Recurrent Neural Network CCG Parser

Sora Tagami
Ochanomizu University
tagami.sora@is.ocha.ac.jp

Daisuke Bekki
Ochanomizu University
bekki@is.ocha.ac.jp

Abstract

The two contrasting approaches are end-to-end neural NLI systems and linguistically-oriented NLI pipelines consisting of modules such as neural CCG parsers and theorem provers. The latter, however, faces the challenge of integrating the neural models used in the syntactic and semantic components. RNNGs are frameworks that can potentially fill this gap, but conventional RNNGs adopt CFG as the syntactic theory. To address this issue, we implemented RNN-CCG, a syntactic parser that replaces CFG with CCG. We then conducted experiments comparing RNN-CCG to RNNGs with-/without POS tags and evaluated their behavior as a first step towards building an NLI system based on RNN-CCG.

1 Introduction

Over the years, two contrasting approaches to natural language inference (NLI) have emerged: end-to-end neural NLI systems based on large language models (LLMs) (Lan et al., 2020; Raffel et al., 2020; He et al., 2021), which we call *mono-modular* approaches, and linguistically-oriented NLI pipelines consisting of syntactic parsers, semantic representations and theorem provers (Bos et al., 2004; Chatzikyriakidis and Luo, 2014; Mineshima et al., 2015; Abzianidze, 2015; Martínez-Gómez et al., 2017; Chatzikyriakidis and Bernardy, 2019), which we call *multi-modular approaches*. While the former has become more popular in recent years and has shown remarkable progress with the increasing scale of LLMs, the latter offers high precision, explanatory properties and strength in higher-order reasoning such as arithmetic. Both approaches have strengths and weaknesses and are expected to complement each other.

A drawback of using neural networks in multi-modular approaches is that their neural models are split between syntax and semantics. For example, the neural part-of-speech (POS) taggers cannot receive feedback from the results of the semantic component. The distributional representations in semantic components considered in works such as Cooper (2019); Larsson (2020); Bekki et al. (2022, 2023) are not connected to syntax. This gap between syntactic and semantic neural models is a potential weakness of multi-modular approaches compared to mono-modular approaches that seek to optimize the whole process of NLI.

The use of Recurrent Neural Network Grammars (RNNGs) (Dyer et al., 2016) is a potential solution to bridge the gap between syntactic and semantic neural models in multi-modular approaches. RNNGs provide syntactic parsers that can function as feeding input (syntactic structures) to semantic components, which is still a non-trivial task for large language models. Furthermore, unlike standard syntactic parsers and large language models, RNNGs provide embedded representations for *phrasal* constituents obtained by training on predicting syntactic structures, which we expect to be useful in a semantic component as well.

One remaining challenge is that the underlying grammar of the current RNNGs is context-free grammar (CFG), while modern syntactic processing in the multi-modular approaches adopts mildly context-sensitive grammars such as combinatory categorial grammar (CCG) (Steedman, 1996, 2000).

Therefore, in this study, we attempt to implement *RNN-CCG*, a syntactic parser that replaces the underlying grammar of RNNGs from CFG to CCG, and compare the performance of RNN-CCG with RNNGs, as a first step towards developing a complete NLI system using RNN-CCG. Technically, RNN-CCG can be built using almost the same techniques as RNNGs, but we will show that its performance is slightly better than RNNGs.

2 Recurrent Neural Network Grammars

RNNGs are language models and syntactic parsers that explicitly model hierarchical structures of words and phrases. Here, we will give an example of their behavior as syntactic parsers. Internally, RNNGs use two data structures: Stack and Buffer. Initially, Buffer contains all the word vectors. Operations on them are defined as Actions:

SHIFT Pop the word vector from Buffer and push it to Stack.

NT X Push a vector corresponding to the non-terminal symbol X to Stack. This non-terminal symbol X is marked as *open*.

REDUCE Pop from Stack all the elements up to the first *open* non-terminal symbol X encountered. Generate a new vector that encodes them and push it back to Stack as a new element.

At each time step, Stack, Buffer, and history of Actions are encoded using LSTMs and RNNs. Parsing is performed by determining the next Action at each parsing state based on this encoding. It is inefficient to recompute the encoding of Stack every time; thus, RNNGs adopts a mechanism called Stack LSTMs (Dyer et al., 2015) for optimization.

RNNGs have been the subject of subsequent researches: stack-only RNNGs (Kuncoro et al., 2017), which eliminate Buffer from the architecture and use only Stack, a Pytorch implementation model (Noji and Oseki, 2021) that enables parallel execution and learning of larger data, and a model that uses Transformer instead of RNNs (Sartran et al., 2022; Qian et al., 2021). However, in this paper, we focus on comparing CFG and CCG as underlying syntactic theories of RNNGs, adopting the simplest model presented in the original paper (Dyer et al., 2016) and conducting experiments focusing on the parsing aspect.

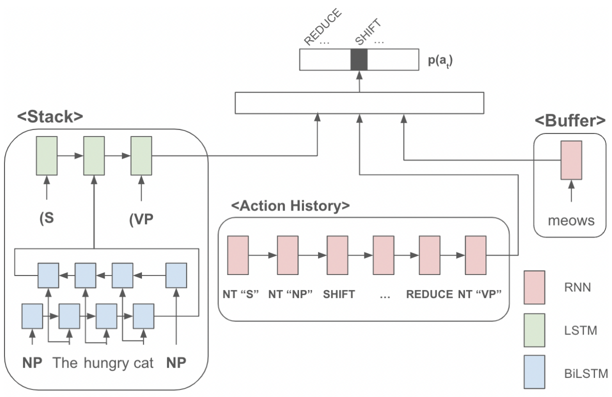

Figure 1: Architecture of RNNGs

3 RNN-CCG

We implemented two models based on RNNGs: RNN-CFG, which is a re-implementation of RNNGs using CFG, and RNN-CCG, which uses CCG instead of a CFG for the grammar used in RNNGs. By treating CCG syntactic categories as CFG terminal symbols, CCG is regarded as an instance of phrase-structure grammar, and the action selection is a multi-class classification task similar to the case of RNN-CFG. However, there was a problem with RNN-CCG in that its syntactic structures do not provide a layer for POS tags, which is insufficient to be used for semantic composition. Therefore, in this research, we extend RNN-CCG so that the structures have syntactic categories corresponding to words. For the sake of comparison, we also implement RNN-CFG that outputs non-terminal symbols corresponding to words.

3.1 Combinatory Categorial Grammar

CCG is a lexicalized grammar, the generative capacity of which is known to be mildly context-sensitive. In phrase structure grammars such as CFGs, most of the syntactic information is described by production rules, and the lexicon is relatively simple. In lexicalized grammars, on the other hand, most of the syntactic information is stored in the lexicon, and the combinatory rules are relatively simple. Additionally, CCG provides semantic information so that the syntactic structures determine the paths for semantic composition.

To generate *The hungry cat meows* in a context-free grammar, the following rules are required[1]:

$$\begin{array}{rcl} S & \rightarrow & NP\ VP \\ VP & \rightarrow & meows \\ NP & \rightarrow & The\ hungry\ cat \end{array}$$

In contrast, in CCG, each lexical item is defined as follows. In this example, two lexical items are combined using the backward function application rule, a combinatory rule in CCG, to generate a sentence.

$$\begin{array}{rcl} The\ hungry\ cat & \vdash & NP \\ meows & \vdash & S\backslash NP \end{array}$$

3.2 Part-of-speech Tags

RNNGs' syntactic structures do not contain POS tags; therefore, when implementing RNN-CCG

[1]The original paper on RNNGs does not consider the POS tags of each word; we follow this convention in this paper.

within the same framework as Dyer et al. (2016), a syntactic structure such as the one shown in Figure 2 is obtained from the output Action sequence. In Figure 2, it can be inferred that the syntactic category of *disclosed* is $S[pss]\backslash NP$, but to obtain the advantage of CCG syntactic structures, which is the path for semantic composition, it is necessary to supplement such syntactic categories of words and restore the detailed syntactic information. For example, it is unclear how to supplement the syntactic category of *were* or *'nt* in Figure 2. Therefore, in this study, as shown in Figure 3, we also make our RNN-CCG predict the category corresponding to each word using the "NT X" action.

```
(S[dcl]
 (S[dcl]
  (NP Terms)
   (S[dcl]\NP
    ((S[dcl]\NP) / (S[pss]\NP) were 'nt)
    disclosed ) )
 . )
```

Figure 2: Part-of-speech-insensitive parse tree

```
(S[dcl]
 (S[dcl]
  (NP
   (N Terms ) )
  (S[dcl]\NP
   ((S[dcl]\NP)/(S[pss]\NP)
    ((S[dcl]\NP)/(S[pss]\NP) were )
    ((S\NP)\(S\NP) n't ) )
   (S[pss]\NP disclosed ) ) )
 (. . ) )
```

Figure 3: Part-of-speech-sensitive parse trees

Commonly, several constraints are imposed in RNNGs (Dyer et al., 2016) to ensure the generation of well-formed constituent structures. In this study, we added POS tags and implemented the following constraints accordingly.

- SHIFT is immediately after "NT X"
- Always REDUCE immediately after SHIFT

These two rules mean that every single terminal symbol is reduced to a non-terminal. This non-terminal corresponds to the POS tag associated with the terminal.

4 Experiment

4.1 Experimental Setup

We implemented RNN-CFG and RNN-CCG for English using hasktorch[2], the Haskell interface for Torch. For training, we used Penn Treebank[3] as CFG data and CCGbank[4] as CCG data. We used sections 2-21 for training, section 24 for validation, and section 23 for evaluation in both corpora. Details are shown in Table 1.

Table 1: Corpus Statistics

	PTB		CCGbank	
	train	test	train	test
Sentences	39,832	2,416	39,604	2,407
Tokens	44,987	8,461	44,211	8,393
Actions(Without POS)	1,182	236	810	258
Actions(With POS)	1,229	282	1,642	542

4.2 Experimental Results

We show the micro F1 score for each model when this model is considered a sequence labeling model that predicts the actions in Table 2.[5]

Table 2: Experimental Results

	RNN-CFG		RNN-CCG	
	Without POS	With POS	Without POS	With POS
micro F1	90.7	91.3	91.3	93.6

Following the previous studies, these F1 scores are calculated for the predictions assuming that all the predictions before that time step coincide with the ground truth data.

4.3 Discussion

POS tags According to Table 2, the POS-tagged models achieved higher scores in both the CFG and CCG models. This is a welcome result given the usefulness of POS tags in semantic composition. On the other hand, this seems counter-intuitive since Table 1 shows that POS-tagged models have more actions than their untagged counterparts in both RNN-CFG and RNN-CCG. Predicting POS tags becomes more difficult when the number of classes increases in multi-class classification tasks.

This seemingly contradictory result can be attributed to the constraints discussed in Section 3.2. While the POS-free models predict a SHIFT action to move a word from Buffer to Stack, the POS-tagged models have to predict three actions in a row: "NT X", SHIFT, and REDUCE. Due to the constraints mentioned earlier, all three predictions

[2]http://hasktorch.org/

[3]https://catalog.ldc.upenn.edu/LDC99T42

[4]https://catalog.ldc.upenn.edu/LDC2005T13

[5]We used rt_G.large (NVIDIA V100 for NVLink 16GiB HBM2) on the ABCI (AI Bridging Cloud Infrastructure) (https://abci.ai/) as the experimental environment.

are guaranteed to be correct, contributing to the F1 score.

RNN-CCG vs. RNN-CFG Both the POS-tagged and POS-free RNN-CCG models outperformed their RNN-CFG counterparts in terms of microF1 score. Considering only the results of the POS-free models, attributing the differences in accuracy to the number of actions, the POS-tagged models in our study had more actions in the CCG case. Therefore, other factors must be at play. One possible explanation is that there are fewer combinatory rules in CCG grammar compared to CFG grammar. This results in a smaller pool of categories to predict with the "NT X" action, which may improve performance.

4.4 Error Analysis

In the above results, all predictions up to each time step used the ground-truth data, but when used as a parser, the prediction at each time step depends on the previous predictions. Therefore, we conducted an error analysis using the predicted results of the evaluation data by the syntax parser, including the state of Stack.

In RNN-CCG with POS, it often occurred that the same category was output repeatedly, as shown in Figure 4.

```
(S[dcl]\NP
   (S[dcl]\NP
       (S[dcl]\NP
           (S[dcl]\NP
               (S[dcl]\NP
                  ...
                  (S[dcl]\NP general )
```

Figure 4: Output of RNN-CCG with POS

This was not observed in RNN-CFG or RNN-CCG without POS. One possible cause is that many training data repeat predicting the same syntactic category during training. This is not the case in CFG, where there are not many production rules that predict the same nonterminal successively in the form of $X \to X, \ldots$. In CCG, however, this occurs when X and Y are the same in the backward function application rule. A typical example is $S\backslash NP \Rightarrow S\backslash NP, (S\backslash NP)\backslash(S\backslash NP)$, which occurs in a structure where an intransitive verb is followed by a VP modifier. While "NT $S[dcl]\backslash NP$" is continuously predicted, an intransitive verb continuously stays at the beginning of Buffer. So to learn when to transition to the SHIFT action, information about whether an adverbial phrase exists in the Buffer must be referred to.

There are also benefits to using CCG. In RNN-CFG, since the production rules are not defined in advance, there is no sense to ask which CFG rule is *correct*. Figures 5, 6, 7, and 8 are the predicted results for the same sentence by RNN-CFG and RNN-CCG, both of which are predicted incorrectly, but in Figure 6, there is no rule in CCG that has S[dcl] as the child and S[dcl] as the parent, so it is possible to judge whether the output tree is consistent according to CCG theory.

```
(NP
 (N
  (N/N
   ((N/N)/(N/N) 10)
   (N/N 1\/2 ) )
  (N
   (N % )
   (. . ) ) ) )
```

Figure 5: Correct

```
(S[dcl]
 (S[dcl]
  (NP
   (N
    (N/N 10 )
  (N
     (N/N 1\/2 )
     (N % ) ) ) )
  (. . ) ) )
```

Figure 6: Prediction by RNN-CCG

```
 (NP
   (QP
     (CD 10 )
     (CD 1\/2 ) )
   (NN % )
   (. . ) )
```

Figure 7: Correct

```
 (S
   (NP-SBJ
     (NP
       (NNP 10 )
       (NNP 1\/2 )
       (NNP % ) ) )
   (. . ) )
```

Figure 8: Prediction by RNN-CFG

5 Conclusion

In this study, we implemented RNN-CCG, a syntactic parser in which the grammar used inside the RNNGs was replaced from CFG to CCG, and conducted comparative experiments with RNN-CFG, a reimplementation of classical RNNGs. We also implemented their extensions with POS tags considering syntactic categories corresponding to words.

The results showed that the implementation of RNN-CCG achieved a higher F1 score than RNN-CFG with respect to the prediction of actions. Moreover, both models function effectively when considering POS tags, providing a better interface for semantic composition in the case of RNN-CCGs.

Overall, RNN-CCG is a prospective candidate of syntactic parsers in a modular NLI approach that bridges the gap between neural networks within CCG parsers and semantic modules. Future research could investigate the fusion of RNN-CCG with semantic composition and logical reasonings.

Acknowledgments

We thank the two anonymous reviewers of NALOMA'23 for their insightful comments. This work is partially supported by JST CREST Grant Number JPMJCR20D2, Japan, and JSPS KAKENHI Grant Number JP23H03452, Japan.

References

Lasha Abzianidze. 2015. Towards a wide-coverage tableau method for natural logic. In T. Murata, Koji Mineshima, and Daisuke Bekki, editors, *New Frontiers in Artificial Intelligence: JSAIisAI 2014 Workshops, LENLS, JURISIN, and GABA, Revised Selected Papers. Lecture Notes in Computer Science, volume 9067*, pages 66–82.

Daisuke Bekki, Ribeka Tanaka, and Yuta Takahashi. 2022. Learning knowledge with neural dts. In *the 3rd Natural Logic Meets Machine Learning (NALOMA III), pp.17-25, Galway, Ireland, Association of Computational Linguistics*, pages 17–25.

Daisuke Bekki, Ribeka Tanaka, and Yuta Takahashi. 2023. Integrating deep neural network with dependent type semantics. In Roussanka Loukanova, Peter LeFanu Lumsdaine, and Reinhard Muskens, editors, *Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021)*, Studies in Computational Intelligence 1081, page 261–284. Springer.

Johan Bos, Stephen Clark, Mark J. Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a ccg parser. In *COLING '04*.

Stergios Chatzikyriakidis and Jean-Philippe Bernardy. 2019. A wide-coverage symbolic natural language inference system. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics, NoDaLiDa 2019, Turku, Finland, September 30 - October 2, 2019*, pages 298–303. Linköping University Electronic Press.

Stergios Chatzikyriakidis and Zhaohui Luo. 2014. Natural language inference in coq. *J. Log. Lang. Inf.*, 23(4):441–480.

Robin Cooper. 2019. Representing types as neural events. *Journal of Logic, Language and Information*, 28:131–155.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. *CoRR*, abs/1505.08075.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 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*, pages 199–209.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DEBERTA: Decoding-Enhanced BERT with disentangled attention. In *International Conference of Learning Representations (ICLR2021)*.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A. Smith. 2017. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for self-supervised learning of language representations. In *International Conference of Learning Representations (ICLR2020)*.

Staffan Larsson. 2020. Discrete and probabilistic classifier-based semantics. In *the Probability and Meaning Conference (PaM 2020)*, pages 62–68. Association for Computational Linguistics.

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2017. On-demand injection of lexical knowledge for recognising textual entailment. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 710–720.

Koji Mineshima, Pascual Martínez-Gómez, Yusuke Miyao, and Daisuke Bekki. 2015. Higher-order logical inference with compositional semantics. In *Conference on Empirical Methods in Natural Language Processing (EMNLP2015)*, pages 2055–2061.

Hiroshi Noji and Yohei Oseki. 2021. Effective batching for recurrent neural network grammars. *CoRR*, abs/2105.14822.

Peng Qian, Tahira Naseem, Roger Levy, and Ramón Fernandez Astudillo. 2021. Structural guidance for transformer language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3735–3745.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

Laurent Sartran, Samuel Barrett, Adhiguna Kuncoro, Miloš Stanojević, Phil Blunsom, and Chris Dyer. 2022. Transformer grammars: Augmenting transformer language models with syntactic inductive biases at scale. *Transactions of the Association for Computational Linguistics*, 10:1423–1439.

Mark J. Steedman. 1996. *Surface Structure and Interpretation*. The MIT Press, Cambridge.

Mark J. Steedman. 2000. *The Syntactic Process (Language, Speech, and Communication)*. The MIT Press, Cambridge.