2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S) | 978-1-6654-3566-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/DSN-S52858.2021.00021

Towards Automated Software Testing with Generative Adversarial Networks

Xiujing GUO
Graduate School of Advanced Science and Engineering
Hiroshima University
Hiroshima, Japan

Abstract—This paper discusses software testing methods based on Generative Adversarial Network (GAN). GAN is a generative model that can create new data instances that resemble training data. A GAN consists of a generator network and a discriminator network. In our testing scheme, the trained generator network is used as a test case generator. In addition, we propose a framework with GAN, which is a testing strategy used to increase the test coverage.

Index Terms—software testing, test case generation, test input, execution path, GAN

I. Introduction

Software testing is one of the most important activities to build a reliable software system in which the program is generated under specified conditions to find errors and evaluate software quality. During the process of dynamic testing, software reliability is improved as the number of test cases increases, and the cost of testing depends on the number of test cases. Therefore, how to reduce the number of test cases without the degradation of reliability is a challenge in software testing.

In this paper, we focus on the AI-based automatic test case generation. The most popular AI-based automatic test case generation approach is search-based software testing (SBST). SBST uses symbolic execution to extract branch conditions and uses optimization algorithm such as simulated annealing to find a test input that satisfies the branch conditions. However, with the increasing scale and complexity of the software system, it is complicated and time-consuming to carry out large-scale branch coverage manually. This paper proposes a software testing framework with GAN, which is used to generate test cases for the software under test that can increase the test coverage.

II. GANs: Generative Adversarial Networks

GANs are algorithmic architectures that use two neural networks, pitting one against the other (thus the "adversarial") in order to learn the underlying distribution of the training data so that it can generate new data instances that resemble training data. GAN consists of a generator and a discriminator. The generator takes random noise z and generates fake data. The discriminator determines if the generated data is real or fake. GAN estimates generative models via an adversarial process. In the adversarial process, the generator aims to maximize the failure rate of the discriminator while the discriminator aims to minimize it [1].

One of the most challenging problems when AI technologies are applied to unstructured data such as program and testing is to determine what data are used as inputs of the AI system. Our main idea is to use not only the test inputs but also the corresponding execution path as inputs of GAN. A pair of test inputs and program execution path includes rich information on the program itself. Also, it is not difficult to observe and collect the execution path for an input in the dynamic testing and it is more reasonable in terms of cost than symbolic execution used in the SBST.

The execution path in our work is extracted by Gcov tool and is defined as the combination of the execution of each branch in the program under a specific input. Fig. 1 illustrates the input and output of GAN in our framework. In our scheme, the trained discriminator of GAN can accurately judge whether a path corresponding to a specific input is correct or not. The trained generator can generate a test input and its corresponding execution path. Therefore, the generator is directly used as an automatic test case generator. On the other hand, the discriminator is recognized as an execution path simulator for software under test.

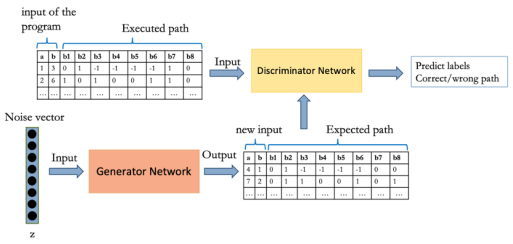

Fig. 1. The input and output of GAN in our framework. ("b1=0" represent branch1 is not taken; "b2=1" represent branch2 is taken at least once; "b3=-1" represent branch3 is not executed.

III. Software Testing with GAN: a testing strategy to increase the coverage

Fig. 2 illustrates the structure of the framework, which is used to generate test cases that can improve branch coverage. It contains four steps:

978-1-6654-3566-6/21/$31.00 ©2021 IEEE
DOI 10.1109/DSN-S52858.2021.00021

Step 1: Select m test inputs and their corresponding paths as training data.
Step 2: Train GAN with training data to generate n data.
Step 3: According to the path information of the generated data, select w(w<n) inputs data from n data sets that may cover unexecuted branches in the training data.
Step 4: Add the selected w input data and their corresponding executed paths to the training data, go to Step2.

Fig. 3 illustrates an example of the framework. In the training data, branch 2 and branch 4 are not taken. Therefore, we select the inputs "a=12, b=3; … ; a=2, b=9; a=2, b=8; a=2, b=6" which their generated path information shows that branch 2 and branch 4 are taken. Then execute program with selected inputs to extract real path, add them into training data. With a high-precision GAN model, the framework has the ability to improve test coverage.

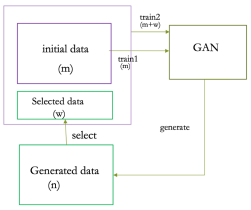

Fig. 2. Structure of the framework for increasing test coverage

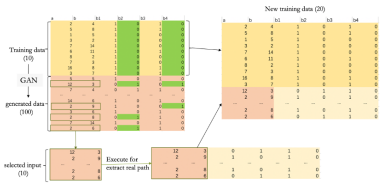

Fig. 3. An example of framework

IV. Experiment

In the past few years, different variants of GAN have been proposed. In this experiment, we choose WGAN-GP to apply in our framework, because WGAN-GP is the most advanced GAN model with the highest training stability. We apply our proposed framework to two software testing levels and conduct two sets of experiments: unit testing and integration testing. The test objects are two modules in the GNU Scientific Library: gamma_inc.c module (contains 13 functions with branches) and hyperg_1F1.c module (contains 21 functions with branches). In the unit testing, the GAN model generates paths in function units. We experimented with three functions in gamma_inc.c module and six functions in hyperg_1F1.c module. In the integration testing, the GAN model generates path information for all functions at once.

In both two experiments, the parameters of framework are set to m=100, n=100, w=10, and iterated 10 times to generate 100 test data. The 100 initial data in the integration testing are randomly generated, and the initial data of each function in unit testing is the selected data that can pass the objective function. We compared the test coverage of the data generated by the above two experiments and the accuracy of the generated path to verify whether the framework can generate data to improve the test coverage and what level of testing is the framework suitable for. We also conducted a random test to generate 200 test data to compare the test coverage with the GAN method.

Table I shows the results of integration testing and unit testing under WGAN-GP model and the random test's test coverage. In order to facilitate comparison, we divide the data generated in the integration testing into function units to compare with the unit testing results. As can be seen, the path prediction accuracy of WGAN-GP in unit testing is higher than that in integration testing. And the coverage of test cases generated by WGAN-GP in unit testing is higher than that of randomly generated test cases. Therefore, the framework has the ability to generate test cases that can improve test coverage, and it is more effective in unit test scheme than integration test scheme.

TABLE I
Comparison of the coverage achievement and the accuracy of path

		No. of branches	methods	WGAN-GP			RAND
				Initial coverage	Trained coverage	Average path accuracy	Random coverage
hyperg_1F1.c	All	594	Integration	24.07%	51.42%	44.82%	45.12%
	Function 1	34	Integration	47.05%	58.82%	37.78%	64.70%
			Unit	38.23%	50.00%	86.13%	
	Function 2	62	Integration	46.77%	67.74%	42.27%	75.80%
			Unit	46.77%	88.70%	82.77%	
	Function 3	72	Integration	41.66%	83.33%	38.10%	79.16%
			Unit	79.16%	94.44%	73.03%	
	Function 4	54	Integration	40.74%	72.22%	27.96%	83.33%
			Unit	61.11%	90.74%	79.00%	
	Function5	38	Integration	63.15%	76.31%	41.54%	84.21%
			Unit	60.52%	84.21%	84.36%	
gamma_inc.c	All	138	Integration	49.28%	52.17%	28.20%	50.00%
	Function 1	24	Integration	45.83%	54.16%	17.91%	66.66%
			Unit	58.33%	75.00%	76.42%	
	Function 2	20	Integration	55.00%	65.00%	23.35%	75.00%
			Unit	60.00%	80.00%	64.35%	
	Function 3	15	Integration	33.33%	33.33%	36.43%	20%
			Unit	66.66%	80.00%	82.81%	

V. Concluding Remark

In this paper, we propose a GAN framework, which is a testing strategy that uses GAN to automatically generate test cases to increase test coverage. We have verified that the framework is more effective in the unit test scheme. Therefore, in the future, we will apply our framework into unit testing and consider performing integration testing based on unit testing. We plan to divide the path prediction into two steps based on the program's structure: firstly, predict function calls based on input values, then predict the execution of each branch contained in the called function based on the input values.

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In Proceedings of NIPS, pages 2672– 2680, 2014.