北京大学软件工程研究所
Software Engineering Institute, Peking University

TreeGen: A Tree-Based Transformer Architecture for Code Generation

Zeyu Sun[†] **Qihao Zhu**[†] **Yingfei Xiong**[*†] **Yican Sun**[†] **Lili Mou**[‡] **Lu Zhang**[†]
[†]Key Laboratory of High Confidence Software Technologies (Peking University), MoE;
Software Institute, Peking University, 100871, P. R. China
{szy_, zhuqh, xiongyf, sycpku, zhanglucs}@pku.edu.cn
[‡]University of Alberta, Edmonton, AB, Canada
doublepower.mou@gmail.com

Introduction

Given a specification written in natural language, a code generation system translates the specification into an executable program.

Open the file, F1 → f = open('F1', 'r')

In this task, the long-dependency problem, where a code element may depend on another far-away element, and the tree-structural information (Yin and Neubig 2017) are two main challenges. (Bengio, Simard, and Frasconi 1994, Sun et al. 2019).

We propose TreeGen based on Transformer (Vaswani et al. 2017). A Standard Transformer does not have layers for encoding structural information, and it is not clear where to add them. Thus, we design tree-based convolutional NNs for the first several Transformer decoder blocks to combine the vector representations of a node and its structural neighbors as the output of a structural convolution sub-layer.

Experiment

Our main experiment is based on an established benchmark dataset, HearthStone (HS), with two different preprocessing methods (Ling et al. 2016).

Model	StrAcc	Acc+	BLEU
LPN (Ling et al. 2016)	6.1	–	67.1
SEQ2TREE (Dong and Lapata 2016)	1.5	–	53.4
YN17 (Yin and Neubig 2017)	16.2	~18.2	75.8
ASN (Rabinovich, Stern, and Klein 2017)	18.2	–	77.6
ReCode (Hayati et al. 2018)	19.6	–	78.4
TreeGen-A	**25.8**	**25.8**	**79.3**
ASN+SUPATT (Rabinovich, Stern, and Klein 2017)	22.7	–	79.2
SZM19 (Sun et al. 2019)	27.3	30.3	79.6
TreeGen-B	**31.8**	**33.3**	80.8

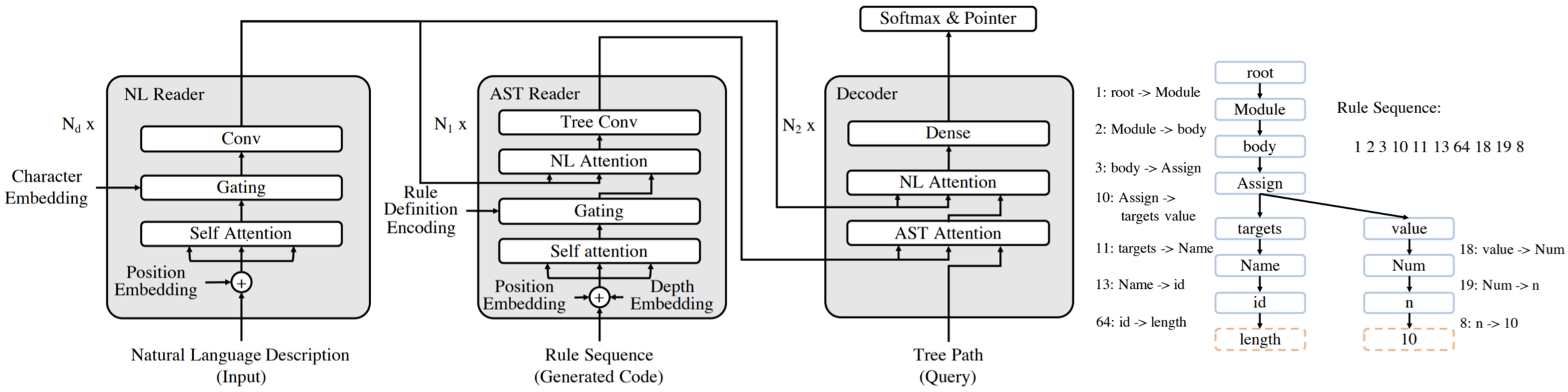

```
class DarkscaleHealer(MinionCard):
    def __init__ (self) :
        super().__init__("Darkscale Healer", 5,
            CHARACTER_CLASS.ALL, CARD_RARITY.COMMON,
            battlecry = Battlecry(Heal(2),
                CharacterSelector()))

    def create_minion (self, player):
        return Minion(4, 5)
```

We also achieved the best accuracy among NN-based approaches on two semantic parsing datasets, ATIS (89.1%) and GEO (89.6%).

Framework

For an input of program description, our task is to generate a piece of executable code.

We make use of the abstract syntax tree (AST) of a program and generate code by predicting the grammar rules and use the predicted rule to expand the non-terminal node in the AST.



As shown, TreeGen is divided into three parts: 1) an NL Reader; 2) an AST Reader; 3) a Decoder. Our core conjecture is that when convolving a node in AST and its structural neighbors, the vector representation should mainly contain the information from the original node. Thus, we add the structural convolution sub-layer (AST Reader) only to the first several Transformer decoder blocks but not all.

For the Input We apply an NL Reader over the input program description to capture the context information.

For the Code We apply an AST Reader for the predicted rules to encode structural information of the generated code via structural convolution sub-layers (Tree Conv) and alleviate the long dependence problem via self-attention.

For the Query The Query denotes position information of the next non-terminal node to be expanded. We represent it as a path from the root to such node and apply a Decoder to predict the rule.

References

[1] Bengio, Y.; Simard, P.; and Frasconi, P. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks* 5(2):157–166.
[2] Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In *ACL*, 33–43.
[3] Ling, W.; Blunsom, P.; Grefenstette, E.; Hermann, K. M.; Kočiský, T.; Wang, F.; and Senior, A. 2016. Latent predictor networks for code generation. In *ACL*, 599–609.
[4] Rabinovich, M.; Stern, M.; and Klein, D. 2017. Abstract syntax networks for code generation and semantic parsing. In *ACL*, 1139–1149.
[5] Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. In *ACL*, 440–450.
[6] Sun, Z.; Zhu, Q.; Mou, L.; Xiong, Y.; Li, G.; and Zhang, L. 2019. A grammar-based structural cnn decoder for code generation. In AAAI, volume 33, 7055– 7062.
[7] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In NIPS, 6000– 6010.
[8] Hayati, S. A.; Olivier, R.; Avvaru, P.; Yin, P.; Tomasic, A.; and Neubig, G. 2018. Retrieval-Based Neural Code Generation. In EMNLP, 925–930.