Towards Robust Models of Code via Energy-Based Learning on Auxiliary Datasets

Nghi D. Q. Bui
dqnbui.2016@smu.edu.sg
Singapore Management University & Huawei Ireland Research Center

Yijun Yu
yijun.yu@huawei.com
Huawei Ireland Research Center

ABSTRACT

Existing approaches to improving the robustness of source code models concentrate on recognizing adversarial samples rather than valid samples that fall outside of a given distribution, which we refer to as *out-of-distribution* (OOD) samples. To this end, we propose to use an auxiliary dataset (out-of-distribution) such that, when trained together with the main dataset, they will enhance the model's robustness. We adapt energy-bounded learning objective function to assign a higher score to in-distribution samples and a lower score to out-of-distribution samples in order to incorporate such out-of-distribution samples into the training process of source code models. Our evaluation results demonstrate a greater robustness for existing source code models to become more accurate at recognizing OOD data while being more resistant to adversarial attacks at the same time.

ACM Reference Format:
Nghi D. Q. Bui and Yijun Yu. 2022. Towards Robust Models of Code via Energy-Based Learning on Auxiliary Datasets. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Learning code representations (a.k.a. embeddings) and developing a prediction model for programs have been found to be beneficial for a variety of software engineering tasks [4–6, 8, 9, 15, 20, 21, 25]. However, existing source code models suffer from two kinds of robustness problems: (1) adversarial robustness: small, seemingly innocuous perturbations to the input that lead to incorrect predictions [2, 17, 24]; (2) out-of-distribution (OOD) uncertainty arises when a machine learning model sees an input that differs from its training data, and while still predicting on an existing class label by the model regardless. To the best of our knowledge, adversarial robustness for code has been studied recently [2, 17–19, 23, 24], while the OOD detection has been neglected in the research on the robustness of code models. We seek to address the OOD robustness problem by proposing a novel learning method that brings two benefits together: (1) making source code models more resilient to adversarial samples; (2) enabling the detection of OOD samples.

ASE '22, October 10–14, 2022, Rochester, MI, USA

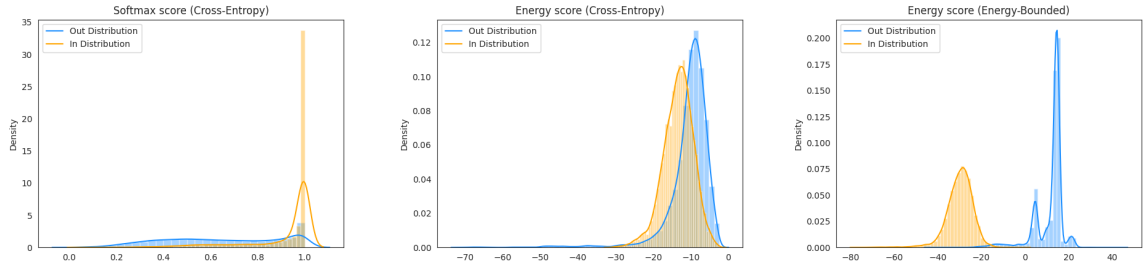
ACM ISBN 978-1-4503-9475-8/22/10…$15.00

To tackle this problem, we aim to enable source code models to say "I don't know" whenever possible instead of making a blind prediction by pretending that they knew the answer. We propose to use an *auxiliary dataset* in addition to the main dataset to train for a specific code learning task. The auxiliary dataset is used as an external resource to improve the model's prediction capability, i.e., the model will know when not to predict something outside of its knowledge. A collection of unlabeled code snippets can be used as the auxiliary dataset. As a result, the auxiliary dataset has the benefit of being inexpensive to collect because one does not need to label its elements. Now, we train the model to generate a single scalar value as the measurement score, so that in-distributions have high scores and out-of-distributions have low scores. We propose to adapt the energy-bounded objective function for the auxiliary dataset that will be jointly trained with the cross-entropy objective function for the main dataset. The additional knowledge from the auxiliary dataset is leveraged to the model during this jointly training phase by pushing the OOD samples further away, in distance, from the in-distribution samples. Since the energy-bounded learning method is agnostic to the cross-entropy objective functions, it is applicable to a wide range of source code models and a several programming tasks. In this work, we evaluated the method on the code classification task train on the Tree-based CNN [13]. The results show that training the source code model with an auxiliary dataset on the energy-bounded objective improves the model's robustness in terms of OOD detection robustness and adversarial robustness.

2 OUR APPROACH

First, an auxiliary dataset including additional code samples will be treated as out-distribution data, while the samples in the main dataset will be treated as in-distribution data. The code samples in the OOD dataset will be encoded in exactly the same way as the samples in the main dataset, except for the steps after obtaining the code embeddings and logits: the energy of both the auxiliary and main code embeddings are computed using an energy-bounded loss function. The intuition of introducing the energy-bound loss function is to assign distinct score ranges to separate out-of-distribution from in-distribution data. The cross-entropy and energy-bounded loss functions are trained jointly in our end-to-end learning framework as the total loss. Here we present the energy-bounded loss function [12], in which the neural network is designed to purposely generate a gap between in-distribution and out-of-distribution data by assigning lower energy to in-distribution data and larger energy to out-of-distribution data. Specifically, the energy-based classifier is trained using the following objective function:

$$\min_{\theta} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{in}}^{\text{train}}}[-\log F_y(x)] + \lambda \cdot L_{\text{energy}} \tag{1}$$



(a) FPR95: 99.71 (b) FPR95: 68.62 (c) FPR95: 0.00

Figure 1: Distribution of softmax scores vs. energy scores from TBCNN-CE vs. TBCNN-EB. (a) FPR95 = 99.71 for the softmax confidence score derived from the TBCNN-CE model. (b) FPR95 = 68.62 for the energy score derived from the TBCNN-CE model. (c) FPR95 = 0.00 for the energy score derived from the TBCNN-EB model.

, where $F(x)$ is the softmax output of the classification model and $\mathcal{D}_{\text{in}}^{\text{train}}$ is the in-distribution training data. The overall training objective combines the standard cross-entropy loss, along with a regularization loss defined in terms of energy:

$$L_{\text{energy}} = \mathbb{E}_{(x_{\text{in}},y)\sim\mathcal{D}_{\text{in}}^{\text{train}}}(\max(0, E(x_{\text{in}}) - m_{\text{in}}))^2 + \mathbb{E}_{x_{\text{out}}\sim \mathcal{D}_{\text{out}}^{\text{train}}}(\max(0, m_{\text{out}} - E(x_{\text{out}})))^2 \quad (2)$$

, where $\mathcal{D}_{\text{out}}^{\text{train}}$ is the unlabeled auxiliary OOD training data.

3 EVALUATION

Datasets: For *in-distribution dataset*, we use the POJ dataset [14] which comprises of 52,000 C programs of 104 classes. For *out-distribution dataset*, we use C data from Project CodeNet [16], a large-scale dataset in multiple programming languages. We use Nicad [3] to remove all of the potential clones that may appears. Then, we randomly sample 60k samples from the OOD datasets to ensure that the data is balanced with the POJ. Then for both the in- and out- distribution dataset, we split them into training/testing/validation with the ratio 70/20/10 into data portions called $in-training$, $in-testing$, $in-validation$ and $out-training$, $out-testing$, $out-validation$.

Training Settings: We train the models using two different loss functions. The first one is cross-entropy loss function used in most of the source code models [22](CE). The second one is the energy-bounded loss function we presented in this study (Equation **??**)(EB). The reason is that we want to see how well the model trained on EB loss compares to the same model trained on CE loss. We choose code classification [13] because it is generic and can represent for the family of classification-based tasks in software engineering, such as malware classification, patch identification, bug triage, etc. Even though the names are different, the general goal is similar, i.e., to classify a piece of code into a given class.

Metrics: We choose false positive rate at $N\%$ true positive rate (*FPRN*) as the metrics for evaluation. The FPRN metric [1, 10, 11] is the probability that an in-distribution example (negative) raises a false alarm when $N\%$ of anomalous examples (positive) are detected, so a lower FPRN is better.

Baselines: In addition to the energy-bounded loss function, we use the softmax score as a baseline to compare against. Note that softmax score has been used as a strong baseline [7] to detect OODs in machine learning.

Evaluation Results: Figure 1 compares the energy and softmax score histogram distributions, derived from the TBCNN model trained on cross-entropy loss (TBCNN-CE), and another TBCNN model trained on energy-bound loss (TBCNN-EB) for the code classification task. As we can see in Figure 1a, the softmax histograms derived from TBCNN-CE for in- and out- distribution makes it difficult to distinguish the two distributions, resulting in FPR95 value of 99.71%. On the other hand, using the energy histograms derived from TBCNN-CE in Figure 1b makes it better to distinguish the two distributions, resulting in FPR95 value of 68.62%. Finally, Figure 1c shows the energy histograms derived from TBCNN-EB, resulting in the perfect value of FPR95=0.0%. This demonstrates the superior performance of the energy-bound learning model trained with the auxiliary dataset in detecting OOD samples.

4 DISCUSSION & CONCLUSION

We proposed to adapt the energy-bound loss as an alternative for the cross-entropy loss commonly used to train source code models. Along with the main dataset (in-distribution dataset), the energy-bounded loss is trained using an auxiliary dataset (out-of-distribution dataset). We showed that this training technique improves the robustness of source code models while preserving their predictive ability and the energy score produced from such models is used to detect code snippets that are out-of-distributions. The results of our evaluation on code classification task indicate that the energy-bound score is much better than the softmax score for the OOD detection task. ***It is therefore recommendable to consider this alternative in classification-based code learning.***

Our technique was evaluated solely on classification-based tasks, but many programming problems, such as code summarization and program translation, could be formulated differently. These two tasks are related to translation and generation tasks in general, and it is necessary to quantify the uncertainty of the translated results, i.e., the model should not produce incorrect translation results if

the output is uncertain. More effort is needed to propose methods for dealing with such translation- and generation-based tasks.

REFERENCES

[1] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. 2016. Learning local feature descriptors with triplets and shallow convolutional neural networks.. In *Bmvc*, Vol. 1. 3.

[2] Pavol Bielik and Martin Vechev. 2020. Adversarial Robustness for Code. *arXiv preprint arXiv:2002.04694* (2020).

[3] James R. Cordy and Chanchal K. Roy. 2011. The NiCad Clone Detector. In *The 19th IEEE International Conference on Program Comprehension, ICPC 2011, Kingston, ON, Canada, June 22-24, 2011*. IEEE Computer Society, 219–220. https://doi.org/10.1109/ICPC.2011.26

[4] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 3422–3426.

[5] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *40th ICSE*. 933–944.

[6] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2017. DeepAM: Migrate APIs with Multi-modal Sequence to Sequence Learning. In *International Joint Conference on Artificial Intelligence* (Melbourne, Australia). 3675–3681.

[7] Dan Hendrycks and Kevin Gimpel. 2016. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136* (2016).

[8] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *International Conference on Program Comprehension*. ACM, 200–210.

[9] Kisub Kim, Dongsun Kim, Tegawendé F Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. FaCoY: a code-to-code search engine. 946–957.

[10] Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. 2016. Learning local image descriptors with deep siamese and triplet convolutional networks by minimising global loss functions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5385–5394.

[11] Si Liu, Risheek Garrepalli, Thomas Dietterich, Alan Fern, and Dan Hendrycks. 2018. Open category detection with PAC guarantees. In *International Conference on Machine Learning*. PMLR, 3169–3178.

[12] Weitang Liu, Xiaoyun Wang, John D Owens, and Yixuan Li. 2020. Energy-based out-of-distribution detection. *arXiv preprint arXiv:2010.03759* (2020).

[13] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional neural networks over tree structures for programming language processing. In *AAAI Conference on Artificial Intelligence*.

[14] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. 2016. Convolutional Neural Networks over Tree Structures for Programming Language Processing. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 1287–1293.

[15] R. Nix and J. Zhang. 2017. Classification of Android apps and malware using deep neural networks. In *International Joint Conference on Neural Networks*. 1871–1878.

[16] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladmir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. 2021. Project CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks. *arXiv preprint arXiv:2105.12655* (2021).

[17] Md Rabin, Rafiqul Islam, Nghi DQ Bui, Yijun Yu, Lingxiao Jiang, and Mohammad Amin Alipour. 2020. On the Generalizability of Neural Program Analyzers with respect to Semantic-Preserving Program Transformations. *arXiv preprint arXiv:2008.01566* (2020).

[18] Md Rafiqul Islam Rabin, Ke Wang, and Mohammad Amin Alipour. 2019. Testing Neural Program Analyzers. In *34th IEEE/ACM International Conference on Automated Software Engineering (Late Breaking Research-Track)*.

[19] Goutham Ramakrishnan, Jordan Henkel, Zi Wang, Aws Albarghouthi, Somesh Jha, and Thomas Reps. 2020. Semantic Robustness of Models of Source Code. *arXiv preprint arXiv:2002.03043* (2020).

[20] Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on Source Code: A Neural Code Search. In *2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (Philadelphia, PA, USA). 31–41. https://doi.org/10.1145/3211346.3211353

[21] 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 *33rd ASE* (Montpellier, France). New York, NY, USA, 397–407. https://doi.org/10.1145/3238147.3238206

[22] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2020. A Systematic Literature Review on the Use of Deep Learning in Software Engineering Research. *arXiv preprint arXiv:2009.06520* (2020).

[23] Noam Yefet, Uri Alon, and Eran Yahav. 2020. Adversarial examples for models of code. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–30.

[24] Huangzhao Zhang, Zhuo Li, Ge Li, Lei Ma, Yang Liu, and Zhi Jin. 2020. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models. In *34th AAAI Conference on Artificial Intelligence*.

[25] Yaqin Zhou, Shangqing Liu, Jing Kai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 10197–10207. http://papers.nips.cc/paper/9209-devign-effective-vulnerability-identification-by-learning-comprehensive-program-semantics-via-graph-neural-networks