Testing Neural Program Analyzers

Md Rafiqul Islam Rabin
University of Houston
mdrafiqulrabin@gmail.com

Ke Wang
Visa Research
kewangad@gmail.com

Mohammad Amin Alipour
University of Houston
amin.alipour@gmail.com

***Abstract*—Deep neural networks have been increasingly used in software engineering and program analysis tasks. They usually take a program and make some predictions about it, e.g., bug prediction. We call these models *neural program analyzers*. The reliability of neural program analyzers can impact the reliability of the encompassing analyses.**

In this paper, we describe our ongoing efforts to develop effective techniques for testing neural program analyzers. We discuss the challenges involved in developing such tools and our future plans. In our preliminary experiment on a neural model recently proposed in the literature, we found that the model is very brittle, and simple perturbations in the input can cause the model to make mistakes in its prediction.

I. Introduction

The advances of deep neural models in software engineering and program analysis research have received significant attention in recent years. Researchers have already proposed various neural models (e.g., Tree-LSTM [11], Gemini [18], GGNN [1], Code Vectors [7], code2vec [3], code2seq [2], DYPRO [14, 16], LIGER [17], Import2Vec [12]) to solve problems related to different program analysis or software engineering tasks. Although each neural model has been evaluated by its authors, in practice, these neural models may be susceptible to untested test inputs. Therefore, a set of testing approaches has already been proposed to trace the unexpected corner cases. Recent neural model testing techniques include [13, 19] for models of autonomous systems, [8]–[10] for models of QA systems, and [15, 17] for models of embedding systems. However, testing neural models that work on source code has received little attention from researchers except the exploration initiated by Wang et al. [15].

Evaluating the robustness of neural models that process source code is of particular importance because their robustness would impact the correctness of the encompassing analyses that use them. In this paper, we propose a transformation-based testing framework to test the correctness of state-of-the-art neural models running on the programming task. The transformation mainly refers to the semantic changes in programs that result in similar programs. The key insight of transformation is that the transformed programs are semantically equivalent to their original forms of programs but have different syntactic representations. For example, one can replace a `switch` statement of a program with conditional `if-else` statements. The original program of the `switch` statement is semantically equivalent to the new program of `if-else` statements. A set of transformations can be applied to a program to generate more semantically equivalent programs, and those new transformed programs can be evaluated on neural models to test the correctness of those models.

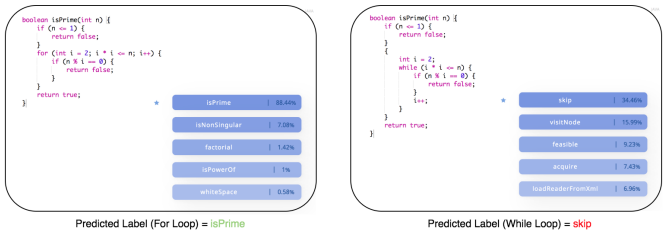

Fig. 1. A failure in the `code2vec` model revealed by a transformation.

The main motivation to apply transformation is the fact that such transformations may cause the neural model to behave differently and mispredict the input. We are conducting a small study to assess the applicability of transformations in the testing of neural models. The preliminary results show that the transformations are very effective in finding irrelevant output in neural models. We closely perceive that the semantic-preserving transformations can change the predicted output or the prediction accuracy of neural models compared to the original test programs.

II. Motivating Example

We use Figure 1 as a motivating example to highlight the usefulness of our approach. The code snippet shown in Figure 1 is a simple Java method that demonstrates the prime functionality. The functions check whether an integer is a prime number. The only difference between these functions is that the implementation on the left uses a `for` loop, while the implementation on the right uses a `while` loop.

We instrument the prediction of the `code2vec` model [5] with these two equivalent functions. The `code2vec` takes a program and predicts its content. The result of the online demo [5] reveals that the `code2vec` model successfully predicts the program on the left as an “isPrime” method, but cannot predict the program on the right as an “isPrime” method. The model mistakenly predicts the program on the right as a “skip” method, even though the “isPrime” method is not included in the top-5 predictions made by the `code2vec` model.

III. Proposed Methodology

In this section, we describe our efforts for testing neural program analyzers. Currently, we are investigating semantic-preserving transformations that can potentially mislead a neural model of programs.

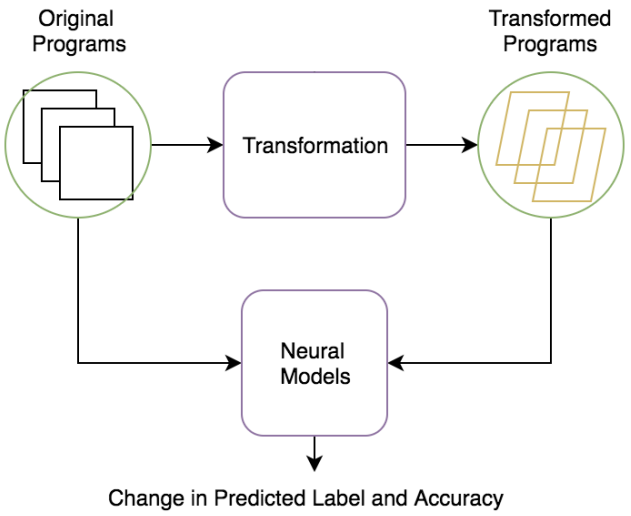

Fig. 2. The workflow of our approach.

Figure 2 depicts an overview of our approach for testing the neural models. It can broadly be divided into two main steps: (1) Generating synthetic test programs using the semantic transformation of the programs in the original dataset, and (2) Comparing the predictions for the transformed programs with those for the original programs.

Semantic-Equivalent Program Transformations We have implemented multiple semantic program transformations to generate synthetic programs. Those semantic-preserving transformations include renaming variables, exchanging loops, swapping boolean values, converting switches and permuting the order of statements. In variable-renaming transformation, we rename all the occurrences of specific variables in program using an arbitrary name. The boolean-swapping transformation refers to swapping `true` with `false` and vice versa, and we also neglect the condition so that the semantic is maintained. In the same way, the loop-exchanging transformation means replacing a `while` loop with a `for` loop, and vice versa. In switch-converting transformation, we replace the `switch` statements with the conditional `if-else` statements. Finally, we include another transformation by permuting the order of statements without any semantic violations. All these transformations maintain semantic equivalence but generate different syntactic programs. Thus far, we have not found any one transformation that works substantially better than others.

Test Oracle We evaluate both the original program and the transformed program in the neural model. We mainly look at the predicted label and the prediction accuracy of the model for both original and transformed programs. The neural model should behave similarly with both the original and the transformed program, which we define as a transformation-based metamorphic relation. The main challenge in this phase is to define a measure for the *similarity* of the predictions. We are experimenting with a few ideas for this phase, for example, setting a threshold for the similarity of the predictions.

Challenges Ahead There are five main challenges that we are aiming to address in this project: (1) what types of transformation should be performed, (2) how to preserve the semantic equivalence during transformations, (3) where to apply those transformations, (4) how to control the transformation strategies, and (5) how to evaluate the transformed programs.

IV. Our Plan

Thus far, we have applied five types of transformation. Those transformations are only capable of making basic changes in the syntactic representations of programs. However, our target is to devise more systematic transformations. We are investigating the techniques and heuristics to suggest *places* in programs to transform, and the *types of transformation* that are most likely to cause the neural model to mispredict.

Moreover, we have only evaluated our transformation on the `code2vec` model [3], where the target task is to label the method name given a method body. We also plan to evaluate the transformation on the `GGNN` model [1], where the target task is to label the correct variable name based on the understanding of its usage.

Additionally, we have only experimented with a small set of examples [5]. Our further plan includes a detailed study with a larger Java dataset [4] for the `code2vec` model and a larger C# dataset [6] for the `GGNN` model.

V. Related Work

Several approaches for transformation-based testing have been proposed, such as DeepTest [13] and COSET [15].

Tian et al. [13] proposed DeepTest, a tool for automated generation of real-world test images and testing of DNN-driven autonomous cars. They introduced potential image transformations (e.g., blurring, scaling, fog and rain effects) that mimic real-world conditions. They applied transformation-based testing to identify the numerous corner cases that may lead to serious consequences, such as a collision in an autonomous car. Another study in this area was conducted by the authors of DeepRoad [19], who applied extreme realistic image-to-image transformations (e.g., heavy snow or hard rain) using the DNN-based UNIT method.

Wang et al. [15] proposed COSET, a framework for standardizing the evaluation of neural program embeddings. They applied transformation-based testing to measure the stability of neural models and identify the root cause of misclassifications. They also implemented and evaluated a new neural model called LIGER [17] with COSET's transformations, where they embedded programs with runtime information rather than learning from the source code.

References

[1] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," *CoRR*, vol. abs/1711.00740, 2017. [Online]. Available: http://arxiv.org/abs/1711.00740

[2] U. Alon, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," *CoRR*, vol. abs/1808.01400, 2018. [Online]. Available: http://arxiv.org/abs/1808.01400

[3] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *CoRR*, vol. abs/1803.09473, 2018. [Online]. Available: http://arxiv.org/abs/1803.09473

[4] "CODE2VEC Dataset," https://github.com/tech-srl/code2vec#additional-datasets/.

[5] "CODE2VEC Online Demo," https://code2vec.org/.

[6] "GGNN Dataset," https://aka.ms/iclr18-prog-graphs-dataset/.

[7] J. Henkel, S. Lahiri, B. Liblit, and T. W. Reps, "Code vectors: Understanding programs through embedded abstracted symbolic traces," *CoRR*, vol. abs/1803.06686, 2018. [Online]. Available: http://arxiv.org/abs/1803.06686

[8] Q. Lei, L. Wu, P. Chen, A. G. Dimakis, I. S. Dhillon, and M. Witbrock, "Discrete attacks and submodular optimization with applications to text classification," *CoRR*, vol. abs/1812.00151, 2018. [Online]. Available: http://arxiv.org/abs/1812.00151

[9] M. T. Ribeiro, S. Singh, and C. Guestrin, "Semantically equivalent adversarial rules for debugging NLP models," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 856–865. [Online]. Available: https://www.aclweb.org/anthology/P18-1079

[10] B. Rychalska, D. Basaj, P. Biecek, and A. Wróblewska, "Does it care what you asked? understanding importance of verbs in deep learning QA system," *CoRR*, vol. abs/1809.03740, 2018. [Online]. Available: http://arxiv.org/abs/1809.03740

[11] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *CoRR*, vol. abs/1503.00075, 2015. [Online]. Available: http://arxiv.org/abs/1503.00075

[12] B. Theeten, F. Vandeputte, and T. Van Cutsem, "Import2vec learning embeddings for software libraries," in *Proceedings of the 16th International Conference on Mining Software Repositories*, ser. MSR '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 18–28. [Online]. Available: https://doi.org/10.1109/MSR.2019.00014

[13] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180220

[14] K. Wang, "Learning scalable and precise representation of program semantics," *CoRR*, vol. abs/1905.05251, 2019. [Online]. Available: http://arxiv.org/abs/1905.05251

[15] K. Wang and M. Christodorescu, "COSET: A benchmark for evaluating neural program embeddings," *CoRR*, vol. abs/1905.11445, 2019. [Online]. Available: http://arxiv.org/abs/1905.11445

[16] K. Wang, R. Singh, and Z. Su, "Dynamic neural program embedding for program repair," *CoRR*, vol. abs/1711.07163, 2017. [Online]. Available: http://arxiv.org/abs/1711.07163

[17] K. Wang and Z. Su, "Learning blended, precise semantic program embeddings," *ArXiv*, vol. abs/1907.02136, 2019.

[18] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," *CoRR*, vol. abs/1708.06525, 2017. [Online]. Available: http://arxiv.org/abs/1708.06525

[19] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE 2018. New York, NY, USA: ACM, 2018, pp. 132–142. [Online]. Available: http://doi.acm.org/10.1145/3238147.3238187