Condition-Guided Adversarial Generative Testing for Deep Learning Systems

Qiyin Dai, Pengcheng Zhang, Shunhui Ji
College of Computer and Information, Hohai University
qiyindai@qq.com, pchzhang@hhu.edu.cn, shunhuiji@163.com

***Abstract*—Over the past decade, Deep Neural Networks (DNNs) have achieved remarkable progress. However, the quality of such kind of systems is far from perfect. Software test is one of the most effective techniques for finding bugs in DNNs. Test case generation is the key factors of the success of software test. Existing test case generation approaches for DNNs always generate a large number of test cases, most of which do not meet the test requirements or the actual situation. In this paper, we propose CAGTest, a condition-guided adversarial generative testing tool for other DNNs to generate their test inputs to find potential defects. In general, CAGTest can generate test cases conditionally, which is not only efficient, but also does not produce a large number of invalid test cases and reduces the scale of test cases.**

***Keywords*—DNNs, Test Case Generation, Coverage Criteria, Conditional Adversarial Generative Network**

I. INTRODUCTION

Recently, Deep Neural Networks (DNNs) have been widely applied in various applications due to their high accuracy and efficiency, such as image recognition [1], natural language processing [2], malware detection [3], and self-driving cars [4]. However, as more and more safety-critical applications begin to adopt DNNs, deploying DNNs without comprehensive testing can have serious problems, such as accidents that may occur during automatic driving [5]. Consequently, it is very urgent to test DNNs-based systems efficiently and reliably.

Several researchers have already proposed different kind of testing approaches for DNN-based systems. Ma et al. [6] generalized neuron coverage in two ways to test DNNs. Sun et al. [7] introduced a family of metrics inspired by Modified Condition/ Decision Coverage. Xie et al. [8] proposed a coverage-guided fuzz testing framework for DNN software that systematically generates tests for detecting potential defects introduced during the DNN development and deployment phase. Experiments have shown that the approach can achieve good results in finding errors and evaluating the quality of the model. However, the approach utilizes a metamorphic mutation algorithm as core algorithm, the generated test cases are uncontrollable. Furthermore, the approach does not indicate whether the accuracy of the system is with respect to the coverage. As an improvement, condition-guided adversarial generative network was used to replace mutation algorithm. The new proposed approach called CAGTest has many advantages, which are shown as follows: *i)* During the input stage, we replaced the single input (i.e., an image) with a batch method, which can not only improve the efficiency of generating test cases, but also improve the quality of the generated cases. *ii)* We use a heuristic method to select a batch from the batch processing pool. This method can consider the time of batch selection and the time of batch joining the batch processing pool, so as to reflect the fairness and timeliness of selection. *iii)* More importantly, we use conditional adversarial generative network as the core of the framework to generate test cases. For CAGTest, condition c is used to control the direction of image transformation to positive aspects.

II. PRELIMINARIES

A. Coverage-Guided Fuzzing

Coverage-guided fuzzing [9] is used to find many serious errors in real software. The two most Coverage-guided fuzzing for general computer programs are libFuzzer [10] and AFL. These have been extended in various ways to make them faster or to increase the degree to which parts of the code can be targeted.

LibFuzzer relies on a corpus of sample inputs for the code under test. This corpus should ideally be seeded with a varied collection of valid and invalid inputs for the code under test. For example, for a graphics library the initial corpus might hold a variety of different small PNG/JPG/GIF files. The fuzzer generates random mutations based around the sample inputs in the current corpus. If a mutation triggers execution of a previously-uncovered path in the code under test, then that mutation is saved to the corpus for future variations.

B. Conditional Generative Adversarial Network

Generative Adversarial Networks. Generative adversarial networks (GANs) [11] were made up of two parts. GANs provide a framework to train a neural network, named the generator G, to generate data that belongs to a distribution (close to the real one) that underlies a target dataset. To train G, another neural network, called the discriminator D, is used. D's objective is to discriminate between real and generated samples.

Conditional GANs. For the past year, GAN-based conditional image generation has also been actively studied [12]. Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information c. c could be any kind of auxiliary information, such as class labels or data from other modalities. We can perform the conditioning by feeding condition c into

978-1-7281-0492-8/19/$31.00 ©2019 IEEE
DOI 10.1109/AITest.2019.000-5

the both the discriminator and generator as additional input layer.

In the generator, the prior input noise $P_z(z)$ and condition c are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator, generated case x and condition c are presented as inputs and to a discriminative function.

The objective function of a two-player minimax game would be as follows:

$$\min_G \max_D V(D,G) = E_{x\sim P_{data}(x)}[logD(x|c)] + E_{z\sim P_z(z)}[log(1-D(G(z|c)))] \quad (1)$$

III. METHODOLOGY

Fig. 1 gives the overview of CAGTest. Our testing framework consists of three major components: *data preprocessing*, *cases generation* and *DNN feedback*. We describe in more detail the components of the three parts in the following.

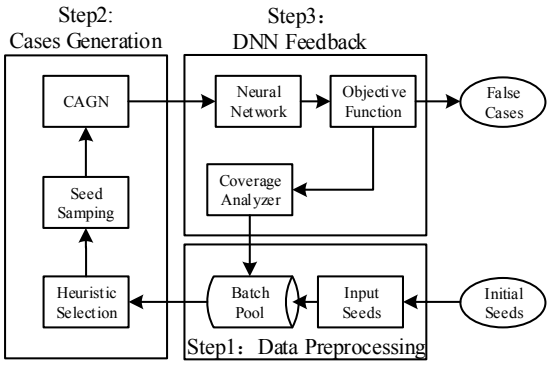

Fig. 1: Coarse descriptions of the main CAGTest loop

Data Preprocessing: In order to improve efficiency, we use batch input instead of single input. Given a DNN model under the test and its required seeds set (e.g., image sets), the set was taken as the initial seeds set and randomly assigned to a certain number of equal subsets. Each subset was added to the batch processing pool as a batch, and time t_i (the time when the batch is added to the batch pool) was set for each batch as the basis for heuristic selection.

Cases Generation: During the data processing phase, we have divided the initial seed set into batches. At any time, we must first select a batch from the existing batch pool. This requires an appropriate selection algorithm that does not miss the selection and does not select inhomogeneity. In our framework, we choose the heuristic algorithm as the selection algorithm. For the neural network we tested, making a uniform random selection worked acceptably well, but we ultimately settled on the following heuristic, which we found to be faster: $h(b_i,t) = \frac{e^{t_i-t}}{\sum e^{t_i-t}}$, where $h(b_i,t)$ gives the probability of choosing batch element b_i at time t and t_i is the time when element b_i was added to the batch pool. After the batch is selected, the framework samples some seeds from batch as input to CAGN to generate test cases. Put these sampled seeds into a vector, that is input, and each dimension is a seed record. Thus, CAGN can generate test cases with seed vector and condition vector C as inputs.

DNN Feedback: The goal of the feedback module is to maximize the network coverage of the test suite. The given DNN will predict all seeds (include initial seeds and the generated seeds) and collect the coverage information of the batch. For each batch of the generate seeds, the objective function will use similarities such as Euclidean distance to determine whether the generated test case is valid or not. If the generated test case is judged to be a failure, it is discarded; Otherwise, send it to Coverage Analyzer to determine whether to increase coverage. If the batch gains the coverage, it will be added into the batch pool.

IV. CONCLUSION AND FUTURE WORK

We present the idea of condition-guided adversarial generative testing framework for hunting potential defects of DNNs during development and deployment phase. In the future we will evaluate the usability and robustness of CAGTest through coverage increasing, bug finding, and accuracy increasing.

Acknowledgements. The work is supported by the National Natural Science Foundation of China (Nos. 61572171, 61702159), and the Natural Science Foundation of Jiangsu Province (No.BK20170893).

REFERENCES

[1] F. Clment, C. Camille, N. Laurent, and L. Yann, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[2] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[3] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in android malware detection," *Acm Sigcomm Computer Communication Review*, vol. 44, no. 4, pp. 371–372, 2014.

[4] P. Sermanet and Y. Lecun, "Traffic sign recognition with multi-scale convolutional networks," in *International Joint Conference on Neural Networks*, pp. 2809–2813, 2011.

[5] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning models," 2017.

[6] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu, *et al.*, "Deepgauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems," *arXiv preprint arXiv:1803.07519*, 2018.

[7] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," 2018.

[8] M. Aizatsky, K. Serebryany, O. Chang, A. Arya, and M. Whittaker, "Announcing oss-fuzz: Continuous fuzzing for open source software," *Google Testing Blog*, 2016.

[9] L. C. Infrastructure, "libfuzzer: a library for coverage-guided fuzz testing," 2017.

[10] C. Wang and S. Kang, "Adfl: An improved algorithm for american fuzzy lop in fuzz testing," in *International Conference on Cloud Computing and Security*, 2018.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[12] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.