GCNGAN: Translating Natural Language to Programming Language based on GAN

Hongming Dai¹, Chen Chen², Yunjing Li¹, Yanghao Yuan¹

¹Department of Computer Science and Engineering, Northeastern University, Shenyang, China
²Department of Electronic Engineering, Tsinghua University, Beijing, China
20184542@stu.neu.edu.cn

Abstract: Cross-language translating has been well solved with the help of the processing of natural language processing (NLP). However, there are a few studies done about the domain of translating the natural language to programming snippets. Traditional method are mostly rule-based and limited to specific areas. In this paper, we propose a model based on Graph Convolutional Network (GCN)-Generative Adversarial Networks (GAN) to translate natural language to programming language. The generator is the encoder-decoder framework in which the encoder and the decoder are all bidirectional RNNs and GCN. And discriminator in GAN is also bidirectional RNNs and GCN. To improve the performance of semantic parsing, we also apply the attention mechanism to it. The experimental results indicate that our model has achieved comparable performance with some other state-of-the-art methods and has the stronger generalization ability.

1. INTRODUCTION

Semantic parsing is the task of converting a natural language utterance to a logical form: machine-understandable representation of its meaning. There are many applications about semantic parsing in text categorization, emotion detection, code generation and so on. In code generation field, few deep learning method has been proposed. Traditional method mainly focus on the expert system and their topics are only limited to specific area[5]. It relies on pattern matching to extract words and sentences from natural language utterances according to predefined semantic rules. Although these methods have achieved some extent of success, its generalization ability is very low because a predefined semantic rules can not be applied on the another field. What’s more, feature extraction by handcraft not only relies on a large amount of manual work but also impacts the performance of such semantic analysis systems which is relatively fragile. Because the existing rules and semantic templates cannot match the characteristics of natural language expression with ambiguity and expression diversity, the early system functionalities are relatively simple.

In this paper, we propose a method based on GCNGAN to accomplish this task. First, we apply generator to extract the semantic from the input natural language sequence and convert that to an Abstract Syntax Tree. The generator is an encoder-decoder framework. Encoder encodes natural language utterances into intermediate semantic vectors, and decoder decodes intermediate vectors into logical forms. Generally speaking, encoder and decoder can be any neural networks, GCN is used in this paper. Then, we input the Abstract Syntax Tree that the generator generates and the real Abstract Syntax Tree together to the discriminator to improve its discriminating ability. Generator and
discriminator play two-players game. Discriminator can differentiate between the two ASTs. During the process, the models’ performance can be improved greatly.

2. **MODEL**

2.1. **GAN-based Natural Language to Programming Language Model**

Generative Adversarial Network (GAN) is a framework for estimating generative models via an adversarial process, in which two models are simultaneously trained: a generative model $G$ that captures the data distribution, and a discriminative model $D$ that estimates the probability that a sample came from the training data rather than $G$[3]. The model’s architecture is shown in Fig.1.

In the model, the generator $G_{\theta}$ uses an encoder-decoder framework, which converts the natural language utterances into program codes, where $\theta$ represents parameters. The encoder encodes the natural language utterances as intermediate semantic vectors, and the decoder decodes the semantic vectors into fake ASTs with the guidance of the programming language grammar information.

The discriminator is responsible for judging whether the ASTs is real or not and gives instant feedback to generator by award motivation function. In other words, $D$ and $G$ play the following two-player minimax game with function $y(G, D)$[2]

$$\min_{G_{\theta}} \max_{D_{\phi}} y(\theta, \phi) = E_{x \sim p(x)} \log D_{\phi}(x) + E_{y \sim G_{\theta}} \log (1 - D_{\phi}(y))$$  \(1\)

Here,

$$G_{\theta} = p(y \mid x) = \prod_{q=1}^{\text{Y}} p(y_q \mid Y_{<q}, X)$$ \(2\)

Fig. 1: Model Architecture

It should be noticed that we apply Context Free Grammar (CFG) to guide the whole process which means AST is grammatically standardized[6].

2.2. **GCN-based GAN Generator**

In generator, we apply encoder-decoder framework to reduce the interaction between different languages. Encoder-Decoder model provides sufficient flexibility for the whole model because its ends can be many deep learning models. Usually, Recurrent Neural Network (RNN) and Long Short-Term
Memory (LSTM) are favored by most researchers about the time series forecasting model like natural language and programming language. RNN is famous for their capability to solve the vanishing gradient and to connect different information with a large time span [8]. Because of that, RNN have extremely successful applications in text classification, machine translation, emotion detection and other NLP fields. We propose Bidirectional RNNs to encode the natural language first in the encoder. Bidirectional RNNs use two RNNs: one runs in the forward direction and another one runs in the backward direction. Consider the encoder takes X as input sentence. The forward one RNN (x(1:t)) computes a representation of the left context and the RNN (x(t:n)) computes the representation of the right context.

To improve the generator’s semantic parsing ability, we propose the strategy gradient used in SeqGAN.

$$\Psi(\theta) = E_{Y \sim G_{\theta}} \log \left( \prod_{t=2}^{T} G_{\theta}(y_t | Y_{1:t-1}) \right) K(Y_{1:t})$$

$$K(Y_{1:t}) = D(A, Y_{1:t})$$

where K(Y 1 : t) represents the generator award function, which quantifies the quality of the generated programming language sequences.

![Fig. 2: Generator](image1)

To better improving our model’s capability of representing the most essential semantic of a natural language utterance. After one sentence is encoded by Bidirectional RNNs, it will be converted to graphs by GCN. The detailed process is as follows:

![Fig. 3: BiRNN-GCN-GCN Encoder](image2)
In the figure above, we consider PropBank-style semantic-role structures[4]. Here, the predicate ‘call’ has three arguments: Program(semantic role A0, the action applicator), function(A1, the thing that Program calls), array(A2, the object operated on by the function). In our model, GCN will capture commonalities between different realizations of the same predicate-argument structures. That can improve the performance of the encoding by BiRNNs greatly because argument switching is one of the frequent and severe mistakes made by neural machine translation (NMT). The detailed process can be described by the figure 3 above.

2.3. Discriminator: Converting AST to Semantics

Discriminator’s job is to judge whether the AST that generator generates is real or not by comparing that with the real AST. It has been mentioned above that the generator form the AST of programming language based on the natural language sequence. What discriminator should do is some what different. In generator, we propose encoder-decoder framework and we combine GCN with bidirectional RNNs to get the better result of semantic parsing. In discriminator, we only propose bidirectional BiRNNs to convert AST to semantics in a bottom-up way, which means convert it from leaf nodes to root nodes.

Let $M_{\text{AST}}$ be the final encoding vector of the AST that generator generates, and $M_{\nu}$ be the semantic vector of the natural language sequence. So the possibility $P$ that the semantic of AST is consistent with the natural language is as follows:

$$S = M_{\text{AST}} W^D M_{\nu} + \beta$$

$$P = \text{softmax} \left( \frac{e^{\text{consistent}}}{\sum_{e^i \in S} e^i} \right)$$

According the similarity of two semantics, discriminator will give instant feedback to generator based on Award Function. So that the generator’s generating ability can become better and better.

3. EXPERIMENTS

3.1. Datasets

- Python is an object-oriented programming language known for its simplicity and convenience. We crawl 20000 training datasets and 5000 verification datasets from Github and we manually labeled every single line of programming code with the corresponding natural language description text.
- Jobs is a job query language, by which the user can query for some jobs with the natural language utterance. We gather 400 training data sets and 150 test datasets in this paper.
- The Code Natural Language Challenge (CoNaLa) is the dataset created by Carnegie Mellon University for generating program snippets from natural language. We have downloaded the corpus from the official website, which has 2379 training and 500 test examples.

3.2. Experiment Results

Some relevant experiment results are demonstrated by the tables below.

| Datasets’ Names | Normal GAN | GCNGAN | GCNGAN with Pretraining |
|----------------|------------|--------|-------------------------|
| Python         | 67.1       | 68.6   | 69.9                    |

TABLE I: Various Models’ Accuracy
Table 1 compare the various model’s accuracy on various datasets. The pretrained model demonstrates the best performance, which explains the significance of pretraining to NLP task and the high efficiency of our model. On Python datasets and Jobs datasets, the GCNGAN improves 0.2% and 1.9% over the normal GAN. That is mainly because Python and Jobs has relatively good syntax information and GCNGAN’s game training method is also good at decoding the difficult logic of the program. However, on CoNaLa dataset, our model don’t show better performance over the normal GAN models. That may be due to the fact that the CoNaLa dataset is logically difficult and its grammar information is relatively simple which can not provide mode details to GCN when constructing AST.

Table 2 compares some state-of-the-art models with our model. In the Python and Jobs datasets, GCNGAN has achieved the best result of the other models and its advantage is relatively obvious. GCNGAN’s accuracy on Python improves around 25% over SEQ2TREE in Python and around 1% in Jobs. However, as what we analyze above, in the dataset like CoNaLa, its performance is poor due to CoNaLa’s characteristics.

### 4. CONCLUSION AND FUTURE WORK

As is demonstrated by our experiment, GCNGAN has achieved excellent performance in the task of converting natural language to programming language. It’s another successful application of these two hot networks. However, this paper mainly focus on the line-to-line conversion. In the real industrial production environment, programming language are dependent on multiple lines. In the future, we will consider designing a special method and data structure to solve the multi-line dependency problem.

### REFERENCES

[1] Li Dong and Mirella Lapata. “Language to Logical Form with Neural Attention”. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers. The Association for Computer Linguistics, 2016. DOI: 10.18653/v1/p16-1004. URL: https://doi.org/10.18653/v1/p16-1004.

[2] Ian J. Goodfellow et al. “Generative Adversarial Networks”. In: CoRR abs/1406.2661 (2014). arXiv: 1406.2661. URL: http://arxiv.org/abs/1406.2661.

[3] Ashish Khetan and Sewoong Oh. “Achieving budget-optimality with adaptive schemes in crowd- sourcing”. In: Advances in Neural Information Processing Systems. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016.

[4] Diego Marcheggiani, Jasminn Bastings, and Ivan Titov. “Exploiting Semantics in Neural Machine Translation with Graph Convolutional Networks”. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers). New Orleans, Louisiana:
Association for Computational Linguistics, June 2018, pp. 486–492. DOI: 10.18653/v1/N18-2078. URL: https://www.aclweb.org/anthology/N18-2078.

[5] Walid Mohamed Taha, ed. Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings. Vol. 5658. Lecture Notes in Computer Science. Springer, 2009. ISBN: 978-3-642-03033-8. DOI: 10.1007/978-3-642-03034-5. URL: https://doi.org/10.1007/978-3-642-03034-5.

[6] Yeo et al. “Abstract Syntax Tree in Extended Context Free Grammar”. In: Journal of Korean Information Science Society 14.2 (1987), pp. 589–592.

[7] Yi Yu and Simon Canales. “Conditional LSTM-GAN for Melody Generation from Lyrics”. In: CoRR abs/1908.05551 (2019). arXiv: 1908.05551. URL: http://arxiv.org/abs/1908.05551.

[8] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent Neural Network Regularization”. In: Éprint Arxiv (2014).