Retrieve and Refine: Exemplar-based Neural Comment Generation

Bolin Wei

Key Laboratory of High Confidence Software Technologies (Peking University)
Ministry of Education, China; Software Institute, Peking University, China
bolin.wbl@gmail.com

Abstract—Code comment generation is a crucial task in the field of automatic software development. Most previous neural comment generation systems used an encoder-decoder neural network and encoded only information from source code as input. Software reuse is common in software development. However, this feature has not been introduced to existing systems. Inspired by the traditional IR-based approaches, we propose to use the existing comments of similar source code as exemplars to guide the comment generation process. Based on an open source search engine, we first retrieve a similar code and treat its comment as an exemplar. Then we applied a seq2seq neural network to conduct an exemplar-based comment generation. We evaluate our approach on a large-scale Java corpus, and experimental results demonstrate that our model significantly outperforms the state-of-the-art methods.

Index Terms—comment generation, program comprehension, deep learning

I. INTRODUCTION

Code comments provide a clear description for a piece of the source code, which can help programmers understand programs quickly and correctly [1] and are beneficial for software maintenance [2]. However, generating proper comments is time-consuming. Hence, carrying out automatic comment generation becomes greatly crucial for software development.

Recently, many researchers have applied encoder-decoder neural networks and achieved state-of-the-art performance on the comment generation [3], [4]. It's worth noting that their methods only take the source code information as input. However, we find that these models tend to generate short comments and produce degenerate outputs in a state of "out of control". For example, on our Java dataset, about 2% of predictions by DeepCom [3] are less than four words, and there are over seventy HAD's [4] predictions repeating a token for twenty times. Therefore, we argue that it is not enough to generate comments only based on the source code.

Software reuse is common in software development [5], [6], that is, for a given input code, we can search for a similar piece of code in a large-scale software library. If source code in this library contains comments, then we can use these comments to assist in the comment generation for the given code. The traditional IR-based comment generation are based on this idea [7]–[10] and treat the retrieved comments directly as comments for a given code. However, the retrieved comment may contain some information that is inconsistent with the content in the given code. A better way is to treat comments of

```
Input code:   public static float abs ( float fvalue ){
                  if ( fvalue < 0 ){
                      return - fvalue ;
                  }
                  return fvalue ;
              }
Exemplar:    returns sine of a value .
Comment:    returns absolute value of a float .
```

Fig. 1. A code-comment pair from our dataset, along with its exemplar.

similar code as an exemplar to guide the comment generation. Figure 1 shows a simple code-comment pair together with its exemplar from our dataset.

Exemplar-based comment generation can be seen as a template-based comment generation process. However, defining a template is a time-consuming task and requires some domain knowledge. By using comments of similar code as exemplars, on the one hand, the workload of constructing templates is reduced, and on the other hand, the process of comment generation is controlled.

In this paper, we present an exemplar-based comment generation model, which consists of two modules, Retrieve and Refine. In Retrieve, we exploit an open source search engine to retrieve the similar code. In Refine, we use LSTMs [11] to encode the given code, the similar code and the comment of the similar code into feature vectors. Based on the correlation of the given code and similar code, a new comment is generated concerning the comment of the similar code.

II. BACKGROUND AND RELATED WORK

The automatic comment generation is a popular research task in the field of software engineering. Researchers have proposed various approaches to solve this task, including manual templates, IR, and deep learning-based methods. Template-based approaches [1], [12]–[14] generate comments based on manually defined rules and information from the source code. IR-based methods [8]–[10], [15] include retrieving terms using text retrieval techniques or retrieving similar code using code clone detection techniques, then combining terms into comments or treating comments of similar code directly as comments. Deep learning-based approaches [3], [4], [16], [17] are data-driven approaches that rely on large-scale code-comment pairs. The existing methods mainly utilize an encoder-decoder neural network, the encoder models the source code sequence or the AST of source code, and the decoder generates comments. However, the existing deep learning-based models do
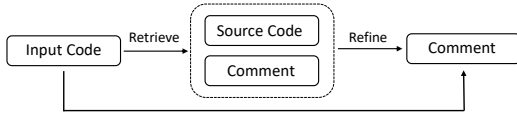
Fig. 2. An overview of our approach for exemplar-based comment generation. We use a dashed box to represent the pair of similar code and comment.

not take advantage of the feature of software reuse and lack control when generating comments, making it easy to produce short and meaningless comments.

III. APPROACH OVERVIEW

Our approach is shown in Figure 2, which consists of two parts, a Retrieve module for searching for similar code and a Refine module for comment generation.

A. Retrieve Module

The purpose of the Retrieve module is to find an appropriate exemplar in the training set. We assume that comments of similar code have similar patterns. Thus the process of finding a proper exemplar can be transformed into the process of finding a similar code. Specifically, given a piece of the input code, we search for code similar to it and treat a comment of the similar code as an exemplar. In this study, we leverage an open source information retrieval engine Lucene[1] to build our code library and keep the default settings[2] in Lucene. For each piece of code, we chose the first-ranked similar code and used its comment as an exemplar.

B. Refine Module

Once we have an exemplar, a straightforward way is to use it as a comment for the input code. However, the code usually contains information that is not in the input code, such as different variables. Therefore, a neural network can be applied to learn the difference between the input code and the similar code to improve the exemplar.

Specifically, the Refine module utilizes a seq2seq neural network [18]. The encoder uses three LSTMs to encode the input code, the similar code, and the comment of the similar code and explores the difference between the source code and the similar code using a nonlinear *sigmoid* function to obtain a similarity score s. The decoder, another LSTM with the attention mechanism [19], generates a new comment whose initial hidden state is the combination of the last hidden states of the input code $\mathbf{h}^x$ and the comment of the similar code $\mathbf{h}^y$:

$$\mathbf{h}^x * (1 - s) + \mathbf{h}^y * s$$

The purpose of this combination is that if the input code is very different from the similar code, that is, the similarity score is lower, then the decoder should pay more attention to the content of the input code. For each output token, the attention mechanism calculates its similarity to all input code

[1] https://lucene.apache.org/
[2] TextField with WhitespaceAnalyzer

Methods	BLEU (%)	METEOR (%)
Vanilla seq2seq	30.90	18.85
DeepCom	32.30	20.74
HAD	33.24	21.27
Our model	**43.78**	**28.12**

tokens and its similarity to all comment tokens of the similar code and obtains two context vectors. Then we use s to sum the two context vectors and get the final context vector.

During inference, we use a beam search [18] to generate comments. The training objective is the cross-entropy between the prediction and the ground truth.

IV. EXPERIMENTS

We collected a large number of Java projects from GitHub, extracted Java methods from them, and used the first sentence of Javadoc as a comment for the method. We processed the dataset as in [3]. After eliminating the duplicate data samples, we can obtain about 370k ⟨method, comment⟩ pairs. The training, validation, and test sets are about 330k, 20k, and 20k, respectively. We evaluate the performance based on BLEU [20] and METEOR [21], which are used in [3], [4].

We compared our approach to three neural network-based baseline methods, including vanilla seq2seq [18], Deep-Com [3], and HAD [4]. The last two methods are state-of-the-art methods on comment generation. Table I illustrates the BLEU and METEOR scores of different methods. As can be seen from the results, our approach is significantly better than all baseline methods on both metrics. The BLEU score of our model improves about 10% compared to HAD. Without similar code and comments, our model is equivalent to the vanilla seq2seq; hence, it is convincing that exemplars are very helpful for generating comments.

V. CONCLUSION AND FUTURE WORK

This paper creatively proposes a method for automatically generating comments based on exemplars, which includes two modules. The Retrieve module finds out a comment of a similar code as an exemplar. The Refine module exploits the correlation of the input code and the similar code to generate comments under the guidance of the exemplar. Experimental results show that our method can significantly outperform all previous state-of-the-art methods on the Java dataset.

This work has some limitations. On the one hand, our Retrieve module leverages an open source search engine. While ensuring efficiency, the token-based code search technique is too simple to retrieve the semantically similar code. In the future, we will use advanced code clone detection to improve the performance of the Retrieve module. On the other hand, the evaluation of the prediction is now based on metrics in machine translation. We expect to use human evaluation approaches to measure whether our methods can be applied in the software development process.

References

[1] G. Sridhara, E. Hill, D. Muppaneni, L. L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *ASE*.   ACM, 2010, pp. 43–52.

[2] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *SIGDOC*.   ACM, 2005, pp. 68–75.

[3] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *ICPC*.   ACM, 2018, pp. 200–210.

[4] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu, "Improving automatic source code summarization via deep reinforcement learning," in *ASE*.   ACM, 2018, pp. 397–407.

[5] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang, "Mining succinct and high-coverage API usage patterns from source code," in *MSR*.   IEEE Computer Society, 2013, pp. 319–328.

[6] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, "MAPO: mining and recommending API usage patterns," in *ECOOP*, ser. Lecture Notes in Computer Science, vol. 5653.   Springer, 2009, pp. 318–343.

[7] S. Haiduc, J. Aponte, and A. Marcus, "Supporting program comprehension with source code summarization," in *ICSE (2)*.   ACM, 2010, pp. 223–226.

[8] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *WCRE*.   IEEE Computer Society, 2010, pp. 35–44.

[9] E. Wong, T. Liu, and L. Tan, "Clocom: Mining existing source code for automatic comment generation," in *SANER*.   IEEE Computer Society, 2015, pp. 380–389.

[10] E. Wong, J. Yang, and L. Tan, "Autocomment: Mining question and answer sites for automatic comment generation," in *ASE*.   IEEE, 2013, pp. 562–567.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *ICPC*.   IEEE Computer Society, 2013, pp. 23–32.

[13] G. Sridhara, L. L. Pollock, and K. Vijay-Shanker, "Automatically detecting and describing high level actions within methods," in *ICSE*.   ACM, 2011, pp. 101–110.

[14] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *ICPC*.   ACM, 2014, pp. 279–290.

[15] B. P. Eddy, J. A. Robinson, N. A. Kraft, and J. C. Carver, "Evaluating source code summarization techniques: Replication and expansion," in *ICPC*.   IEEE Computer Society, 2013, pp. 13–22.

[16] M. Allamanis, H. Peng, and C. A. Sutton, "A convolutional attention network for extreme summarization of source code," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48.   JMLR.org, 2016, pp. 2091–2100.

[17] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, "Summarizing source code using a neural attention model," in *ACL (1)*.   The Association for Computer Linguistics, 2016.

[18] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.

[19] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[20] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *ACL*.   ACL, 2002, pp. 311–318.

[21] S. Banerjee and A. Lavie, "METEOR: an automatic metric for MT evaluation with improved correlation with human judgments," in *IEEvaluation@ACL*.   Association for Computational Linguistics, 2005, pp. 65–72.