Guided Test Case Generation Through AI Enabled Output Space Exploration

Christof Budnik
Siemens Corporate Technology
Princeton, New Jersey
christof.budnik@siemens.com

Marco Gario
Siemens Corporate Technology
Princeton, New Jersey
marco.gario@siemens.com

Georgi Markov
Siemens Corporate Technology
Princeton, New Jersey
georgi.markov@siemens.com

Zhu Wang
Siemens Corporate Technology
Princeton, New Jersey
zhu.wang@siemens.com

ABSTRACT

Black-box software testing is a crucial part of quality assurance for industrial products. To verify the reliable behavior of software intensive systems, testing needs to ensure that the system produces the correct outputs from a variety of inputs. Even more critical, it needs to ensure that unexpected corner cases are tested. Existing approaches attempt to address this problem by the generation of input data to known outputs based on the domain knowledge of an expert. Such input space exploration, however, does not guarantee an adequate coverage of the output space as the test input data generation is done independently of the system output. The paper discusses a novel test case generation approach enabled by neural networks which promises higher probability of exposing system faults by systematically exploring the output space of the system under test. As such, the approach potentially improves the defect detection capability by identifying gaps in the test suite of uncovered system outputs. These gaps are closed by automatically determining inputs that lead to specific outputs by performing backward reasoning on an artificial neural network. The approach is demonstrated on an industrial train control system.

KEYWORDS

Black-box Testing, Artificial Intelligence, Neural Networks, Test Design, Test Oracle, Adversarial Examples

ACM Reference Format:
Christof Budnik, Marco Gario, Georgi Markov, and Zhu Wang. 2018. Guided Test Case Generation Through AI Enabled Output Space Exploration. In *AST'18: AST'18:IEEE/ACM 13th International Workshop on Automation of Software Test , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3194733.3194740

AST'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5743-2/18/05…$15.00
https://doi.org/10.1145/3194733.3194740

1 MOTIVATION

The success of industrial systems is driven by low maintenance costs. A major cost factor in software maintenance is the repair of failures that are revealed while the product is in operation. Dynamic testing has been sufficient for traditional software to detect defects but comes short for complex industrial systems that control physical devices. The main reason is the incomparably larger input space of such systems, which makes testing under all conditions impractical and sometimes even impossible. We propose a new test method, based on neural networks, that can be used to derive unique test inputs for system outputs which have not been explored.

Black-box testing is predominant in industry where no structural information of the system under test (SUT) is available such as products that consist of legacy code and third-party components. Test data represented by input/output relations plays a crucial role in the design of black-box test cases. This data describes the initial input conditions of a test and the influenced output behavior of the SUT. Test input conditions are generated either at random or through some form of systematic guidance given by a fault ontology or a coverage metric of the specification. The nature of such input space exploration, however, makes both the probability and the time frame for discovering relevant faults highly non-deterministic. More importantly, the coverage degree of the input space does not correlate to a similar coverage of the output space.

The novel contribution of this paper is two-fold. First, we define a black-box testing approach to systematically explore the output space and to derive new test inputs with the goal of increasing the output space coverage. Exploring the previous uncovered input/output pairs will potentially expose system failure. The approach enables the exploration of the output space through multiple strategies. In this paper, we focus on two: output categorization and functional continuity. Second, we demonstrate how this problem can be solved by leveraging artificial Neural Network (NN) and adversarial example generation. In particular, the NN is trained to represent a good approximation of the SUT, and then used to perform backward reasoning to identify new "interesting" input/output combinations.

The approach is applied to a brake distance calculation function of a train control system which monitors the motion and the operator's control decisions to safely stop the train.

The reminder of the paper is organized as follows: Section 2 discusses the exploration of the output space followed by the proposed test generation approach in Section 3. A preliminary case study shows the feasibility of our solution in Section 4. Finally, Section 5 discusses related approaches before Section 6 concludes the work and gives an outlook on future work.

2 GUIDED OUTPUT SPACE EXPLORATION

Given a system under test $SUT : X \rightarrow Y$, and an output value $y \in Y$, the *output space exploration* problem is the problem of identifying an input value $x \in X$ s.t. $SUT(x) = y$.

The choice of the new output value y can be driven by multiple considerations, and typically requires the involvement of a domain expert to identify key insights into valid input and output ranges (i.e., the definition of X and Y), and requirements that the SUT is expected to satisfy. In particular, while a proper characterization of the input domain (X) might be known, it is typically less common for a characterization of the output domain (Y) to be available (e.g., $Y \subseteq [5, 10]$). However, a proper characterization of both the input and output space is needed to generate meaningful test cases.

During our investigation, we identified two useful strategies for guided output space exploration: *output categorization* and *functional continuity*.

Output Categorization. Systematic coverage criteria like category partition method and boundary value analysis are typically used for input space coverage. In output categorization, we apply the same concepts to the output space. The choice of interesting and relevant y candidates is aimed at identifying desirable, but non-covered or weakly covered output space areas. This activity can be supported by a visual distribution plot as shown in Figure 1. These plots can be generated using existing test-cases, and be examined and compared to identified coverage goals.

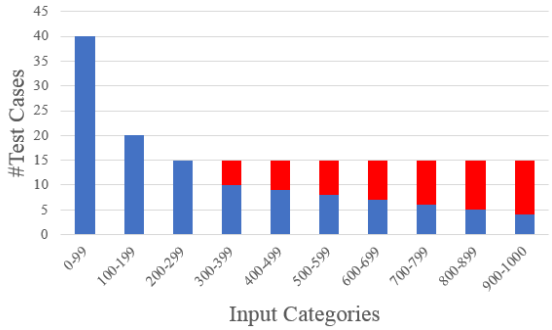

Figure 1: Output Categories: Comparing actual vs expected output distribution

An example of a simple categorization for the output consists in defining a uniform distribution of the output values, and randomly sample y from this distribution. This provides a simple random test approach that can be fully automated.

Functional Continuity. Most functions related to cyber-physical systems are continuous [4]. This implies that small perturbations of the input space are expected to lead to small changes in the output space. Therefore, during our exploration, we aim at identifying a small change in the input $x' = x + \delta$ that would lead to a big change in the output $y' = y + \Delta$.

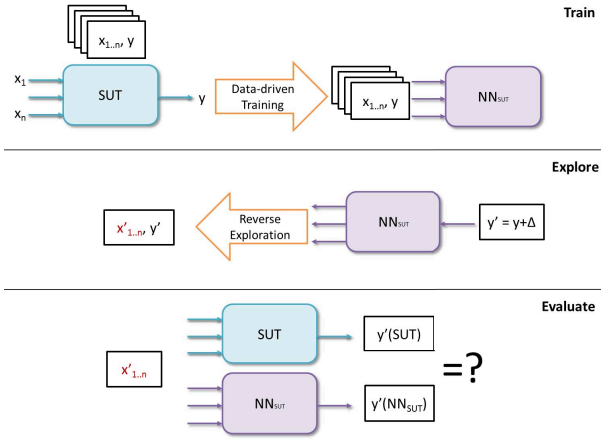

Figure 2: Output Space Exploration via Neural Network

3 TEST CASE GENERATION THROUGH NEURAL NETWORKS

The challenge of performing output space exploration in a black box setting is the lack of a model of the system that can be used for reasoning. We overcome this limitation by using a neural network as an approximation of the target system. Once the NN represents a good approximation of the SUT, we use the NN to generate new test-cases. In particular, we perform backward reasoning on NN to identify new input/output pair of values, according to one of the strategies defined in the previous section. As illustrated in Figure 2, we distinguish three main phases: *Train*, *Explore*, and *Evaluate*.

Train Phase. In the *Train* phase, we create the neural network NN_{SUT} with data sets of input/output vectors describing test cases of the SUT. The data sets can be extracted from existing scenarios (such as reference test cases or operational log files), or via random generation of new inputs and evaluation on the actual SUT. The neural network NN_{SUT} learns the input/output relationships from the data sets. The phase is completed when it can approximate the existing data with a high level of accuracy. The actual stopping criteria depends on the availability of data, but it is important to notice that even with limited accuracy, the network can be used in the following steps to perform output space exploration.

Explore Phase. In the *Explore* phase, we pick a new output value y' (by applying one of the output space exploration strategies) and explore our neural network to obtain an input vector x', such that giving x' to NN_{SUT} yields y' (i.e., $NN_{SUT}(x') = y'$). The exploration of the neural network uses the same back-propagation and projected gradient descent techniques used for the learning step, however, the model is fix (weights and biases will not change), and only the input vector x is free to change. Further, for a given y', we constrain the adversarial input x' by some context-dependent distance metric used to quantify similarity, so that the new input is close to an existing one (i.e., $\|x - x'\| \leq \epsilon$). This techniques is

called an *adversarial sample generation*. The general idea of adversarial examples is to applying small perturbation to an example from the data set in order to generate a new input that yields a desired output, most often with the goal of intentionally producing incorrect answers with high degree of confidence by exploring the linearity of neural networks [3]. We use an adaptation of the *Limited-memory Broyden-Fletcher-Goldfarb-Shannon* (L-BFGS) algorithm, which models the problems as a constrained minimization of the standard Euclidean distance between x and x'. The adaptation, also referred to as the Fast Gradient Sign Method (FGSM) optimizes the L-BFGS algorithm for speed rather than trying to produce an adversarial example very close to the target.

Evaluate Phase. In the *Evaluate* phase, we use the new input value x', derived from the modified output y', to define a new test case and execute it against the *SUT*. We, then, compare the execution results from the *SUT* with the expected output y'. If the two outputs differ significantly (i.e., $\|NN_{SUT}(x') - SUT(x')\| > \epsilon$), the cause can have two potential meanings:

- The outcome is a correct behavior but has not been learned by the *NN*. This is the most likely scenario, and requires repeating the Train phase with additional data points (including the x' and y').
- The outcome represents a failure in the *SUT*. In this case the failure can be analyzed and fixed.

If the two outputs are similar (i.e., $\|NN_{SUT}(x') - SUT(x')\| < \epsilon$), then we have identified an input vector x' that produces the expected output, and therefore increases the output space coverage. In this case we add the valid test to our set of test cases and repeat the process at the *Explore* phase defining a new output y'. Figure 3 depicts the sequence of activities to be followed based on the evaluation outcome as described above.

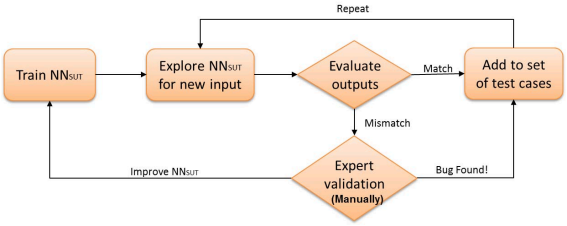

Figure 3: The method process

4 PRELIMINARY CASE STUDY AND RESULTS

In order to demonstrate the plausibility of the presented method, we conducted a preliminary case study. The case study builds on our previous work in the railway domain, and focuses on the evaluation of a train controller for an automatic train protection (ATP) system. The ATP is designed to monitor and control train movements with the goal to increase safety, reduce infrastructure utilization and increase operational efficiency by automatically stopping the train in cases where the train engineer fails to act according to protocol. For the purpose of our case study we chose the crucial implementation of the braking distance calculation algorithm (whose formal prove has been shown in a previous contribution [8]) which needs to ensure the safety (collision freedom) and efficiency (no unnecessary or early stops) of the overall system. The reference implementation (from [8]) considers ten dependent input variables like speed, weight, and maximum acceleration, and outputs a stopping distance in feet. The goal of the case study was to apply our AI Enabled Output Space Exploration approach to find new input vectors to cover unexplored areas of the output space.

Setup and Conduction. The first step of the analysis is to obtain and prepare the data to train the *NN*. We used our reference implementation to generate a dataset of 2000 observations randomly and used a 80:20 training/validation split. This was sufficient to validate the neural network's accuracy to be more than 99%. However the required amount and quality (in terms of perturbation) of data will likely vary when applying the approach to other problems. Next, we used the TensorFlow [1] framework to defined a standard 3-layer Neural Network regressor, with a single fully-connected hidden-layer. The *Mean Squared Error* metric, which adds a Sum-of-Squares loss to the training procedure was used as loss metric, and we trained the network using *Adaptive Moment Estimation (Adam)* optimizer, which is a good standard choice for many problems, as it uses a combination of mechanisms like adaptive learning rates and decaying gradients to speed-up learning. Once defined, the neural network was trained until the accuracy of the predictions on the validation set exceeded 99% after 2000 epochs with an average error of only 0.72 ft, which in our case was considered a good approximation of the SUT.

The next step focused on the synthesis of the adversarial example. In our case, given a new breaking distance $y\prime$, we try to find an adversarial input $x\prime$.

Due to the nature of the technique we chose for the generation of the adversarials, it is unlikely that we obtain inputs that exactly match the desired output. However, the results are typically sufficiently close as to be useful. In our particular case study this translated to a standard deviation of 9.0142 ft. on overall 777 generated $y\prime$ values in the range $(0, 800]$. In doing so, we ensured that all identified output space categories have a minimum of 250 randomly distributed data points by completely ignoring areas where the coverage already exceeds our target, and focusing on areas with insufficient coverage.

The last step in our method involved the use of the newly generated input vectors against the SUT, analyzing the results and determining if further training is necessary or if some of the obtained results indicate potential problems.

Discussion of the Results. The results of the case study demonstrates the effectiveness of our proposed approach. We randomly generated 2000 test cases which are well-distributed in the input space, to test the use case. The resulting output space coverage distribution is shown in Figure 4. The blue dots indicate coverage by an existing test case in each respective category: $(0, 100)$, $(100, 200)$,. . . , $(700 - 800)$. Intervals with few blue dots indicate areas with weak coverage. For instance, the coverage of the Braking Distance between $700 - 800$ ft. is less than 2%. The target output space coverage, after the application of our method is shown in Figure 4. Red dots in the picture illustrate the random output we generated to increase the coverage of the weak covered areas to a minimum of 250 data points, such as randomly generating Braking Distance ($y\prime$) in the range of $[700, 800]$.

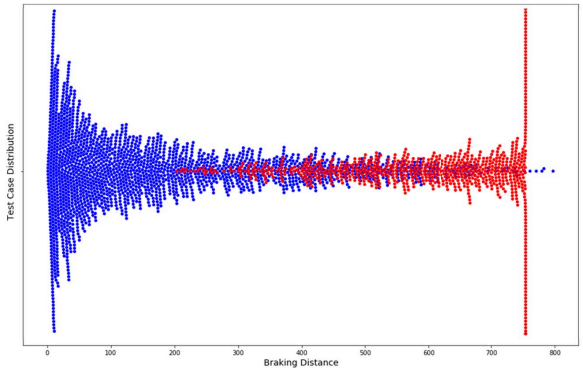

Figure 4: Original (blue) and new test case distribution (red)

The results are encouraging, in that, while acknowledging the limitations of a single case study, they demonstrate that the framework enables the identification of new test inputs that provide increased coverage of the output space.

Additionally, it is important to note that while in this particular case the initial neural network design was sufficient to achieve performance of >99% accuracy, when applying this approach to other kinds of problems it might be necessary to change the network architecture, perform hyper-parameter tuning, optimization, and regularization in order to improve the neural network's performance and increase its accuracy.

5 RELATED WORK

Black-box testing becomes dominant when the source code of the SUT is not available as well as when the system behavior is too complex to be represented by reusable design artifacts for testing. Traditional black-box testing techniques help to automatically generate test data without the knowledge of the software internals. The algorithm designed to create test cases often relies on a divide and conquers approaches. These are test techniques such as equivalence class testing or combinatorial test case generation which have been established in industry [7]. While such techniques are efficient at providing solutions to the coverage of combinatorial permutations over test input sets or configuration set-ups, they struggle to cover all outputs of the system as relation to the test inputs. Other test generation and reduction techniques are based on the discovery of the input/output (In/Out) relationship of the system dynamically such as [6, 9]. These approaches are exclusively generating the expected results from the stimulated input data but do not guarantee to find all of the In/Out relationships and thus to explore the output space sufficiently. Systematic test techniques also require considerable time and resources for test case generation and execution. Random testing is an alternative approach which is more feasible. It allows to run large number of test cases due to its simplicity and cost effectiveness. The empirical comparison shows that random testing and partition testing are equally effective [5]. Adaptive Random Testing (ART) aims to randomly select test cases, but also to spread them evenly. The intuition is that new test cases that are located away from the test cases that reveal no failure are more likely to find failures. ART was proposed by Chan et al. [2] to improve the fault detection capability of random testing. ART exploits information on executed test cases, particularly successful test cases, to guide the random test case generation. Because failure-causing inputs are often clustered together to form one or more contiguous regions, subsequent test cases that are close (similar) to successful test cases are less likely to hit the failure-causing region than those that are far away. Hence, ART promotes diversity among random test cases but also fails to explore the output space sufficiently.

6 CONCLUSION & OUTLOOK

This paper presents a novel approach for functional black-box testing focused on the exploration of the output space of the system under test. A NN is used to create an approximation of SUT, and then explored for generating input test data that improves the coverage of the output space according to various strategies. The case study demonstrates the feasibility of the approach, and the potential of the overall idea. A more exhaustive experimental evaluation of the approach is under way, where we artificially seed bugs in the SUT and evaluate the detection rate of the output space exploration approach compared to other techniques, and across strategies (e.g., comparing functional continuity with output categorization). The initial experiments have shown that the success of the approach is largely dependent on the neural network architecture, hyper-parameters used during training, and the the quantity and quality of data, where the quality of data refers to the distribution and perturbation of the test data in order to cover key areas of the SUT behavior. Guidelines for addressing these concerns need to be developed for simplifying the application of our techniques. The general technique of NN enabled output exploration is applicable to a wide range of systems. The challenge is to build a NN that can approximate with high-accuracy a chosen aspect of interest of the SUT. Hence, we started with an initial application of the approach to a system with continuous behavior and which is often stateless, but we are looking into options to extend it to more complex situations.

REFERENCES

[1] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.
[2] F. T. Chan, Tsong Yueh Chen, I. K. Mak, and Yuen-Tak Yu. 1996. Proportional sampling strategy: guidelines for software testing practitioners. *Information & Software Technology* 38, 12 (1996), 775–782.
[3] I. J. Goodfellow, J. Shlens, and C. Szegedy. 2014. Explaining and Harnessing Adversarial Examples. *ArXiv e-prints* (Dec. 2014). arXiv:stat.ML/1412.6572
[4] Dick Hamlet. 2002. Continuity in Software Systems. *SIGSOFT Softw. Eng. Notes* 27, 4 (July 2002), 196–200.
[5] Dick Hamlet and Ross Taylor. 1990. Partition Testing Does Not Inspire Confidence (Program Testing). *IEEE Trans. Softw. Eng.* 16, 12 (Dec. 1990), 1402–1411. https://doi.org/10.1109/32.62448
[6] Patrick J. Schroeder, Pat Faherty, and Bogdan Korel. 2002. Generating expected results for automated black-box testing. In *IEEE Trans. Autom. Sci. Engin.* 139–148.
[7] D. Richard Kuhn, Raghu N. Kacker, and Yu Lei. 2010. *SP 800-142. Practical Combinatorial Testing.* Technical Report. Gaithersburg, MD, United States.
[8] Stefan Mitsch, Marco Gario, Christof J. Budnik, Michael Golm, and André Platzer. 2017. Formal Verification of Train Control with Air Pressure Brakes. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*, Alessandro Fantechi, Thierry Lecomte, and Alexander Romanovsky (Eds.). Springer International Publishing, Cham, 173–191.
[9] Patrick J. Schroeder and Bogdan Korel. 2000. Black-box Test Reduction Using Input-output Analysis. *SIGSOFT Softw. Eng. Notes* 25, 5 (Aug. 2000), 173–177.