Boosting Neural Commit Message Generation with Code Semantic Analysis

Shuyao Jiang
School of Computer Science, Fudan University, Shanghai, China
Shanghai Institute of Intelligent Electronics & Systems, Shanghai, China

***Abstract*—It has been long suggested that commit messages can greatly facilitate code comprehension. However, developers may not write good commit messages in practice. Neural machine translation (NMT) has been suggested to automatically generate commit messages. Despite the efforts in improving NMT algorithms, the quality of the generated commit messages is not yet satisfactory. This paper, instead of improving NMT algorithms, suggests that proper preprocessing of code changes into concise inputs is quite critical to train NMT. We approach it with semantic analysis of code changes. We collect a real-world dataset with 50k+ commits of popular Java projects, and verify our idea with comprehensive experiments. The results show that preprocessing inputs with code semantic analysis can improve NMT significantly. This work sheds light to how to apply existing DNNs designed by the machine learning community, *e.g.*, NMT models, to complete software engineering tasks.**

I. Introduction

Commit messages, usually short descriptions on the contents and reasons of code changes, are critical to code comprehension, and in turn, software development process [1]. However, in current practice, it is quite common that developers may write poor-quality commit messages [2], [3]. Automatic commit message generation has long been suggested as a viable means to solve this problem [4], [5]. Recently, with the advancement of deep learning, extensive research efforts have been put on applying deep neural networks (DNNs) in this task [6], [7]. The underlying idea is that human experiences are good resource to commit message generation. Specifically, good, manually-written commit messages in existing projects can help train a DNN which learns how to map code changes to their corresponding commit messages. This mapping is called neural machine translation (NMT), like that applied in natural language processing [8]. Thus, given new changes, suitable commit messages can be produced via NMT [6], [7].

Unfortunately, despite the capability of summarizing human experiences, the state-of-the-art NMT techniques are not yet able to produce satisfactory commit messages. Recent research show that it can only achieve a BLEU-4 score of 14.19 in a cleaned dataset [9], where BLEU-4 is a value typically used to evaluate the quality of text generation [10]. How to best exploit good, manually-written commit messages in automatic commit message generation remains unclear. Unlike recent efforts that attempt to improve the DNNs in NMT [11], [12], we analyze this problem in a different perspective: We suggest that the data we typically use to train the DNNs is inappropriate.

The key of NMT is a sequence-to-sequence mapping, basically designed for natural language translation. Hence, the nature of its design is to guarantee its input and output sequences contain similar information. However, in automatic commit message generation, the inputs are typically code changes. Codes are generally not concise *per se* to describe their meanings, which is why we require commit messages [1]. The lengthy, complicated codes contains far more information than that required in NMT. We suggest that proper preprocessing of code changes is also critical to improve NMT.

To verify our idea, we propose code semantic analysis [13], [14] to preprocess code changes and generate more concise inputs for NMT. This paper reports our experience in this attempt. We collect a real-world dataset with 50k+ commits in 18 top popular GitHub Java projects. We adopt a state-of-the-art semantic analysis method [5] to preprocess the data. A comprehensive evaluation is conducted to prove that data preprocessing does improve NMT. Finally, we also open our dataset to facilitate further research.

II. Related Work

With the recent development of deep learning, NMT has been proposed to generate commit messages automatically. Jiang *et al.* [6] and Loyola *et al.* [7] apply the encoder-decoder architecture to automatically generate commit messages from code changes. Hu *et al.* [15] suggest NMT to summarize codes in text. Much work has focused on improving the algorithms in NMT for commit message generation [9], [11], [12].

This work aims at showing input preprocessing is critical to NMT. Since the core of commit message generation is to summarize code changes, we survey related research efforts as follows. Code changes can be summarized by analyzing edit distance between two code versions [16]–[19]. Codes are typically modeled as abstract syntax tree (AST) and the tree difference can summarize the changes [20], [21]. Recent approaches include GumTree [22] and many improvements [23], [24]. A recent method to analyze AST difference is proposed by Huang *et al.* [25]. Code change analysis can further help generate human-readable texts. Buse and Weimer [4] suggest the impact of code changes on runtime can be inferred by symbolic execution. ChangeScribe [5], [26] can fill the predefined templates with key information in AST-based code changes. Shen *et al.* [27] further suggest to polish the results of ChangeScribe.

There are also many other measures to model codes, including control flow graph [28] and UML class model [29]. Le *et al.* [30] propose a dynamic framework to infer code changes.

978-1-7281-2508-4/19/$31.00 ©2019 IEEE
DOI 10.1109/ASE.2019.00162

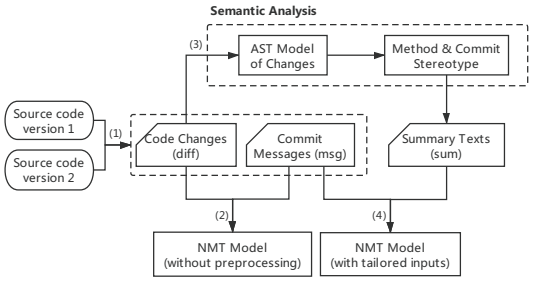

Fig. 1. Approach Overview

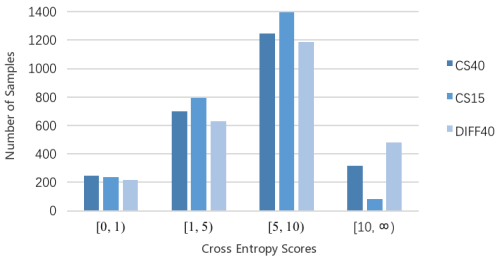

Fig. 2. Cross entropy loss scores on the test set

Other studies [31]–[34] suggest software documents can be used to summarize code changes.

III. Tailoring Inputs for NMT-based Commit Message Generation

Traditional NMT-based commit message generation typically takes raw code changes as inputs. It trains its DNN with the code changes in commits, and their corresponding good, manually-written commit messages in existing projects, shown as steps (1) and (2) in Fig 1. The core idea of NMT is to learn a best sequence-to-sequence mapping from the inputs to the outputs. Then, given new code changes as an input sequence, NMT can generate a new text sequence as its commit message.

As we discussed, we propose that the input to NMT should be carefully tailored to adapt to the commit message generation task. We suggest that code semantic analysis should be applied to polish the code changes first into short descriptions that summarize such changes. To this end, we apply ChangeScribe [5], a well-designed semantic analysis tool to perform such input preprocessing. The approach extracts code changes in AST between two source-code versions. It then identifies method stereotypes [35] and commit stereotypes [36] by analyzing the AST. The stereotypes describe the main behavior of this commit. Finally, it generates summary texts by filling a predefined template.

Note that the summary texts generalized by ChangeScribe are not instantly ready as commit messages. The reason is straight-forward: ChangeScribe simply enumerates code changes without describing their purposes. Hence we adopt NMT which, learning from previous human experiences, translates the summary texts into more meaningful messages.

Finally, similarly as in traditional NMT-based commit message generation, our approach also trains a DNN with the preprocessed code changes in commits, and their corresponding good, manually-written commit messages in existing projects, shown as steps (1), (3), and (4) in Fig 1. Next, we present our experimental study to verify our idea with real-world cases.

IV. Experimental Study

We collect 18 popular Java projects from GitHub. Each project has accumulated 20k+ stars (positive user comments), contains 100k+ lines of codes and 3k+ commits averagely. We use ChangeScribe to summarize code changes for each commit. We also preprocess the data with the following steps: 1) deleting labels and redundant statements in summary texts; 2) clearing empty texts; 3) limiting the text length to 100. The dataset has 50k+ commits in total. We select 45k+ for training NMT methods, 2.5k for testing, and 2.5k for validation with systematic sampling method. Our dataset is released on https://github.com/ShuyaoJiang/CommitDataset.

We use Nematus [37] to train the NMT models due to its robustness, usability, and excellent performance [38]. Its DNN architecture is *attentional RNN encoder-decoder*, a state-of-the-art model for the NMT task. We train two models, namely, CS40 and CS15 based on our polished, short summary of code changes with ChangeScribe. The two models are both based on Nematus with batch size values 40 and 15. For comparison purpose, we also train a model with raw code changes, namely, DIFF40, based on Nematus with batch size value 40.

We use cross entropy loss and BLEU [10] to evaluate the model performance on the test set. Figure 2 shows the results of the cross entropy loss values. Our approach can obtain more commit messages with low cross entropy loss. In other words, more results of CS40 and CS15 are similar to human-written commit messages than those of DIFF40. Table I further shows that CS40 has the highest BLEU scores, indicating it outperforms DIFF40. It further confirms that input preprocessing is helpful to NMT.

TABLE I
BLEU Scores on the Test Set

Model	BLEU-4	p_1	p_2	p_3	p_4
CS40	1.10	4.7	1.7	0.5	0.4
CS15	0.44	9.1	3.2	0.1	0.0
DIFF40	0.41	3.9	0.9	0.1	0.1

p_n is the modified n-gram precision used to calculate BLEU-4 [10].

V. Conclusion

This paper contributes to the promising line of research that considers codes as sequences and conducts machine code-comprehension with deep learning. We show the importance of tailoring the inputs. It is promising that tremendous existing DNNs designed by the machine learning community, *e.g.*, NMT models, can be instantly adopted, with task-specifically-tailored inputs, to complete software engineering tasks.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (Project No. 61672164) and a CERNET Innovation Project (No. NGII20180110).

REFERENCES

[1] P. Hallam, "What do programmers really do anyway," *Microsoft Developer Network (MSDN) C# Compiler*, 2006.

[2] W. Maalej and H.-J. Happel, "Can development work describe itself?" in *Proc. of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 2010, pp. 191–200.

[3] R. Dyer, H. A. Nguyen, H. Rajan, and T. N. Nguyen, "BOA: A language and infrastructure for analyzing ultra-large-scale software repositories," in *Proc. of International Conference on Software Engineering*, 2013, pp. 422–431.

[4] R. P. Buse and W. Weimer, "Automatically documenting program changes." in *ASE*, vol. 10, 2010, pp. 33–42.

[5] L. F. Cortés-Coy, M. Linares-Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *Proc. of the 14th IEEE International Working Conference on Source Code Analysis and Manipulation*, 2014, pp. 275–284.

[6] S. Jiang, A. Armaly, and C. McMillan, "Automatically generating commit messages from diffs using neural machine translation," in *Proc. of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 135–146.

[7] P. Loyola, E. Marrese-Taylor, and Y. Matsuo, "A neural architecture for generating natural language descriptions from source code changes," *arXiv preprint arXiv:1704.04856*, 2017.

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[9] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proc. of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 373–384.

[10] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," in *Proc. of the 40th Annual Meeting on Association for Computational Linguistics*, 2002, pp. 311–318.

[11] S. Gao, C. Chen, Z. Xing, Y. Ma, W. Song, and S. Lin, "A neural model for method name generation from functional description," in *Proc. of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 414–421.

[12] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *Proc. of the 27th International Conference on Program Comprehension*, 2019, pp. 70–80.

[13] G. Cosma and M. Joy, "An approach to source-code plagiarism detection and investigation using latent semantic analysis," *IEEE Transactions on Computers*, vol. 61, no. 3, pp. 379–394, 2011.

[14] A. Kuhn, S. Ducasse, and T. Gírba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, 2007.

[15] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proc. of the 26th ACM Conference on Program Comprehension*, 2018, pp. 200–210.

[16] W. Miller and E. W. Myers, "A file comparison program," *Software: Practice and Experience*, vol. 15, no. 11, pp. 1025–1040, 1985.

[17] S. Reiss, "Tracking source locations," in *Proc. of the 30th ACM/IEEE International Conference on Software Engineering*, 2008, pp. 11–20.

[18] G. Canfora, L. Cerulo, and M. Di Penta, "Tracking your changes: A language-independent approach," *IEEE Software*, vol. 26, no. 1, pp. 50–57, 2009.

[19] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and M. Di Penta, "Lhdiff: A language-independent hybrid approach for tracking source code lines," in *Proc. of IEEE International Conference on Software Maintenance*, 2013, pp. 230–239.

[20] B. Fluri, M. Wuersch, M. PInzger, and H. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 725–743, 2007.

[21] M. Hashimoto and A. Mori, "Diff/ts: A tool for fine-grained structural change analysis," in *Proc. of the 15th Working Conference on Reverse Engineering*, 2008, pp. 279–288.

[22] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Monperrus, "Fine-grained and accurate source code differencing," in *Proc. of the 29th ACM/IEEE International Conference on Automated Software Engineering*, 2014, pp. 313–324.

[23] Y. Higo, A. Ohtani, and S. Kusumoto, "Generating simpler ast edit scripts by considering copy-and-paste," in *Proc. of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 532–542.

[24] G. Dotzler and M. Philippsen, "Move-optimized source code tree differencing," in *Proc. of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 660–671.

[25] K. Huang, B. Chen, X. Peng, D. Zhou, Y. Wang, Y. Liu, and W. Zhao, "Cldiff: generating concise linked code differences," in *Proc. of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 679–690.

[26] M. Linares-Vásquez, L. F. Cortés-Coy, J. Aponte, and D. Poshyvanyk, "Changescribe: A tool for automatically generating commit messages," in *Proc. of the 37th IEEE/ACM International Conference on Software Engineering*, vol. 2, 2015, pp. 709–712.

[27] J. Shen, X. Sun, B. Li, H. Yang, and J. Hu, "On automatic summarization of what and why information in source code changes," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2016, pp. 103–112.

[28] T. Apiwattanapong, A. Orso, and M. J. Harrold, "A differencing algorithm for object-oriented programs," in *Proc. of the 19th International Conference on Automated Software Engineering*. IEEE, 2004, pp. 2–13.

[29] Z. Xing and E. Stroulia, "Umldiff: an algorithm for object-oriented design differencing," in *Proc. of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 54–65.

[30] T.-D. B. Le, J. Yi, D. Lo, F. Thung, and A. Roychoudhury, "Dynamic inference of change contracts," in *Proc. of the IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 451–455.

[31] S. Rastkar and G. C. Murphy, "Why did this code change?" in *Proc. of International Conference on Software Engineering*, 2013, pp. 1193–1196.

[32] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "Automatic generation of release notes," in *Proc. of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014, pp. 484–495.

[33] ——, "Arena: an approach for the automated generation of release notes," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 106–127, 2017.

[34] Y. Huang, Q. Zheng, X. Chen, Y. Xiong, Z. Liu, and X. Luo, "Mining version control system for automatically generating commit comment," in *Proc. of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2017, pp. 414–423.

[35] N. Dragan, M. L. Collard, and J. I. Maletic, "Reverse engineering method stereotypes," in *Proc. of the 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 24–34.

[36] N. Dragan, M. L. Collard, M. Hammad, and J. I. Maletic, "Using stereotypes to help characterize commits," in *Proc. of the 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 520–523.

[37] R. Sennrich, O. Firat, K. Cho, A. Birch, B. Haddow, J. Hitschler, M. Junczys-Dowmunt, S. Läubli, A. V. M. Barone, J. Mokry *et al.*, "Nematus: a toolkit for neural machine translation," *arXiv preprint arXiv:1703.04357*, 2017.

[38] R. Sennrich, B. Haddow, and A. Birch, "Edinburgh neural machine translation systems for wmt 16," *arXiv preprint arXiv:1606.02891*, 2016.