Generating CCG Categories

Yufang Liu, Tao Ji, Yuanbin Wu, Man Lan
School of Computer Science and Technology, East China Normal University
{yfliu.antnlp, taoji.cs}@gmail.com, {ybwu, mlan}@cs.ecnu.edu.cn

Abstract

Previous CCG supertaggers usually predict categories using multi-class classification. Despite their simplicity, internal structures of categories are usually ignored. The rich semantics inside these structures may help us to better handle relations among categories and bring more robustness into existing supertaggers. In this work, we propose to generate categories rather than classify them: each category is decomposed into a sequence of smaller atomic tags, and the tagger aims to generate the correct sequence. We show that with this finer view on categories, annotations of different categories could be shared and interactions with sentence contexts could be enhanced. The proposed category generator is able to achieve state-of-the-art tagging (95.5% accuracy) and parsing (89.8% labeled F1) performances on the standard CCGBank. Furthermore, its performances on infrequent (even unseen) categories, out-of-domain texts and low resource language give promising results on introducing generation models to the general CCG analyses.

Introduction

Supertagging is the first step of parsing natural language with Combinatory Categorial Grammar (Steedman 2000) (Figure 1). The morpho-syntax enriched categories are of great interest not only because they can help to build hierarchical representations of sentences, but also because they provide a compact and concise way to encode syntactic functions behind words. As many other data-driven NLP models, applications of CCG analyses are constrained with the quality of annotations and the pre-defined category set. Here, for helping parsing texts in various domains, we aim to improve the robustness of current taggers and extend their abilities on discovering new unknown categories.

The primary model for CCG supertagging is sequence labeling: a classifier autoregressively predicts categories of sentence words. Internal structures of categories, however, are often ignored in this classification-based methods. As a key feature of CCG, these structures are actually quite informative for handling relations among categories. For example, the two categories in Figure 2 are different, but the functions they represent have the same combining strategy (first takes an argument from left, then from right) and identical argument types (two NPs). For building data-driven taggers, this fine-grained view on categories is able to expose more shared information and thus helps to build a more robust model. For instance, we may rely on internal structures to improve performance of infrequent categories by transferring knowledge from more frequent categories (which are learned more robustly). We can also use them to induce unknown categories by building new structures or filling new arguments, which is impossible for existing supertaggers.

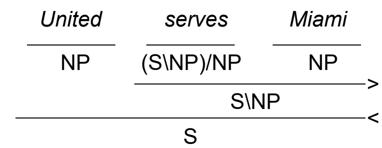

Figure 1: An example of CCG supertagging and parsing. “United” and “Miami” are noun phrases (NP). The transition verb “serves” has a category (supertag) “(S\NP)/NP” which means it first combines a right NP, then combines a left NP, and finally forms a sentence S.

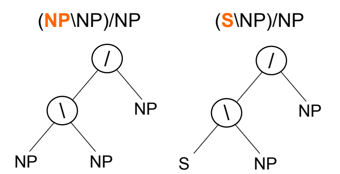

Figure 2: Two categories with similar internal structures. The left represents a transitive verb, and the right represents a preposition attached to a noun phrase.

Following Kogkalidis, Moortgat, and Deoskar (2019) work on fine-grained type-logical category generation, in this paper, we propose generation paradigms for CCG supertagging. Instead of viewing categories as simple class labels, we decompose them into smaller atomic tags. Predicting a category is now equal to generate the corresponding atomic tag sequence. For example, one decomposition of (NP\NP)/NP could be [(, NP, \, NP,), /, NP] which is identical to the same decomposition of category (S\NP)/NP except the first NP is replaced by S. Based on the tag sequences, the classifier can know more about the shared and the private learning signals of the two tags (e.g., by using new loss functions based on internal structures). It also provides a simple method to recognize new categories.

We introduce two types of category generator, the *tag-wise generator* (like NMT) which predicts atomic tags at each generation step and the *transition-based generator* (like parsing) which runs a transition system to get tag sequences in provable correct form. We also study a spectrum of atomic tag sets (from the smallest tokens to the original categories, from deterministic to non-deterministic) to illustrate the potential power of category generators: it is a flexible framework to study how categories are formed and applied. Comparing with vanilla sentence-level sequence to sequence generation (Kogkalidis, Moortgat, and Deoskar 2019; Bhargava and Penn 2020), the proposed generators consider hierarchical structures of categories (transition-based) and issue multiple decoders for sentence words (faster and suitable for capturing the property of 'localized' syntax structure).

Experiments on the CCGBank show that supertagging with generation can outperform a strong classification baseline. With various decoding oracles and a simple reranker, the tagger achieves the state-of-the-art supertagging accuracy (95.5%, without using additional external resources, 96.1% with BERT). Furthermore, on low frequency and unseen categories, the category generator is significantly better than the traditional category classifier. On out-of-domain texts (Wiki and biomedical texts) and an Italian dataset, the category generator can also perform more robustly.

Category Classifier

In CCG analyses, supertagging is known as *almost parsing* (Bangalore and Joshi 1999): most syntax ambiguities will be solved if correct categories (supertags) are assigned to each word in a text. Given a sentence $\mathbf{x} = x_1, x_2, ..., x_n$, a supertagger predicts a tag sequence $\mathbf{t} = t_1, t_2, ..., t_n$ where x_i is a word and t_i is x_i's category taking from a category set $\mathcal{T}$.

In this section, we introduce a typical category classification model (Lewis, Lee, and Zettlemoyer 2016). It basically first encodes sentence words into vectors, then performs a multi-class category classification on them. Each word x_i is mapped to a vector (also denoted by x_i) by concatenating a randomly initialized vector, a pre-trained word embedding, and a CNN-based character embedding. A two-layer Bi-LSTM on $\mathbf{x}$ is then applied to obtain hidden states $h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}]$, $\overrightarrow{h_i} = \text{LSTM}(x_i, \overrightarrow{h_{i-1}}, \overrightarrow{\theta})$, $\overleftarrow{h_i} = \text{LSTM}(x_i, \overleftarrow{h_{i-1}}, \overleftarrow{\theta})$. After a softmax operator, we obtain the probability of a category $p(t_i|\mathbf{x})$, and apply the loss function $\mathcal{L} = -\sum_i \log p(t_i|\mathbf{x})$.

In this vanilla setting of sequence labeling, the relation between two tags t' and t'' is discarded. As a consequence, annotations of t' give no suggestion on correctly tagging t''. For example, if t', t'' are not the true tag, they will suffer a same loss even one of them has more overlapping with the gold category (Figure 2). As we have discussed, internal structures of categories can make the fine-grained sharing of annotation possible. In the following section, we are going to incorporate them in the process of CCG supertagging.

Category Generator

To explore the inner structures of categories, we first decompose them into smaller *atomic tags*[1]. For example, a category (NP\NP)/NP (prepositions attached to noun phrases) can be seen as a sequence,

[(, NP, \, NP,), /, NP].

One advantage of such decomposition is that now the tag has a connection with a different category (S\NP)/NP. Specifically, a model can recognize that both of the categories require a left NP. Atomic tags make these connections explicit (rather than hidden in model parameters) and provide a way for including them in taggers.

Atomic tags can also make recognizing unknown category possible. For categories not shown in the pre-defined label set, the classification model can never predict them correctly. However, by decomposing into atomic tags, even if a category is not presented in the label set, it is highly possible that subsequences of the category have been seen in the training set, which enables the model to generate correct unknown categories.

Different from Kogkalidis, Moortgat, and Deoskar (2019), we propose to deploy decoders for each individual word instead of decoding a single sequence for all words. Our setting may have following advantages, first of all, it is less sensitive to error propagation among tags due to the decoupling of the decoding sequence. Second, tags can be parallelized in the same sentence . Third, the decoder can explicitly include knowledge of the current word which fits the idea of assigning tags to words.

Formally, we define $\mathcal{T}_a$ to be an atomic tag set of the original set $\mathcal{T}$ if for every category t in $\mathcal{T}$, it can be expressed with a sequence of atomic tags in $\mathcal{T}_a$,[2] $t = a^1, a^2, ..., a^m$ where $a^j \in \mathcal{T}_a$. For a sentence word x_i, our tagger's object changes to generate the correct sequence of atomic tags. We deploy two types of sequence decoders for the category generation task. The first type follows the tag-by-tag generation paradigm. It is simple and fast, but the generated sequences are not guaranteed to be well-formed. The second type runs a transition system. The validity of its output is guaranteed with the cost of additional computation steps and hard to batch.

Tag-wise Generator

The tag-wise generator starts an LSTM at every x_i. Let g_i^j be the j-th hidden state of the generator, $g_i^j = \text{LSTM}(g_i^{j-1}, d_i^j, \theta)$, where $d_i^j = [h_i; a_i^{j-1}]$, h_i is the hidden state vector of x_i from the encoder (which keeps the generator watching x_i at each step), and a_i^{j-1} represents the embedding of the output tag from the previous generation step. The probability of an atomic tag is defined as,

$$p(a_i^j = a|\mathbf{x}) = \text{Softmax}_{a \in \mathcal{T}_a}\ w_a^\intercal g_i^j. \quad (1)$$

[1]To avoid confusion, we always use *atomic tag* to refer to tokens in a (arbitrary) decomposition of original categories, and following the CCGBank's user manual (Hockenmaier and Steedman 2005), we use *atomic category* to refer categories without arguments (e.g., S, NP, N, PP, see the manual for a full definition), which is denoted by $\mathcal{A}$.

[2]An "EOS" is attached to every sequence as a stop sign.

axiom:	$\epsilon : 0$	
goal:	$\epsilon : t \quad (t \neq 0)$	
$\mathsf{gen}(a)$	$\dfrac{\sigma : t}{\sigma\|a : t+1}$	$\sigma = \epsilon$ or top(σ) is an operator
$\mathsf{op}(X)$	$\dfrac{\sigma\|s_0 : t}{\sigma\|s_0\|X : t+1}$	s_0 is not an operator
reduce	$\dfrac{\sigma\|s_1\|X\|s_0 : t}{\sigma\|X_{s_1,s_0} : t+1}$	
stop	$\dfrac{\sigma : t}{\epsilon : t+1}$	len(σ) = 1

Table 1: The transition system of generating categories.

The loss function is $\mathcal{L} = -\sum_i \sum_j \log p(a_i^j|\mathbf{x})$. Comparing with the loss of classification model, the loss here is computed on the finer atomic tags, which can assign credit for partially correct category predictions.

Furthermore, the generator is able to handle relations between sentence contexts and categories in a better way. For example, when generating the second NP in NP/NP, it might be helpful to know whether words on the left form a noun phrase. Due to the decomposition of categories, each atomic tag is able to search related information from the sentence. We apply attention layers to help such context-aware category generation. Specifically, at step j of the category generator on word i, we use hidden state g_i^{j-1} to query which sentence words are more important for predicting the next atomic tag, $\alpha_i^{j,l} = \mathrm{Softmax}_{l\in[1,n]}(w^\intercal \tanh(W_1 g_i^{j-1} + W_2 h_l))$, where w, W_1, W_2 are parameters. A soft aggregation of all encoder vectors h_l becomes a part of the generator's input

$$d_i^j = [h_i;\; a_i^{j-1};\; \textstyle\sum_{l=1}^n \alpha_i^{j,l} h_l]. \tag{2}$$

In order to reduce computation costs, we could also compute a single attention vector using h_i as query and apply it in every generation step, $\alpha_i^l = \mathrm{Softmax}_{l\in[1,n]}(w^\intercal \tanh(W_1 h_i + W_2 h_l))$

$$d_i^j = [h_i;\; a_i^{j-1};\; \textstyle\sum_{l=1}^n \alpha_i^l h_l]. \tag{3}$$

Transition-based Generator

We can also explicitly explore tree structures of categories during the generation. In fact, by seeing combination operators ("/", "\") as non-terminals, atomic categories as terminals, categories resemble (binarized) constituent trees. We can therefore adopt parsing algorithms to obtain a well-formed categories, which is generally not guaranteed in tag-wise generators. Here, we investigate an in-order transition system (Liu and Zhang 2017), which is a variant of the top-down system (Dyer et al. 2016).

Table 1 illustrates the deduction rules of the transition-based generator. Each transition state contains a stack σ and the current timestep t. $\mathsf{gen}(a)$ generates an atomic category $a \in \mathcal{A}$[1] and pushes a to the stack σ. $\mathsf{op}(X)$ generates a combination operator $X \in \{/, \backslash\}$ and push X to σ. reduce combines the top three elements of σ and concatenates them

T	Stack	Buffer	Action
0			gen(S)
1	S	S	op(\)
2	S|\	S	gen(NP)
3	S|\\|NP	S|NP	reduce
4	S\NP	S|NP	op(/)
5	S\NP|/	S|NP	gen(NP)
6	S\NP|/|NP	S|NP|NP	reduce
7	(S\NP)/NP	S|NP|NP	stop

Figure 3: An example of category (S\NP)/NP for transition-based generator.

to the output t. stop is the stopping rule. An example of transition is shown in Figure 3.

At each step of the generation, a classifier predicts which action to perform. Following Dyer et al. (2016), we use a stack-LSTM to encode stack states. The detailed configuration is in the supplementary due to the lack of space.

Discussions The transition-based generator produces categories with provably correct form, which is not guaranteed in the tag-wise generator. On the other side, the tag-wise generator is easier to batch and much faster. Empirically, we find that the problem of illegal categories is not severe in the tag-wise generation: all 1-best outputs of the generator are legal and only 0.05% of 4-best outputs are wrong. In fact, like recent practice of sequence-style parsing (Zhang, Cheng, and Lapata 2017; Fernández-González and Gómez-Rodríguez 2019; Shen et al. 2018), it is possible to drop structure constraints with a well-learned sequence decoder. Categories are usually short (average length is 4) and their number is also limited (10^3). All these factors increase the chance of obtaining well-formed categories directly from the tag-wise generator. We thus focus on this simpler implementation.

We also note that it's straightforward to apply advanced encoder structures(in fact, we apply BERT(Devlin et al. 2019) in our experiments). However, we would like to think the main contribution here is to study CCG Supertagging from a new perspective, rather than a new generation model.

Decoding Oracles

One key point in category generators is how to define the atomic tag set $\mathcal{T}_a$ which determines the learning targets (*oracles*) of the decoder. For the transition-based generator, $\mathcal{T}_a$ is simply the transition action set. In the following, we are going to show different settings of $\mathcal{T}_a$ for the tag-wise generator. Following the semantics of CCG, we have a natural choice of $\mathcal{T}_a$,

$$\mathcal{T}_a = \mathcal{A} \cup \{(,), \backslash, /\}, \tag{AC}$$

where $\mathcal{A}$ contains atomic categories[1] of the grammar.

It's easy to see that each category $t \in \mathcal{T}$ corresponds to a unique atomic tag sequence from **AC**, which forms a *deterministic oracle* for the category generator.

We can enrich **AC**, for example, with some parentheses expressions (e.g., "NP\NP" in category "(NP\NP)/NP") in the original category (which may help to handle some common local syntactic functions),

$$\mathcal{T}_a = \textbf{AC} \cup \mathcal{P}_k, \qquad \textbf{(PA)}$$

where $\mathcal{P} = \{\tau | (\tau)$ is a substring of a $t \in \mathcal{T}\}$, $\mathcal{P}_k$ is the subset of $\mathcal{P}$ with top-k frequent items.

Furthermore, we could also either completely ignore the semantics of categories by adding their n-grams or completely accept them by adding all items in $\mathcal{T}$,

$$\mathcal{T}_a = \textbf{AC} \cup \mathcal{N}_k^n, \qquad \textbf{(NG)}$$

$$\mathcal{T}_a = \textbf{AC} \cup \mathcal{T}, \qquad \textbf{(OR)}$$

where $\mathcal{N}^n = \{\tau | \tau$ is a n-gram of a $t \in \mathcal{T}\}$, $\mathcal{N}_k^n$ is the subset of $\mathcal{N}$ with top-k frequent n-grams.

Unlike **AC**, when $\mathcal{T}_a$ is set to **PA**, **NG** and **OR**, a category t may have more than one correct sequences. For example, with **PA**, the tag (NP/NP)\NP may have two gold standard atomic tag sequences, [(, NP/NP,), \, NP] and [(, NP, /, NP,), \, NP].

We can still pick a deterministic oracle by applying some heuristic rules. Here, the deterministic oracles always perform the longest forward matching (i.e., with a prefix $a^1, a^2, \ldots, a^j$, a^{j+1} is set to a feasible atomic tag with the longest length).[3] On the other hand, we also investigate *non-deterministic oracles* for training the tag-wise generator. Instead of using a fixed oracle during the entire training process, we select oracles randomly for each category, and all oracles will participate in the learning of the supertagger.

Re-ranker

To combine a category generator and the category classifier, we further introduce a simple re-ranker. First, using beam search, we can obtain k-best categories from the category generator. For each category $t = a^1, a^2, \cdots, a^m$, we assign it a confidence score using probabilities of tags (Equation 1), $u_t = \frac{1}{m^\nu} \sum_{j=1}^{m} \log p(a^j | \mathbf{x})$ where $\nu \leq 1$ is a hyperparameter using to penalize long tag sequences.

Next, we use the category classifier to obtain category t's probability $\log p(t|\mathbf{x})$ as its confidence score v_t. The final score of t is defined as the weighted sum of the two scores $\lambda u_t + (1 - \lambda) v_t$. The category with the highest score is taken as the final output. We set $\nu = 0.15, \lambda = 0.9$ by selecting them on the development data.

Experiments

Datasets and Criteria We conduct experiments mainly on CCGBank (Hockenmaier and Steedman 2007). We follow the standard splits of CCGBank using section 02-21 for training set, section 00 for development set, and section 23 for test set. There are 1285 different categories in training set, following the previous taggers, we only choose 425 of them which appear no less than 10 times in the training set, and assign UNK to the remaining tags.

For out-of-domain evaluation, we use the Wikipedia corpus (Clark et al. 2009) and the Bioinfer corpus (Rimell and Clark 2009).[4] We also test our models on the news corpus of the Italian CCGBank(Johan, Bosco, and Mazzei 2009), We use the token-POS-category tuples file from the Italian news corpus.[5]

The main criterion for evaluation is tag accuracy. To measure statistical significance, we employ t-test (Dror et al. 2018) with $p < 0.05$.[6] The settings of network hyperparameters are in the supplementary. We compare several models,

- CC, the category classifier in Section .
- CG, the tag-wise generator with deterministic oracle **AC**.
- CGNG2, the tag-wise generator with deterministic **NG** ($k = 10, n = 2$).
- CT, the transition-based system in Section .
- rerank, combining CG and CGNG2 with the ranker (beam size is 4).

Model	**Dev**	**Test**	**Size**	**Speed**
C&C	91.50	92.02	-	
Lewis, Lee, and Zettlemoyer (2016)	94.10	94.30	48.88	-
+tri-training	94.90	94.70	-	-
Vaswani et al. (2016)	94.24	94.50	-	-
Wu, Zhang, and Zong (2017a)	94.50	94.71	99.16	-
Wu, Zhang, and Zong (2017b)	94.72	95.08	189.37	-
CC	94.89	95.21	77.11	466
CG	95.10	95.28	79.94	199
CGNG2	95.26*	95.44*	80.02	199
CT	94.06	94.09	77.97	21
rerank	**95.27***	**95.48***	-	199
Pre-training				
Clark et al. (2018)	-	96.10	-	-
Bhargava and Penn (2020)	**96.27**	96.00	-	-
BERT+CC	96.01	95.93	78.62	231
BERT+CG	96.13	95.97	81.35	131
BERT+CGNG2	96.18	95.99	81.53	131
BERT+CT	95.28	94.91	79.48	17
BERT+rerank	96.24*	**96.05***	-	131

Table 2: Comparing with existing supertaggers. Model sizes are the number of parameters (MB). Speeds are in sentence per second. We use BERT-base without fine-tuning. All results of our models are averaged over 3 runs. * indicates significantly better.

Main Results

Table 2 lists overall performances on CCGBank. C&C is a non-neural-network-based CCG parser, (Lewis, Lee, and Zettlemoyer 2016) is a LSTM-based supertagger similar to our CC model (with less parameters). It also uses tri-training-based semi-supervised learning (Weiss et al. 2015). Shortcut LSTM (Wu, Zhang, and Zong 2017b) performs best in previous works, which uses the shortcut block as a basic architecture for constructing deep stacked models. Their final

[3]We assume there is only one tag with the longest length.

[4]They include 1000 Wikipedia sentences and 1000 biomedical (GENIA) sentences with noun compounds analysed.

[5]We use period to spilt the dataset and get 740 sentences as train/dev/test(8:1:1). Dataset can be download from `http://www.di.unito.it/~tutreeb/CCG-TUT/`.

[6]`https://github.com/rtmdrr/testSignificanceNLP`

	oracles	Dev	Test
CC	-	94.89	95.21
D	**AC**	95.10	95.28
	PA	95.12	95.31
	NG($n = 2$)	**95.26**	95.44
	NG($n = 3$)	95.13	95.37
	NG($n = 4$)	95.25	**95.48**
ND	**PA**	95.08	95.43
	OR	95.00	95.24
	NG($n = 2$)	95.07	95.39
	NG($n = 3$)	95.23	95.38
	NG($n = 4$)	95.20	95.35

Table 3: Results of tag-wise generators combined with various oracles. D and ND represent deterministic and non-deterministic oracles. For PA and NG, we set $k = 10$. Except AC and OR, the improvements are significant (with $p < 0.05$).

model uses 9-layer stacked shortcut block as encoder. And a contemporaneous work (Bhargava and Penn 2020) which use a single decoder for the whole sentence. From the results, we find that,

- Our implementation of the category classifier (CC) outperforms the best previous system (Shortcut LSTM) with much less parameters.
- With the same encoder and a small increase of model size, tag-wise generators could bring further performance gains (CG and CGNG2). However, our current transition-based generator underperforms the classification model. Regarding the implementation of transition systems, we adopt the standard stack-LSTM which doesn't fully explore the features of transition structures. It is possible that further feature engineering and advanced encoders will improve the performances. Finally, the reranker can reach a new state-of-the-art in supertaggers using no external data.
- Regarding tagging speeds, since tag-wise generators need additional decoding steps on sentence words, they speeds are roughly two-fifths of the classification model. The transition-based generator is much slower since it needs to build features from the stack, the current output and history actions using LSTMs at every decoding step.
- All of our models obtain an appreciable increase in performance with the help of BERT (Devlin et al. 2019). The results of NGCG2 (with rerank) are comparable to the results of cross view training (Clark et al. 2018) which uses unsupervised data and annotations from other tasks and the contemporaneous work (Bhargava and Penn 2020) which shares the same idea of generating categories.
- We also test our models on the Italian CCGBank, it shows there is no significant difference between the results of CC and CG models.And our CGNG2 model performs best(64.10%). It proves that our tag-wise generators can still perform well with few data. Detailed results are in the supplementary.

Next, we show performances of the tag-wise generator with different oracles (Table 3). In general, comparing with the category classifier, the sequence oracles could effectively boost tagging accuracies. The following are some observations.

Model	Acc
CG$^{\triangle}$	94.30
+ attention(Equation 3)	+0.63
+ attention(Equation 2)	+0.65
CC	94.89
w/o cnn	-0.19
w/o dropout	-1.49
CG	95.10
w/o cnn	-0.36
w/o dropout	-1.30
w/o tag embedding	-0.59

Table 4: Ablation studies. The last row means the model without atomic tag embedding in the last decoding step. $\triangle$ denotes a smaller model for running attention.

- It is interesting to see that n-gram oracles **NG** perform better (on Dev) than other oracles both on deterministic and non-deterministic settings. We guess that, besides existing atomic categories in **AC** (and their simple combinations in **PA**), which have clear definitions from linguistic prior, there still exist some other latent linguistic structures which might help CCG analyses. How to uncover them is our important future work.
- Except **NG** ($n = 3$), the non-deterministic oracle is not able to get better accuracies than deterministic oracles. One reason might be that simply using random learning targets may make the generator harder to learn, thus more advanced fusion strategies are desired.
- We have tested oracle **AC** and **NG** with larger k (i.e., including more items). The results are similar to those in Table 3, which may suggest that the oracles are not quite sensitive to items' frequencies when choosing properly.

Third, we show the effectiveness of attention layers. Constrained by our hardware platform, instead of using the default setting, we evaluate a smaller model (the batch size becomes 128, the dimensions of the encoder and the decoder LSTM are decreased to 300 and 200). The results (Table 4) show that, though attention layers require more computation resources, they can help to achieve significantly better tagging accuracies than the vanilla category generator. The two different attention settings (Equation 2, 3) performs nearly the same (thus we may prefer the faster one (Equation 3)).

Finally, we test the percentages of illegal categories generated from category generators, the results show that all 1-best outputs of CC and NGCG2 are legal and only 0.05%, 0.04% of 4-best outputs are wrong. It suggests that it is not hard for tag-wise generators to build well-formed categories given our moderate capacity decoding structures.

Robustness

By inspecting the CCGBank training set, we see that there are about two-thirds of categories which appear less than 10 times ($1 - 425/1285$), and more than half of the remaining

	10 ∼ 100	100 ∼ 400	400 ∼ 2000
CC	60.41	77.06	86.77
CT	41.86	63.81	79.73
CG	62.44	77.51	87.58*
CGNG2	**65.83***	78.95*	87.93*
rerank	64.25*	**79.84***	**88.16***
% in test	40.46%	17.70%	11.49%

Table 5: Accuracy of infrequent categories on the test set. We group categories with their frequency in the training set. The last row shows the proportion of categories in the test set. * indicates significantly better than model CC.

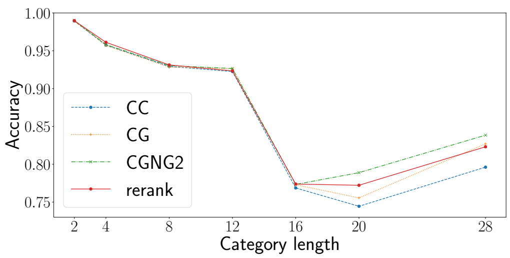

Figure 4: Accuracy on categories of different lengths.

categories appear less than 100 times (253/425). We now show how the category generator performs on these infrequent (or even unknown) categories, which can be a sign for model robustness.

First, from Table 5, category generators exhibit significantly better tagging accuracies on infrequent categories, and as the annotation becomes less, the gap between generators and the classifier becomes larger. Similarly, we compare systems with respect to the length of a category (empirically, a longer length implies a small frequency). Figure 4 shows that for categories with length less than 16, the models perform almost identical, but for longer categories, the category generator give more robust tagging results (especially for CGNG2 which has fewer generation steps than CG).

Next, we show performances of category generators on unknown categories. Recall that we only use 425 of 1285 categories in the training set (the remaining categories are tagged with `UNK`, and we still test all categories on test set)[7]. In order to avoid models considering `UNK` tag as a true tag, for `UNK` tags in the training set, we exclude their loss during the training (they are still fed into encoders in order to not break the input sentence). On the test set, there are 104 words with categories not included in the 425 training tags, and we show the results on these tags in Table 6. We can observe that, given the top-k candidates, the unseen tags can have a chance to be included, thus the generator might be a reasonable method to deal with unseen categories. We also find that CGNG2 now has lower performances comparing with CG. One reason

[7]We actually do experiments on models training on the whole tag set, the results are almost the same. Considering there are too few (specifically, 22) unseen categories which are not shown in the 1285 tag set to test performances. Thus we finally choose to train on the 425 categories.

	p@1	p@2	p@4	p@8
CG	11.54	17.31	20.19	25.00
w/o feature	20.19	32.69	35.58	43.27
CGNG2	8.65	14.42	15.38	22.12
w/o feature	21.15	29.81	31.73	39.42
CT	0.96	4.81	5.77	7.69
w/o feature	6.73	14.42	25.96	31.73

Table 6: The results on unknown categories. “p@k” measures whether the correct category appears in the top-k outputs of category generators. “w/o feature” means when comparing categories, we ignore their features (e.g., S[dcl] equals S).

Category	Prediction
S[wq]/N	S[wq]/(S[q]/NP)
(NP/NP)/N	NP[nb]/N
conj/PP	conj
(((S[pt]\NP)/PP)/PP)/NP	((S[pt]\NP)/PP)/NP
N/S[qem]	N/S[dcl]
S[wq]/S[dcl]	S[wq]/S[q]
(N\N)/(N\N)	conj

Table 7: Some examples of prediction on categories not in the training label set.

might be that when generating unseen categories, due to the lack of prior knowledge, the semantic of original atomic categories (established by linguists) are more important than the implicit (raw) information hidden in n-gram tags.

Some failed examples of generating unknown category are shown in Table 7. In the first and second lines, CG gives partially correct results. In the third and fourth line, an argument (PP) is missing. In the fifth and sixth lines, the prediction is mostly right except for wrong features of S (a declarative sentence is predicted as a yes-no question, since we have no special treatment on features of categories, it could be further improved). CG is completely wrong in the last row.

We also show overall performances when we reduce the size of the training set in Figure 5 (which may not increase the number of unknown tags, but provide an approximate setting). The generation model consistently outperforms the classification model with limited training data.

Then, we show the tagging results on out-of-domain data (Table 8) using models trained on the CCGBank. We find that CG performs significantly better than the baseline CC model. Therefore, the robustness of category generator can also extend to texts in different domains.

Parsing Results

To show the CCG parsing performances(Table 9), we feed outputs of our supertaggers into the C&C parser (Clark and Curran 2007). We compare our models with the C&C parser with a RNN supertagger (Xu, Auli, and Clark 2015), the A* parser with a feed-forward neural network supertagger (Lewis and Steedman 2014b), the A* parser with a LSTM supertagger (Lewis, Lee, and Zettlemoyer 2016), the A* parser with a language model enhanced biLSTM supertagger (Vaswani

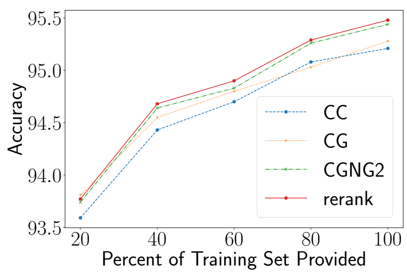

Figure 5: Tagging accuracy on test set with different training set sizes.

Method	Bioinfer	Wiki
CC	80.68	92.05
CG	80.68	92.24*
CGNG2	80.99*	92.34*
rerank	**81.05***	**92.42***

Table 8: Results on out-of-domain data sets. * denotes the difference between one model and the CC model is significant.

Model	F1	UF1	F1†	UF1†
C&C	85.45	91.65	-	-
Lewis and Steedman (2014a)	83.37	-	-	-
Xu, Auli, and Clark (2015)	87.04	-	-	-
Lewis, Lee, and Zettlemoyer (2016)	87.80	-	-	-
Vaswani et al. (2016)	88.32	-	-	-
Yoshikawa, Noji, and Matsumoto (2017)	88.80	94.00	-	-
CC	89.52	94.05	90.69	94.71
CG	89.68	94.14	90.77	94.76
CGNG2	89.76	94.22	90.82	94.79
CT	88.37	93.36	89.70	94.17
rerank	**89.80**	**94.22**	**90.87**	**94.83**
Gold pos tag				
Bhargava and Penn (2020)	90.2	-	90.90	-
rerank	**90.24**	94.52	**91.15**	95.01

Table 9: Parsing results on test set. † means using BERT. All results of our models are averaged over 3 runs.

et al. 2016), the A* CCG parser with a factorized biLSTM supertagger (Yoshikawa, Noji, and Matsumoto 2017), and C&C parser with a category generator for the whole sentence (Bhargava and Penn 2020).

In general, the parsing performances are consistent with supertagging results in Table 2: the rerank model achieves the best labeled and unlabeled parsing results. We also see that, even tag-wise generators may output illegal outputs, their parsing performances are better than the transition-based generator. An explanation, in addition to better supertagging results, is that CCG parsers are able to utilize k-best supertagger sequences (which further reduce the influence of one single illegal category) and ignore ill-formed categories easily (as the combination rules are always non-applicable to them).

Related Work

Traditionally, CCG supertagging is seen as a sequential labelling task. Clark and Curran (2007) propose C&C tagger which uses a log-linear model to build the supertagger. Recent works have applied neural networks to supertagging (Xu, Auli, and Clark 2015; Vaswani et al. 2016; Wu, Zhang, and Zong 2017b). These works perform a multi-class classification on pre-defined category sets and they can't capture the inside connections between categories because categories are independent of each other. Clark et al. (2018) propose Cross-View Training to learn the representations of sentences, which effectively leverages predictions on unlabeled data and achieves the best result. However, their model needs a large amount of unlabeled data. Vaswani et al. (2016) also want to model the interactions between supertags, but unlike our methods they use a language model to capture these connections. The difference is that we no longer treat every category as a label but a sequence of atomic tags.

The work closest to ours is Bhargava and Penn (2020). We share the same idea of generating categories but there are still some key differences. They decode a single sequence for all words while we deploy decoders for each individual words which may solve some problems(see Section). Besides, Prange, Schneider, and Srikumar (2020) also investigate the internal structure of CCG supertag. They treat each category as a single tree (just like our transition system) and use TreeRNNs for tree-structured category prediction.

Seq2Seq model has been used in many NLP tasks, such as machine translation (Sutskever, Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2015), text summarization (Nallapati et al. 2016; See, Liu, and Manning 2017), and especially on syntax parsing. More related, Vinyals et al. (2015) and Ma et al. (2017) use Seq2Seq model to generate constituency grammar, and Li et al. (2018), Zhang et al. (2017) use Seq2Seq model to generate dependency grammar. Inspired by their works, we apply Seq2Seq model to generate CCG supertags. But the difference is our generation is token level while theirs are sentence level. By splitting categories into smaller units, we decrease the size of label set. And the results show our category generating model performs well.

Techniques for classifying the unseen label have been investigated in many tasks, such as computer vision (Torralba, Murphy, and Freeman 2007; Bart and Ullman 2005; Lampert, Nickisch, and Harmeling 2014) and transfer learning (Yu and Aloimonos 2010; Rohrbach, Stark, and Schiele 2011). It would be an important future work to introduce advanced algorithms for dealing with these unknown categories.

Conclusion

We proposed a category generator to improve supertagging performance. It provides a new way to capture relations among different categories and recognizing unseen categories. We studied a Seq2Seq-based model, as well as a set of learning targets for the generator. Experiments on CCGBank, out-of-domain datasets and an Italian dataset show the effectiveness of our model. Future work will explore improving the accuracy of non-deterministic oracle and different rerankers. We will also study how to further improve tagging infrequent categories.

Acknowledgments

We would like to thank all reviewers for their helpful comments and suggestions. The corresponding author is Yuanbin Wu. This research is (partially) supported by NSFC (62076097), STCSM (18ZR1411500, 19511120200), and the Foundation of State Key Laboratory of Cognitive Intelligence, iFLYTEK(COGOS-20190003).

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Bangalore, S.; and Joshi, A. K. 1999. Supertagging: An approach to almost parsing. *Computational linguistics* 25(2): 237–265.

Bart, E.; and Ullman, S. 2005. Cross-Generalization: Learning Novel Classes from a Single Example by Feature Replacement. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, 672–679.

Bhargava, A.; and Penn, G. 2020. Supertagging with CCG primitives. In Gella, S.; Welbl, J.; Rei, M.; Petroni, F.; Lewis, P. S. H.; Strubell, E.; Seo, M. J.; and Hajishirzi, H., eds., *Proceedings of the 5th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2020, Online, July 9, 2020*, 194–204. Association for Computational Linguistics.

Clark, K.; Luong, M.; Manning, C. D.; and Le, Q. V. 2018. Semi-Supervised Sequence Modeling with Cross-View Training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 1914–1925.

Clark, S.; Copestake, A.; Curran, J. R.; Zhang, Y.; Herbelot, A.; Haggerty, J.; Ahn, B.-G.; Wyk, C. V.; Roesner, J.; Kummerfeld, J.; and Dawborn, T. 2009. Large-Scale Syntactic Processing : Parsing the Web Final Report of the 2009 JHU CLSP Workshop.

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

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Burstein, J.; Doran, C.; and Solorio, T., eds., *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 4171–4186. Association for Computational Linguistics.

Dror, R.; Baumer, G.; Shlomov, S.; and Reichart, R. 2018. The Hitchhiker's Guide to Testing Statistical Significance in Natural Language Processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, 1383–1392.

Dyer, C.; Kuncoro, A.; Ballesteros, M.; and Smith, N. A. 2016. Recurrent Neural Network Grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 199–209. San Diego, California: Association for Computational Linguistics.

Fernández-González, D.; and Gómez-Rodríguez, C. 2019. Left-to-Right Dependency Parsing with Pointer Networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 710–716. Minneapolis, Minnesota: Association for Computational Linguistics.

Hockenmaier, J.; and Steedman, M. 2005. CCGbank: Users manual. Technical Report MS-CIS-05-09, Department of Computer and Information Science University of Pennsylvania, Philadelphia.

Hockenmaier, J.; and Steedman, M. 2007. CCGbank: A Corpus of CCG Derivations and Dependency Structures Extracted from the Penn Treebank. *Computational Linguistics* 33(3): 355–396.

Johan, B.; Bosco, C.; and Mazzei, A. 2009. Converting a dependency treebank to a categorial grammar treebank for Italian. In *Eight international workshop on treebanks and linguistic theories (TLT8)*, 27–38. Educatt.

Kogkalidis, K.; Moortgat, M.; and Deoskar, T. 2019. Constructive Type-Logical Supertagging With Self-Attention Networks. In Augenstein, I.; Gella, S.; Ruder, S.; Kann, K.; Can, B.; Welbl, J.; Conneau, A.; Ren, X.; and Rei, M., eds., *Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019*, 113–123. Association for Computational Linguistics.

Lampert, C. H.; Nickisch, H.; and Harmeling, S. 2014. Attribute-Based Classification for Zero-Shot Visual Object Categorization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36(3): 453–465.

Lewis, M.; Lee, K.; and Zettlemoyer, L. 2016. LSTM CCG Parsing. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, 221–231.

Lewis, M.; and Steedman, M. 2014a. A* CCG Parsing with a Supertag-factored Model. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 990–1000.

Lewis, M.; and Steedman, M. 2014b. Improved CCG Parsing with Semi-supervised Supertagging. *TACL* 2: 327–338.

Li, Z.; Cai, J.; He, S.; and Zhao, H. 2018. Seq2seq Dependency Parsing. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, 3203–3214.

Liu, J.; and Zhang, Y. 2017. In-Order Transition-based Constituent Parsing. *Transactions of the Association for Computational Linguistics* 5: 413–424.

Ma, C.; Liu, L.; Tamura, A.; Zhao, T.; and Sumita, E. 2017. Deterministic Attention for Sequence-to-Sequence Constituent Parsing. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 3237–3243.

Nallapati, R.; Zhou, B.; dos Santos, C. N.; Gülçehre, Ç.; and Xiang, B. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, 280–290.

Prange, J.; Schneider, N.; and Srikumar, V. 2020. Supertagging the Long Tail with Tree-Structured Decoding of Complex Categories. *CoRR* abs/2012.01285.

Rimell, L.; and Clark, S. 2009. Porting a lexicalized-grammar parser to the biomedical domain. *Journal of Biomedical Informatics* 42(5): 852–865.

Rohrbach, M.; Stark, M.; and Schiele, B. 2011. Evaluating knowledge transfer and zero-shot learning in a large-scale setting. In *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, 1641–1648.

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 1073–1083.

Shen, Y.; Lin, Z.; Jacob, A. P.; Sordoni, A.; Courville, A.; and Bengio, Y. 2018. Straight to the Tree: Constituency Parsing with Neural Syntactic Distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1171–1180. Melbourne, Australia: Association for Computational Linguistics.

Steedman, M. 2000. *The syntactic process*, volume 24. MIT press Cambridge, MA.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 3104–3112.

Torralba, A.; Murphy, K. P.; and Freeman, W. T. 2007. Sharing Visual Features for Multiclass and Multiview Object Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 29(5): 854–869.

Vaswani, A.; Bisk, Y.; Sagae, K.; and Musa, R. 2016. Supertagging With LSTMs. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, 232–237.

Vinyals, O.; Kaiser, L.; Koo, T.; Petrov, S.; Sutskever, I.; and Hinton, G. E. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2773–2781.

Weiss, D.; Alberti, C.; Collins, M.; and Petrov, S. 2015. Structured Training for Neural Network Transition-Based Parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, 323–333.

Wu, H.; Zhang, J.; and Zong, C. 2017a. A Dynamic Window Neural Network for CCG Supertagging. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, 3337–3343.

Wu, H.; Zhang, J.; and Zong, C. 2017b. Shortcut Sequence Tagging. In *Natural Language Processing and Chinese Computing - 6th CCF International Conference, NLPCC 2017, Dalian, China, November 8-12, 2017, Proceedings*, 196–207.

Xu, W.; Auli, M.; and Clark, S. 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 of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, 250–255.

Yoshikawa, M.; Noji, H.; and Matsumoto, Y. 2017. A* CCG Parsing with a Supertag and Dependency Factored Model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, 277–287.

Yu, X.; and Aloimonos, Y. 2010. Attribute-Based Transfer Learning for Object Categorization with Zero/One Training Example. In *Computer Vision - ECCV 2010 - 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part V*, 127–140.

Zhang, X.; Cheng, J.; and Lapata, M. 2017. Dependency Parsing as Head Selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, 665–676. Valencia, Spain: Association for Computational Linguistics.

Zhang, Z.; Liu, S.; Li, M.; Zhou, M.; and Chen, E. 2017. Stack-based Multi-layer Attention for Transition-based Dependency Parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, 1677–1682.