Improving Bug Detection and Fixing via Code Representation Learning

Yi Li
New Jersey Institute of Technology, USA
yl622@njit.edu

ABSTRACT

The software quality and reliability have been proved to be important during the program development. There are many existing studies trying to help improve it on bug detection and automated program repair processes. However, each of them has its own limitation and the overall performance still have some improvement space. In this paper, we proposed a deep learning framework to improve the software quality and reliability on these two detect-fix processes. We used advanced code modeling and AI models to have some improvements on the state-of-the-art approaches. The evaluation results show that our approach can have a relative improvement up to 206% in terms of F-1 score when comparing with baselines on bug detection and can have a relative improvement up to 19.8 times on the correct bug-fixing amount when comparing with baselines on automated program repair. These results can prove that our framework can have an outstanding performance on improving software quality and reliability in bug detection and automated program repair processes.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**;

ACM Reference Format:
Yi Li. 2020. Improving Bug Detection and Fixing via Code Representation Learning. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3377812.3382172

1 INTRODUCTION

Improving software quality and reliability is a never-ending demand [3, 4, 6, 12, 16, 22]. One study from the US Department of Commerce' National Institute of Standards and Technology (NIST) concluded that software bugs cause serious loss, about $59 billion or about 0.6 percent of the GDP, each year. Various approaches can help improve the software quality and reliability [5, 8, 9, 14, 21, 24], such as bug detection and automated program repair (APR).

Existing Approaches. In this research, we focus on the detection and auto-fixing of bugs. Thus, we briefly introduce the related studies in bug detection and automated program repair problems with their main limitations.

There have been three different types of bug detection approaches have been proposed in the literature, including rule based [13], mining based [5, 9, 14, 21], and machine learning based [28–30]. All of these existing approaches have some limitations, rule based ones need manually defined rules for new types of bugs, and the mining and machine learning based ones typically have high false-positive rates. Through our previous study [19], we observed that the existing approaches, especially machine learning based ones, do not work well on detecting cross-method bugs and their code modeling is not effective and accurate.

In the APR research, there are two main streams of approaches, including pattern based [15, 17, 20, 23] and learning based approaches [10, 11, 25, 31]. The pattern based approaches need generated rules. However, the learning based approaches have a hard time learning code changes and the context of the surrounding code which may lead to lower accuracy and wrong fixing positions. Through our previous study [18], we observed that the existing state-of-the-art APR approaches do not work well on separating and modeling the buggy code and its surround code context.

Our Work. In this research, we aim to improve the existing state-of-the-art bug detection and auto-fixing (namely detect-fix) approaches via accurate, effective, and specialized code representation learning. Our code representation learning relies on the following pillars: code representations (i.e., data structures) obtained from advanced program analysis and deep neural network models.

Currently, we focus on two detect-fix processes: Bug Detection (BD) and Automated Program Repair (APR). To overcome the limitations of the state-of-the-art BD and APR approaches, we propose to improve DB and APR as follows:

Bug Detection. To identify cross-method bugs and have effective code modeling, in our previous study [19], we first extract paths from Abstract Syntax Trees of code methods for local code contexts, then use program dependency and data flow graphs to model relations among methods. We come up with a new neural network based code representation learning model specialized for bug detection by adding a weight to buggy code, considering method code relationship with graphs, and using AST paths to represent the code methods. Our empirical results on a corpus of 5 million Java methods show that our bug detection specialized detector can improve the state-of-the-art baselines by up to 206%.

APR. To separate and model bug fixes and their surrounding unchanged code as contexts, in our previous study [18], we propose a two-layer tree-based model, namely DLFix to learn code transformations from buggy to healthy code.

Therefore, in our two-layer model, the first layer is used to learn the surrounding code context and the other one is used to learn the buggy code fixing. Our code representation learning is based on

ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05…$15.00
https://doi.org/10.1145/3377812.3382172

these two layers to help improve the APR performance. Our empirical results show that our DLFix can outperform all studied Deep Learning based APR approaches, also generate comparable results compared with the most state-of-the-art pattern-based approaches.

2 OUR APPROACH

Bug Detection. In our approach [19], we use deep learning models with graphs to catch code context information and code relationship information. Then we use a CNN layer with softmax as a classifier to do the bug detection. Specifically, our approach works in three phases. We first learn local context by extracting the paths along with the AST's nodes, converting them into vectors using a Gated Recurrent Unit (GRU) layer [7] and an attention Convolutional layer [32], combining all vectors using Multi-Head Attention [26] to obtain the path *local context* representation. Second, to generate the *global context* modeling relations among paths from methods, we build the program dependency and data flow graphs and extract the subgraphs relevant to a method. After having both local and global context representations for each path, we can get the representation for each method by directly linking all merged path vectors.

The uniqueness of our approach: (1) using program dependency and data flow graphs to catch code relationship among methods; and (2) adding weights to buggy paths when doing the training for specializing our code representation learning for bug detection.

Automated Program Repair. In our approach [18], we propose a two-layer tree-based deep learning model, namely DLFix, to learn code transformations by using one layer to learn the surrounding code and the other one layer to learn the bug-fixing changes. We *separate the learning of the context of surrounding code* of bug fixes from *the learning of the code transformations* for bug fixes with two layers in our model. The changed (buggy) sub-tree in the AST of a buggy method is identified and replaced with a summarized node using a deep-learning based code summarization technique [27]. The un-changed AST sub-trees together with the summarized node constitute the context and are learned with a RNN model at the context learning layer. Following existing state-of-the-art APR tools, DLFix is designed for one statement auto-fixing.

The uniqueness of our approach: A novel two-layer tree-based code transformation learning model.

3 EVALUATION

Bug Detection: *Dataset and Metrics.* We evaluated our approach and the baselines on eight well-known and large open-source Java projects with 92 versions +4.9 million Java methods. We mainly use F-score as the evaluation metric.

Results. Our key empirical results show that our approach can have a relative improvement up to 160% in terms of F-score when comparing with other baselines in the cross-project settings in Fig. 1. Due to the page limit, more results can be found in [19].

Automated Program Repair: *Dataset and Metrics.* We did the experiments on well-known dataset Defects4J [1]. We use the number of auto-fixed bugs as the evaluation metric.

Results. Fig. 2 shows that DLFix can auto-fix 30 bugs and its results are comparable and complementary to the top APR tools (Simfix, Hercules, and TBar) **on one statement auto-fixing** with the Ochiai [2] as the fault localization. Also, DLFix outperformed

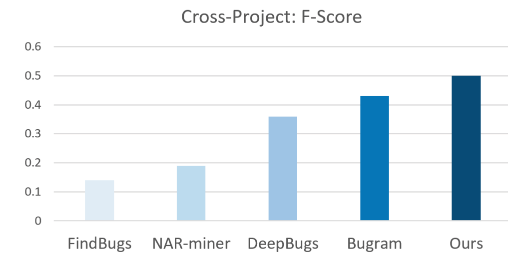

Figure 1: Comparison with the Bug Detection Baselines in Cross-Project Setting

all of the existing Deep Learning based APR tools. DLFix can fix 2.5 times more bugs than the best performing Deep Learning baseline. Due to the page limit, more results can be found in [18].

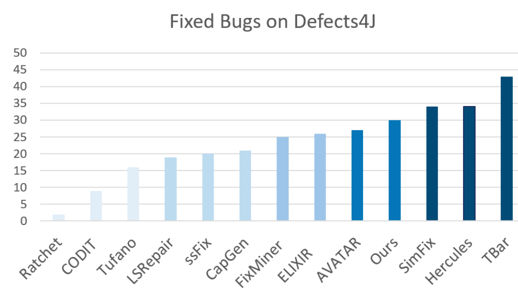

Figure 2: Comparison with the APR Baselines on Defect4J

4 ADVANCING REPRESENTATION LEARNING TO IMPROVE BUG DETECTION AND FIXING

We plan to improve and advance code modeling in the following detect-fix bug process:

- **Bug Detection:** Using code representation learning models to explain bug types when doing bug detection.
- **Fault Localization:** Applying deep learning models on code coverage information to improve the code representation learning to locate bugs.
- **Automated Program Repair:** Improving the code representation learning model to repair multi line bugs.
- **Concolic Testing:** Applying code representation learning models to generate test cases for execution paths.

5 CONCLUSION

In this research, we proposed two novel code modeling approaches to improve two processes: bug detection and automated program repair. The key ideas that enable our work: using code representation learning models can help improve the state-of-the-art approaches on bug detection and APR. Our evaluation results on published papers [18, 19] could prove our model and key ideas can work well.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Shaohua Wang from Informatics, New Jersey Institute of Technology, and Dr. Tien N. Nguyen from Computer Science, The University of Texas at Dallas who gave valuable suggestions to improve the quality of the research and language editing.

REFERENCES

[1] 2019. The Defects4J Data Set. https://github.com/rjust/defects4j

[2] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. 2006. An Evaluation of Similarity Coefficients for Software Fault Localization. In *Proceeding of the 12th Pacific Rim International Symposium on Dependable Computing*. 39–46. https://doi.org/10.1109/PRDC.2006.18

[3] Matthew Amodio, Swarat Chaudhuri, and Thomas W. Reps. 2017. Neural Attribute Machines for Program Generation. *CoRR* abs/1705.09231 (2017). arXiv:1705.09231

[4] Sahil Bhatia and Rishabh Singh. 2016. Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks. *CoRR* abs/1603.06129 (2016). arXiv:1603.06129

[5] Pan Bian, Bin Liang, Wenchang Shi, Jianjun Huang, and Yan Cai. 2018. NAR-miner: Discovering Negative Association Rules from Code for Bug Detection. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 411–422. https://doi.org/10.1145/3236024.3236032

[6] Pavol Bielik, Veselin Raychev, and Martin Vechev. 2016. PHOG: Probabilistic Model for Code. In *Proceedings of the 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.), Vol. 48. PMLR, New York, New York, USA, 2933–2942.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* abs/1406.1078 (2014). arXiv:1406.1078

[8] Brian Cole, Daniel Hakim, David Hovemeyer, Reuven Lazarus, William Pugh, and Kristin Stephens. 2006. Improving Your Software Using Static Analysis to Find Bugs. In *Proceeding of the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06)*. ACM, New York, NY, USA, 673–674. https://doi.org/10.1145/1176617.1176667

[9] Dawson Engler, David Yu Chen, Seth Hallem, Andy Chou, and Benjamin Chelf. 2001. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. (2001), 57–72. https://doi.org/10.1145/502034.502041

[10] Rahul Gupta, Soham Pal, Aditya Kanade, and Shirish Shevade. 2017. DeepFix: Fixing Common C Language Errors by Deep Learning. https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14603

[11] Hideaki Hata, Emad Shihab, and Graham Neubig. 2018. Learning to generate corrective patches using neural machine translation. *arXiv preprint arXiv:1812.07170* (2018).

[12] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the Naturalness of Software. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 837–847. http://dl.acm.org/citation.cfm?id=2337223.2337322

[13] David Hovemeyer and William Pugh. 2007. Finding More Null Pointer Bugs, but Not Too Many. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE '07)*. ACM, New York, NY, USA, 9–14. https://doi.org/10.1145/1251535.1251537

[14] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. 2012. Understanding and Detecting Real-World Performance Bugs. (2012), 77–88. https://doi.org/10.1145/2254064.2254075

[15] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic patch generation learned from human-written patches. In *Proceeding of the 35th International Conference on Software Engineering (ICSE)*. 802–811. https://doi.org/10.1109/ICSE.2013.6606626

[16] Hyeji Kim, Yihan Jiang, Sreeram Kannan, Sewoong Oh, and Pramod Viswanath. 2018. Deepcode: Feedback Codes via Deep Learning. *CoRR* abs/1807.00801 (2018). arXiv:1807.00801

[17] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2012. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering* 38, 1 (Jan 2012), 54–72. https://doi.org/10.1109/TSE.2011.104

[18] Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. DLFix: Context-based Code Transformation Learning for Automated Program Repair. ICSE (2020).

[19] Yi Li, Shaohua Wang, Tien N Nguyen, and Son Van Nguyen. 2019. Improving bug detection via context-based code representation learning and attention-based neural networks. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 162.

[20] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F Bissyandé. 2019. AVATAR: Fixing Semantic Bugs with Fix Patterns of Static Analysis Violations. In *Proceeding of the 26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 1–12. https://doi.org/10.1109/SANER.2019.8667970

[21] Oswaldo Olivo, Isil Dillig, and Calvin Lin. 2015. Static Detection of Asymptotic Performance Bugs in Collection Traversals. (2015), 369–378. https://doi.org/10.1145/2737924.2737966

[22] Jibesh Patra and Michael Pradel. 2016. Learning to Fuzz: Application-Independent Fuzz Testing with Probabilistic, Generative Models of Input Data.

[23] Yuhua Qi, Xiaoguang Mao, Yan Lei, Ziying Dai, and Chengsong Wang. 2014. The Strength of Random Search on Automated Program Repair. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 254–265. https://doi.org/10.1145/2568225.2568254

[24] John Toman and Dan Grossman. 2017. Taming the Static Analysis Beast. In *2nd Summit on Advances in Programming Languages (SNAPL 2017) (Leibniz International Proceedings in Informatics (LIPIcs))*, Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi (Eds.), Vol. 71. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 18:1–18:14. https://doi.org/10.4230/LIPIcs.SNAPL.2017.18

[25] Michele Tufano, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, Martin White, and Denys Poshyvanyk. 2018. An Empirical Investigation into Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 832–837. https://doi.org/10.1145/3238147.3240732

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762

[27] Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. 2018. Improving Automatic Source Code Summarization via Deep Reinforcement Learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, 397–407.

[28] Song Wang, Devin Chollak, Dana Movshovitz-Attias, and Lin Tan. 2016. Bugram: Bug Detection with N-gram Language Models. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*. ACM, New York, NY, USA, 708–719. https://doi.org/10.1145/2970276.2970341

[29] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically Learning Semantic Features for Defect Prediction. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 297–308. https://doi.org/10.1145/2884781.2884804

[30] Andrzej Wasylkowski, Andreas Zeller, and Christian Lindig. 2007. Detecting Object Usage Anomalies. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC-FSE '07)*. ACM, New York, NY, USA, 35–44. https://doi.org/10.1145/1287624.1287632

[31] Martin White, Michele Tufano, Matias Martinez, Martin Monperrus, and Denys Poshyvanyk. 2019. Sorting and transforming program repair ingredients via deep learning code similarities. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 479–490.

[32] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. 2015. ABCNN: Attention-Based Convolutional Neural Network for Modeling Sentence Pairs. *CoRR* abs/1512.05193 (2015). arXiv:1512.05193