Integrating Grammatical Evolution with Neural Fitness Functions

Claudio Ferretti[1] and Martina Saletta[1]

Dept. of Informatics, Systems and Communication (DISCo),
University of Milano-Bicocca, Milan, Italy
{claudio.ferretti, martina.saletta}@unimib.it

Abstract. Artificial neural networks for source code processing are often able to develop rich representations for different program properties. In this software demo, we exploit the knowledge learned by a neural network, i.e. the activation of a chosen neuron, for guiding the evolution of programs that satisfy given requirements.

1 Introduction

Grammatical evolution (GE) [8], an evolutionary technique for evolving individuals that comply with a given formal grammar, despite some limitations in solving general related tasks [11], has proven to be effective for addressing the problem of program synthesis under certain conditions [5]. Crucial aspects in this research domain are the design of an appropriate grammar and the choice of a suitable fitness function.

In the literature, there exists a wide range of machine learning (ML) systems used for solving different source code-related tasks, such as summarization [2], classification [7] or completion [6]. The promising results obtained with such models suggest that ML systems (namely neural networks) are able to learn valuable and diverse properties related to the source code.

The founding idea of our approach is to exploit the knowledge learned by ML systems dealing with source code for leading the evolution of programs that satisfy given specifications or bear certain properties. To this aim, we show how it is possible to effectively use the activation of a neuron in a neural model as the fitness function for a GE algorithm by running some experiments with software we developed.

2 Usage and Examples

In this work, we propose a demonstration for the outlined idea of tapping into the knowledge learned by the neurons in a neural network that works on source code for addressing the challenging task of program synthesis. We show two examples in which, by using the GE implementation provided by the PonyGE2 library [3], GE is used for evolving the individuals and, at each generation, the fitness of each individual is computed by feeding an external neural model with its phenotype and by considering the corresponding output as the fitness of that individual. The proposed examples are the following:

1. A simple usage example in which a minimal grammar that yields the production of strings of bits is used for building strings having an high number of zeros.
2. A demonstration of the approach proposed in [4], in which the authors design a set of experiments for synthesising adversarial examples for misleading the vulnerability detector described in [9].

While the first example is a mere proof of concept and it is designed for allowing a novice to become familiar with the interaction among genetic algorithms, neural networks and formal grammars, the other one is the actual core of this demonstration. We show a direct application of the proposed approach by providing a set of experiments designed for deceiving a neural classifier for software vulnerabilities. Through the application of a simplified C grammar, we show how it is possible to easily evolve programs that maximise or minimise the output of a single neuron that classifies a source code instance as vulnerable or safe with respect to a given vulnerability. Furthermore, the same approach can be used for evolving code snippets that, if injected in a given C function (after the return statement, so not to affect its functionality), are able to switch its classification from safe to vulnerable or vice-versa.

The source code and the complete instructions for running the examples are available at `https://github.com/Martisal/adversarialGE`.

3 Conclusion and Further Directions

Exploiting the behaviour of ML systems for guiding the evolution of programs that satisfy given requirements seems to be promising. In general, the founding idea of using the activation of a neuron for guiding the evolution of programs can be applied also for analysing the internal behaviour of a neural model. A recent work [10] points out how the internal neurons of a neural network trained, in unsupervised mode, in the reconstruction of program vectors are able to autonomously learn different and specific program features. Maximising the activation of such neurons or, possibly, of neurons that exhibit an interesting behaviour with respect to their activation pattern, can be useful both for synthesising programs that satisfy given specification but even for studying and characterising the internal behaviour of a neural network. From a wider perspective, by exploiting the integration between evolutionary algorithms and deep neural networks, we could examine the evolved individuals to understand what the trained systems are actually modeling, and thus to help the research in the growing area that aims at the explainability of artificial intelligence tools [1].

References

1. Adadi, A., Berrada, M.: Peeking inside the black-box: A survey on explainable artificial intelligence (xai). IEEE Access **6**, 52138–52160 (2018)
2. Allamanis, M., Peng, H., Sutton, C.A.: A convolutional attention network for extreme summarization of source code. In: Proceedings of the 33nd International Conference on Machine Learning, ICML. pp. 2091–2100 (2016)
3. Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., O'Neill, M.: Ponyge2: grammatical evolution in python. In: Companion Material Proceedings of Genetic and Evolutionary Computation Conference. pp. 1194–1201. ACM (2017)
4. Ferretti, C., Saletta, M.: Deceiving neural source code classifiers: finding adversarial examples with grammatical evolution. In: GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume. pp. 1889–1897. ACM (2021)
5. Hemberg, E., Kelly, J., O'Reilly, U.: On domain knowledge and novelty to improve program synthesis performance with grammatical evolution. In: GECCO '19: Genetic and Evolutionary Computation Conference. pp. 1039–1046. ACM (2019)
6. Liu, F., Li, G., Wei, B., Xia, X., Fu, Z., Jin, Z.: A self-attentional neural architecture for code completion with multi-task learning. In: Proceedings of the 28th International Conference on Program Comprehension, ICPC. pp. 37–47. ACM (2020)
7. Mou, L., Li, G., Zhang, L., Wang, T., Jin, Z.: Convolutional neural networks over tree structures for programming language processing. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 1287–1293 (2016)
8. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Trans. Evol. Comput. **5**(4), 349–358 (2001)
9. Russell, R.L., Kim, L.Y., Hamilton, L.H., Lazovich, T., Harer, J., Ozdemir, O., Ellingwood, P.M., McConley, M.W.: Automated vulnerability detection in source code using deep representation learning. In: Proceedings of 17th IEEE International Conference on Machine Learning and Applications, ICMLA. pp. 757–762. IEEE (2018)
10. Saletta, M., Ferretti, C.: Mining program properties from neural networks trained on source code embeddings. CoRR **abs/2103.05442** (2021)
11. Sobania, D., Rothlauf, F.: Challenges of program synthesis with grammatical evolution. In: Proceedings of Genetic Programming - 23rd European Conference (EuroGP), held as Part of EvoStar. Lecture Notes in Computer Science, vol. 12101, pp. 211–227. Springer (2020)