims in this paper is to establish a baseline for this approach to supertagging and evaluate its difficulty as a task in comparison to the usual methods.

4.1 Linearization

Lexical categories in categorial grammars are composed of a relatively small, fixed set of primitive types (*S*, *NP*, etc.) with indications for precedence/grouping (parentheses) and ordering (forward and backward slashes). In this paper, we approach the generation of CCG lexical categories as the prediction of a linear sequence of decomposed symbols. We use a simple linearization scheme: we split each lexical category label into tokens, using parentheses and slashes as delimiters. We keep the delimiters as units in the output sequence as well, as they crucially define the structure of the category. This linearization method yields an output vocabulary of size 38 (including parentheses and slashes); many of these are feature-typed versions of plain primitive types; *e.g.*, S_{to} and N_{num}. We refer to all units resulting from this decomposition, whether they are primitive types, slashes, or parentheses, as primitives for brevity.

It may seem difficult to try to learn to predict a sequence such as {*(*, S_{dcl}, \, *NP*, *)*, /, *NP*} consistently and correctly, or to produce sequences in general that are well-formed. Any model attempting this will have to implicitly learn the rules for constructing lexical categories from primitives, such as the balancing of parentheses, or that primitive types cannot occur directly adjacent to one another and must be joined with a slash. But recent work suggests that this is not an unreasonable ask: Vinyals et al. (2015) used a similarly simple linearization scheme to convert a constituent parse tree into a sequence predictable by a linear decoder. The model did produce malformed parses on occasion, such as by forgetting to close open parentheses, but in general, it was able to perform near or above the state-of-the-art at the time, depending on how much data were used for training.

4.2 Decoding sequences of primitives

The most recent, highest-performing supertaggers are all based on bidirectional LSTM architectures. At each time step, the forward and backward LSTM outputs are combined and fed through a softmax layer to produce a distribution over categories. In order to construct a supertagger that works at the level of primitives, we propose a model that replaces the softmax prediction layer with a separate LSTM that predicts primitives in a manner similar to the decoder in RNN encoder-decoder architectures (Cho et al., 2014; Sutskever et al., 2014), or to how text is generated from neural language models.

In encoder-decoder LSTM models, an encoder LSTM is run over the input sequence. The final LSTM cell is used to initialize the decoder's LSTM cell, after which the decoder is trained to predict the output sequence. During training, the decoder receives as input the correct output for time $t-1$, and asked to predict the output for time t. During inference, the model makes its predictions autoregressively, since the correct previous output is unknown at test time. Output sequences are padded with `[START]` and `[STOP]` symbols: the former allows the model to learn a distribution over initial output symbols, as well as providing a means to trigger the output sequence prediction process (*e.g.*, after an input sentence has been read by the encoder); the latter is how the decoder indicates its completion of the current sequence.

Standard use cases for encoder-decoder models, such as machine translation, have the property that the output sequence lengths are not easily determinable from the input sequence lengths; nor is there an easy, strictly monotonic correspondence between input and output tokens. The usual application of encoder-decoder models handles this discrepancy by mostly separating the encoding and decoding parts of the model, leaving them connected only at their ends (*i.e.*, via the copying of the encoder's hidden state to the decoder's).

A naive application of encoder-decoder models to supertagging would simply output the sequence of categories (or primitives, in our case) for a sentence, after having encoded the entire input. For supertagging, this would be less than ideal. If one were to treat the sequence of categories as the target output sequence, there would be a long path through the network from the input word to the output supertag. One could remedy this with atten-

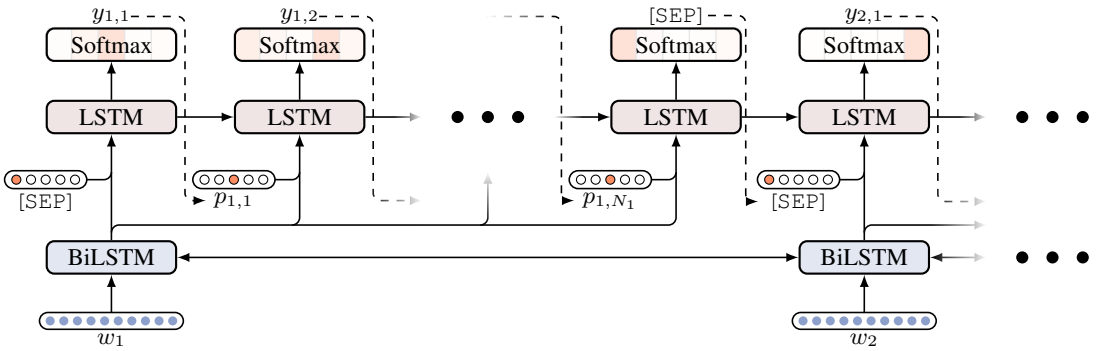

Figure 2: Our supertagging model. Where traditional models would classify an entire category for each word w_i at a time, we decode a sequence of primitives $y_{i,1}, \ldots, y_{i,N_i}$. The BiLSTM forward/backward combination layer is omitted for brevity.

tion mechanisms, but since there is a known and direct correspondence between a given encoder step and the output time steps, it is simpler to link the encoder output to the decoder directly.[5]

Our model, illustrated in Figure 2, consists of a bidirectional LSTM over the input sentence words, a feed-forward layer to combine and project the two LSTM directions, and finally by a unidirectional LSTM to produce sequences of primitives.[6] We refer to the bidirectional (base) LSTM as the encoder and the unidirectional primitive LSTM as the decoder to help differentiate the two, even though our use isn't exactly the same as in standard encoder-decoder models. Instead of initializing the decoder's cell with an encoder's final cell state, we directly use the encoder's output as inputs to the decoder, concatenated with the primitive from the previous time step.[7] During training, it is known which primitives correspond to which words, so aligning the encoder outputs to the decoder inputs is straightforward. During inference, we maintain a pointer i to select the relevant encoder output, initialized to $i = 1$. Then, whenever the decoder predicts the end of the current word's category, i is incremented so that the next decoder step gets the correct encoder output; decoding is stopped when the decoder predicts the end of the last word's category. Since one word's `[STOP]` symbol indicates the next word's `[START]` symbol, we combine the two symbols into a single `[SEP]` symbol, which can be interpreted as a word boundary marker.

Importantly, we do not reset the model state between words. This enables the decoder to maintain a memory of the primitives (and, by extension, categories) previously predicted in the sentence.

5 Experimental setup

5.1 Data

As is standard, we train our model on sections 2–22 of CCGbank (Hockenmaier and Steedman, 2005), keep section 0 for development and tuning, and evaluate on section 23. As required for decoding, we decompose the categories for each word, inserting the `[SEP]` token at word boundaries.

To represent the input words at the lowest level of our model, we use the (frozen) 5.5B ELMo embeddings (Peters et al., 2018). Because ELMo embeddings are cased and character-based, we need very little preprocessing of the input data: we convert all "*n't*" tokens to "*'t*" and append "*n*" to the preceding token, unless it is "*can*" or "*won*"; we convert the bracket tokens to their original characters (*e.g.*, "*-LRB-*" to "*(*", etc.); and we replace "\/" and "*" with "/" and "*" respectively. These steps are solely to more closely match what the ELMo model saw during its training. The inputs are otherwise untouched. There is also no need for a separate token for unknown words. Comparing previous work indicates that ELMo (Clark et al., 2018) drastically outperforms GloVe (Wu et al., 2017) and even custom WSJ-trained word embeddings (Vaswani et al., 2016); we also observed this difference ourselves during early development.

5.2 Evaluation

In order to control for minor implementation differences, we implement a baseline classification-

[5] Our initial (non-exhaustive) tests found no benefit to adding attention to our model, instead serving only to increase memory usage and slow training down.

[6] We stick with LSTMs, as with previous work, in order to conduct a well-controlled comparison.

[7] We did experiment with priming the decoder's initial cell state; our tests found this to yield a lower word accuracy compared to including the encoder output in the decoder input, and there was no benefit to doing both.

based bidirectional LSTM supertagger. This model is the same as ours shown in Figure 2, but replaces our decoder LSTM with a softmax layer, producing one category prediction per word. All recent supertagging work has been based on this architecture, with minor variations. We refer to the baseline as **BILSTM** and our model as **PRIMDECODER**.

Since our decoder can maintain a history of previous outputs, it may be better able to produce a sequence of supertags that form a parsable sentence, even if it makes mistakes on individual words. Therefore, in addition to the usual word accuracy and parser labelled F_1, we also measure parser coverage; we use the Java version of the C&C parser (Clark et al., 2015) to parse the sentences with our predicted supertags and gold part-of-speech tags. Coverage denotes the percentage of sentences for which the parser yields a complete parse, even if the derivation is not exactly correct; it therefore serves as a measure of how well the supertagging model is learning to be syntagmatically consistent, according to the rules of the relevant grammar. Lastly, since our model has the ability to generate arbitrary tags, we additionally measure word accuracy on word tokens tagged with OOV categories. Since the parser cannot handle OOV categories, we instead give it the predicted tag from the baseline in the cases where our model generates novel tags.

5.3 Model and training details

All layers in our models other than the softmax layer use size 512. Our LSTMs use standard LSTM activations (sigmoid for the gates, tanh for the state) and we use ReLU activations (Nair and Hinton, 2010) for the layer that combines the forward and backward encoder LSTMs. ReLU layer weights are initialized according to He et al. (2015), LSTM recurrent weights according to Saxe et al. (2013), and all other weights according to Glorot and Bengio (2010). We apply variational recurrent dropout (Gal and Ghahramani, 2016) throughout our model[8], including on the embeddings, except on the encoder output that is fed to the decoder, as we found it detrimental in initial tests. For the same reason, we do not use layer normalization on the ELMo embeddings, pre-trained primitive embeddings in the decoder (standard decoders typically take pre-trained word embeddings as inputs), attention, or scheduled sampling.

We train our models with the Adam optimizer (Kingma and Ba, 2014) for 25 epochs, halving the learning rate whenever there is no improvement in the development set loss, and keep the model weights from the epoch with the best development set accuracy. Training examples are sorted by output sequence length to yield efficient batches; the batches are subsequently processed in a semi-shuffled order, with batches being read through a shuffling buffer. We clip gradients, scaling accordingly, if the sum of gradient norms exceeds 1. During inference, we impose a maximum length on each word's predicted category; the maximum length is set to that of the longest category in the training set. Post-processing of the decoder outputs is limited to the removal of redundant parentheses.

Our models have four hyperparameters: the initial learning rate, the dropout rate on the input (*i.e.*, on the ELMo embeddings), the dropout rate on the output immediately prior to the softmax layer, and dropout rate elsewhere in the model. We tune the hyperparameters over 50 value sets sampled according to the tree-structured Parzen estimator method (Bergstra et al., 2011) as implemented in the Optuna[9] package.[10] The initial learning rate is sampled from a log-uniform distribution on $[10^{-4}, 10^{-2})$ while the dropout rates are independently sampled from a uniform distribution on $[0, 0.8)$. For each hyperparameter value set, we train the model five times with different random seeds and select the values yielding the best accuracy on the development set. We use the best values to run each model with 15 additional seeds so that we have a better estimate of the variance in model performance over random initializations. For PRIMDECODER, we decode the output sequences greedily for the hyperparameter search but evaluate with beam search, with a beam width of 5.

6 Results

Table 1 summarizes our main results. Our model outperforms the baseline on all measures

The Cochran-Mantel-Haenszel (CMH) test indicates that the difference in test set word accuracy between our model and the baseline is statistically significant ($p \approx 1.6 \times 10^{-7}$); likewise for the difference in coverage ($p \approx 0$).[11] Averaging sentence

[8] For dropout directly between adjacent LSTM states, we use the same dropout mask not just at each time step, but for all sentences in a batch. This allows us to use recurrent dropout with the fast cuDNN LSTM implementation.

[9] https://optuna.org/

[10] We were able to execute many runs in parallel, resulting in sampling more akin to standard random sampling.

[11] For a single run, McNemar's is the usual test. Since we

	Development set				Test set			
Model	Acc	OOV	F_1	Cov	Acc	OOV	F_1	Cov
Clark et al. (2018)								
CVT	—	0	—	—	95.7	0	—	—
ELMo-based	—	0	—	—	95.8	0	—	—
BiLSTM	96.15	0	90.6	87.0	95.89	0	90.2	84.6
PrimDecoder	**96.27**	**11**	**91.3**	**96.0**	**96.00**	**5**	**90.9**	**96.2**

Table 1: Our model's word accuracy, OOV category word accuracy, parser F_1, and parser coverage on CCGbank, compared to the bidirectional LSTM classifier baseline and comparable results from the most recent previous work. All accuracies are averaged over 20 runs with different random seeds. Standard deviations range around 0.05% for word accuracy, 0.1% for F_1, 0.4% for BiLSTM coverage, and 0.2% for PrimDecoder coverage. All improvements are statistically significant with $p \ll 0.001$.

F_1 scores over the 20 runs, the Wilcoxon signed-rank test indicates statistical significance for the difference in F_1 scores ($p \approx 6.7 \times 10^{-15}$).

It is extremely rare for our model to produce malformed categories. *Over all 20 runs*, our model produces only two instances of redundant parentheses, which are automatically repaired, and seven instances of malformed categories[12], which are left as-is and therefore counted as incorrect predictions. The malformations consist entirely of missing closing parentheses or extraneous opening parentheses.

6.1 Comparison to Clark et al. (2018)

In addition to besting the baseline, our model also yields a higher word accuracy than the single-task models reported by Clark et al. (2018). The focus of their work was their novel cross-view training (CVT) approach, which allowed for efficient and effective augmentation of model performance using *unlabelled* data. They compared their approach to the alternative use of ELMo over a variety of tasks, and CCG supertagging was the only one in which CVT underperformed the incorporation of ELMo. Their result with the ELMo-based model set the state-of-the-art word accuracy for single-task CCG supertagging.

It is therefore worth briefly discussing the similarities and differences between their ELMo-based model and our baseline. Their models used two-layer LSTMs with hidden units of size 1024, projected to 512 units between/after layers; our baseline has a single layer of width 512. Where we simply include ELMo representations as inputs to our models, they followed the recommendation of Peters et al. (2018) to include GloVe representations along with ELMo as well as to additionally provide the ELMo representations to the final output layer of the model. Since our baseline model is smaller and simpler than theirs but both are ELMo-based, it is mildly curious that our baseline outperforms their model. We expect that this difference is attributable to minor differences in implementation details.[13]

For reference, Clark et al. (2018) also reported a word accuracy of 96.0% if they trained their CVT-based model, but only if it was trained in a multi-task setting.[14] This constitutes a separate task, so the results are not directly comparable, but we note that our model achieves the same accuracy without the necessity for multi-task training, which could presumably benefit our model as well.

6.2 Novel categories

Of course, prior work as well as the baseline model cannot handle OOV categories at all, and accordingly have zero accuracies for such categories. Our model *can* generate novel categories, and can even do so correctly, though the accuracy is admittedly low, around 5% on the test set. These results indicates that our model is not merely memorizing the sequences of primitives that constitute the categories in the training set, but is learning some notion of the structure of CCG lexical categories and how subcategorical units are related among words.

Although we cannot make general claims about

have multiple runs, the CMH test is appropriate. The CMH test reduces to McNemar's test in the case of a single run.

[12] Which is to say, an average of 0.35 instances over a single run over the test set, which has around 55k words.

[13] For example, our decision to match the tokenization that ELMo saw during its training may have contributed to this difference.

[14] Or 96.1% with a *much* larger model, with LSTMs of width 4096.

when our model generates novel categories, it is still interesting to look at the cases where it does. We discuss some examples below where our model consistently generates novel categories, excerpting or rephrasing sentences for brevity as needed.

- In the sentence "*She was prosecuted under a law that makes it a crime to breach test security.*", the word "*makes*" has OOV category $(((S_{dcl}\backslash NP)/(S_{to}\backslash NP))/NP)/NP_{expl}$, which our model gets correct. The baseline predicts a similar (incorrect) tag where the final primitive is NP instead of NP_{expl}. Our model seems to pick up on contextual cues to generate the correct category; there are other such cases where our model selects the correctly typed primitive over the baseline.

- In the phrase "*Edward L. Cole, Jackson, Miss., $ 10,000 fine*", the "*$*" has OOV category $((NP\backslash NP)/(NP\backslash NP))/N_{num}$, modifying the word "*fine*" with category $NP\backslash NP$. Our model correctly generates the new category while the baseline incorrectly predicts $(N/N)/N_{num}$.

- In "*..., as has been the case...*", both the baseline and our model incorrectly tag "*as*" with $((S\backslash NP)\backslash(S\backslash NP))/S_{inv}$ while the correct tag is $((S\backslash NP)\backslash(S\backslash NP))/(S_{dcl}\backslash NP)$. Then, for "*has*", our model generates the *incorrect but novel* category $S_{inv}/(S_{pt}\backslash NP)$ in place of the correct $(S_{dcl}\backslash NP)/(S_{pt}\backslash NP)$ and in contrast to the baseline's $(S_{pss}\backslash NP)/(S_{pt}\backslash NP)$. Our model adjusted for its error and thus produced a parsable sequence.

6.3 Improvement effect analysis

Although PRIMDECODER outperforms the baseline on all fronts, there is a potential confound in determining which aspect of the model is responsible for this improvement. PRIMDECODER differs from BILSTM in two respects: the production of primitives and knowledge of prediction history. Since previous work has found that incorporating prediction history can increase both word accuracy and parser coverage (Vaswani et al., 2016), we cannot immediately attribute PRIMDECODER's higher performance to production of primitives alone.

In order to isolate these effects, we test two additional model variants. First, to examine the effect of history alone, we modify the BILSTM baseline system to include an LSTM over the lexical categories. This is similar to Vaswani et al. (2016), but our model differs in that we feed the base LSTM outputs directly into the top "language model" LSTM rather than into a further MLP layer that combines the two LSTMs. This keeps the changes from our PRIMDECODER model minimal.

	–History			+History		
	Acc	F_1	Cov	Acc	F_1	Cov
Cat	95.9	90.2	84.6	95.8	90.3	90.8
Prim	95.9	90.2	84.9	96.0	90.9	96.2

Table 2: Word accuracy, parser F_1, and parser coverage for the four model variants on the CCGbank test set.

Second, to examine the effect of outputting primitives alone, we alter our model to reset the decoder's LSTM state between words, and so cannot maintain a history between words. Other than these noted changes, these two additional variants are trained in the same way as above, with the same layer sizes, same hyperparameter optimization and training procedure, and same beam width.

Combined with PRIMDECODER and BILSTM, these alterations allow us to examine all four combinations of whether the model does or doesn't have history and whether it predicts whole categories or decodes primitive sequences.

Table 2 shows the results of the four options on the CCGbank test set. On the history axis, we note a result similar to Vaswani et al. (2016): adding history to the baseline model provides useful information about past prediction history, substantially boosting parser coverage. Vaswani et al. (2016) observed a slight word accuracy decrease when doing this without scheduled sampling; since we did not use scheduled sampling to keep the comparison well-controlled, we attribute the slight decrease in word accuracy for our version to this omission.

On the other axis, we find that very little changes when allowing the model to decode primitives instead of classifying categories *if prediction history is unavailable*. Word accuracy and F_1 stay about the same, but there is a slight increase in parser coverage. This indicates that there is no significant detriment to supertag prediction quality when predicting primitives over categories. Although we omit the value from Table 2, the memoryless primitive decoding model can also correctly tag some words with OOV categories, though not as well as PRIMDECODER: 2% word accuracy on the development set and 0.3% on the test set. Even with a similar word accuracy to the BILSTM baseline,

this model at least has the *ability* to produce new categories, an important property for a supertagger.

These results lead us to conclude that PRIMDECODER's outperformance of BILSTM is due to the *conjunction* of decoding primitives and allowing the decoder to keep a memory of previous predictions. Moreover, this improvement is synergistic: the increases in word accuracy, parser F_1, and coverage are substantially greater in magnitude than the sum of the increases from the two control models. We hypothesize that our model is better able to learn associations between categories given that it has direct access to the categories' primitive units.

7 Conclusion and future work

In this paper, we have presented an alternative view to classification-based CCG supertagging where lexical categories are constructed from CCG primitives. Where CCG categories are traditionally predicted atomically, we instead found that breaking them down into their primitive types and operators provides a substantial increase in word accuracy, parser F_1, and parser coverage for English CCG supertagging. Even with a simple linearization scheme, our LSTM decoder–based model outperformed the baseline in all respects, and was also able to generate correct categories during inference that were unseen during training. Our analysis showed that there is a strong interplay between knowledge of prediction history and prediction of primitive units, with both aspects being necessary to obtain the full increases in performance that our model exhibits. We conclude that our novel consideration of CCG lexical categories as the complex units that they are is worthwhile and beneficial.

Our model demonstrates the benefit of a more careful, informed consideration of the structure of supertagging and, by extension, CCG parsing. We expect that further, more sophisticated incorporation of category structure will yield additional benefit, and are investigating such extensions in place of the straightforward linearization of the category strings we applied in this paper; this is somewhat similar to some work in LTAG supertagging (Bangalore and Joshi, 1999; Kasai et al., 2017). At the same time, other categorial grammars, such as Lambek categorial grammar, are likely to be amenable to such improvements as well, but their theoretical properties may allow for more principled methods of decomposing lexical categories, allowing the supertagger's role to be more tightly integrated in the parsing process.

Acknowledgments

We thank Elizabeth Patitsas for her helpful discussions and comments, as well as our anonymous reviewers for their questions, suggestions, and encouragement. This research was enabled in part by support provided by NSERC, SHARCNET, and Compute Canada. Training of our models was aided by the use of GNU Parallel (Tange, 2011).

References

Jason Baldridge. 2008. Weakly Supervised Supertagging with Grammar-Informed Initialization. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, pages 57–64, Manchester, UK. COLING 2008 Organizing Committee.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc.

Yonatan Bisk and Julia Hockenmaier. 2012. Simple robust grammar induction with combinatory categorial grammars. In *AAAI Conference on Artificial Intelligence*, pages 1643–1649.

Yonatan Bisk and Julia Hockenmaier. 2013. An HDP Model for Inducing Combinatory Categorial Grammars. *Transactions of the Association for Computational Linguistics*, 1:75–88.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Philipp Koehn, and Tony Robinson. 2014. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. In *Proceedings of the 14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 2635–2639, Singapore. International Speech Communication Association.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-Supervised Sequence Modeling with Cross-View Training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. Association for Computational Linguistics.

Stephen Clark. 2002. Supertagging for Combinatory Categorial Grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 19–24, Universitá di Venezia. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2004. The Importance of Supertagging for Wide-Coverage CCG Parsing. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 282–288, Geneva, Switzerland. COLING.

Stephen Clark and James R. Curran. 2007. Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. *Computational Linguistics*, 33(4):493–552.

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.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc.

Dan Garrette, Chris Dyer, Jason Baldridge, and Noah A. Smith. 2014. Weakly-Supervised Bayesian Learning of a CCG Supertagger. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 141–150, Ann Arbor, Michigan. Association for Computational Linguistics.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Julia Hockenmaier and Mark Steedman. 2005. *CCGbank LDC2005T13*. Linguistic Data Consortium, Philadelphia, PA, 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.

Aravind K. Joshi and B. Srinivas. 1994. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*, pages 154–160.

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 the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1713–1723, Copenhagen, Denmark. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, San Diego, USA.

Konstantinos Kogkalidis, Michael Moortgat, and Tejaswini Deoskar. 2019. Constructive Type-Logical Supertagging With Self-Attention Networks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 113–123, Florence, Italy. Association for Computational Linguistics.

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. LSTM CCG Parsing. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 221–231, San Diego, USA. Association for Computational Linguistics.

Mike Lewis and Mark Steedman. 2014. Improved CCG Parsing with Semi-supervised Supertagging. *Transactions of the Association for Computational Linguistics*, 2:327–338.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, Haifa, Israel. Omnipress.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVE: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association*

for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany. Association for Computational Linguistics.

Laura Rimell and Stephen Clark. 2008. Adapting a Lexicalized-Grammar Parser to Contrasting Domains. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 475–484, Honolulu, Hawaii. Association for Computational Linguistics.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the 2nd International Conference on Learning Representations*, Banff, Canada.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Ole Tange. 2011. GNU parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47.

Ann Taylor, Mitchell Marcus, and Beatrice Santorini. 2003. The Penn Treebank: An Overview. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*, Text, Speech and Language Technology, pages 5–22. Springer Netherlands, Dordrecht.

Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394, Uppsala, Sweden. Association for Computational Linguistics.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging With LSTMs. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 232–237, San Diego, California. Association for Computational Linguistics.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems 28*, pages 2773–2781. Curran Associates, Inc.

Huijia Wu, Jiajun Zhang, and Chengqing Zong. 2017. A Dynamic Window Neural Network for CCG Supertagging. In *AAAI Conference on Artificial Intelligence*, pages 3337–3343.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG Supertagging with a Recurrent Neural Network. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 250–255, Beijing, China. Association for Computational Linguistics.

Wenduan Xu, Michael Auli, and Stephen Clark. 2016. Expected F-Measure Training for Shift-Reduce Parsing with Recurrent Neural Networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 210–220, San Diego, California. Association for Computational Linguistics.