Reliability Evaluation of ML systems, the oracle problem

Antonio Guerriero
DIETI - Università degli Studi di Napoli Federico II
Napoli, Italy
antonio.guerriero@unina.it

***Abstract*—The growing adoption of machine learning (ML) in safety-critical contexts makes reliability evaluation of ML systems a crucial task. Although testing represents one of the most used practices to evaluate the reliability of "traditional" systems, just few techniques can be used to evaluate ML-systems' reliability due to the *oracle problem*. In this paper, I present a *test oracle surrogate* able to automatically classify tests' outcome to obtain feedback about tests whose expected output is unknown. For this purpose, various sources of knowledge are considered to evaluate the outcome of each test. The aim is to exploit this *test oracle surrogate* to apply classical testing strategies to perform reliability assessment of ML systems. Some preliminary experiments have been performed considering a Convolutional Neural Network (CNN) and exploiting the well known MNIST dataset. These results promise that the presented technique can be effectively used to evaluate the reliability of ML systems.**

***Index Terms*—Testing, Machine Learning, Test Oracle problem**

I. Introduction

The increasing trend in adopting Machine Learning (ML) solutions in safety-critical context makes the *reliability* evaluation of these systems a great concern. In software reliability engineering [1], *testing* represents one of the most used solutions, but it is challenging to apply it. Murphy *et al.* define ML systems as "non-testable", because *there is no reliable "test oracle" to indicate what the correct output should be for arbitrary input* [2].

System reliability is commonly defined as the probability of failure-free operation under specified conditions for a specified time [3]. As ML systems offer a response on demand, their reliability can be computed using a pragmatic metric, namely as a percentage of correct predictions. In the classification domain, a misclassification is a failure: given a sample, the ML system labels it with a class different from the expected one. The training phase is an important step of the paradigm adopted to define ML systems. In particular, at the end of this phase, the generalization error is computed to forecast the accuracy of the system under test in operation. This estimate could be not very accurate, in particular, if operational data are very different from the ones considered during the training phase. For a tester, it is difficult to estimate the reliability of an ML system considering only operational data, because the expected output is unknown for an arbitrary input.

For traditional software systems, testers exploit various system-specific characteristics to implement test oracles. For instance, an unhandled exception generated by a test represents an undesired system behavior and may thus be considered a failure. Likewise, in distributed systems, when a request is sent, the lack of a response may indicate a failure. Similarly, for ML systems, each time the behavior of the system differs from the one encoded in the training set, it can be considered a failure.

In this paper, I present TOS, a *test oracle surrogate* able to automatically detect failed tests based on a set of rules representing the expected behavior of the system under test so that each time a rule is violated a failure is said to have occurred.

The definition of the test oracle surrogate for ML-systems consists of three main steps:

- Listing of both execution domain and testing environment information as rules.
- Encoding of the expected behavior of the system under test (represented by the training set) as rules.
- Exploiting specific ML algorithm's features to evaluate the output.

Preliminary experimentation is reported, considering the MNIST dataset and a Convolutional Neural Network (CNN). The reliability values estimated by TOS are compared to a Cross-Referencing Oracle (CRO), as considered by Srisakaokul *et al.* [4] for multiple implementation testing.

The preliminary results point out that both approaches influence the reliability estimate in different ways: TOS is more prone to underestimate the reliability; CRO is more prone to overestimate it.

II. Related Work

Reliability assessment of ML systems represents a growing trend in current research. Li *et al.* propose a technique for operational testing of Deep Neural Networks (DNNs) to estimate the reliability considering unlabeled data [5]. The solution proposed by the authors consists of selecting a set of representative samples to estimate the reliability of the DNN under test. These samples have to be manually labeled, due to the absence of a test oracle for arbitrary input.

In the current literature, there are various solutions to address the oracle problem in ML system testing. In particular, Zhang *et al.*, in their survey on ML testing [6], highlight three main solutions: *mutation testing*, *metamorphic testing*, and *cross-referencing*.

978-1-7281-7735-9/20/$31.00 ©2020 IEEE
DOI 10.1109/ISSREW51248.2020.00050

Mutation testing [7] refers to generate new tests starting from already labeled ones, performing mutation without changing the semantic of the test case. Metamorphic testing [8], [9] consists of generating tests considering semantic-conservative mutations, due to the challenges in metamorphic relations definition [10], [11]. The impossibility to perform tests with unlabeled data makes these testing strategies not considerable to perform reliability assessment of ML systems as they are.

Cross-referencing [4], [12] refers to a test oracle detecting failed tests by observing whether similar applications yield different outputs regarding identical inputs. Srisakaokul *et al.* propose multiple-implementation testing to test supervised learning software [4]. They exploit cross-referencing to implement a majority oracle, which selects the most voted output running the test input of multiple implementations of the same algorithm (based on a predefined percentage threshold). Although the cross-reference oracle strategy is characterized by a high cost (multiple implementations of the same system are needed), and bugs in the training set are very difficult to detect (all the implementations might be affected in the same way), it represents a valid approach to evaluate the reliability of the system under test. In fact, operational data, with unknown expected output, can be evaluated.

Other techniques can be considered to address the oracle problem in reliability testing. In particular, Ma *et al.* [13] propose to exploit neural networks invariants in order to unveil adversarial examples. In particular, they show that for different inputs, different sets of neurons are activated; invariants may be mined looking for neurons activation patterns. I argue that if for a given ML system similar invariants can be mined, which are related to failures, they can be used as an additional inductive approach to automatically detect failed tests.

III. Proposal

A. ML-based systems

As stated in [14], *AI-based software and applications use machine learning models and techniques through large-scale data training to implement diverse artificial intelligent features and capabilities*. I regard an ML-based system (Figure 1) as taking a feature vector (f_i) as input; an internal component uses an ML algorithm to compute a response (r), while other components (c_i), not based on ML, produce additional outputs (a_i) which, combined with r, yield the ultimate output o. For instance, an autonomous driving system exploits an ML algorithm to process the cameras' images, while other components process on-board sensors' data, ultimately determining a final output (speed or steering angle). The behavior of an ML system is strongly dependent on both the ML algorithms and the training data.

B. Detection of failing tests

The architecture defined for a TOS is depicted in Figure 2. It is meant to exploit various ways to discover failed tests, using different sources of knowledge (domain knowledge, testing assumptions, training set, system's internal parameter

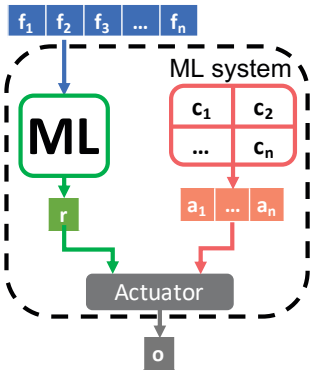

Fig. 1. ML system

values) to look for conditions, e.g. *invariants* [13], which hold when the system output can be judged incorrect, despite the correct output is unknown or uncertain. Consider, for instance, the mentioned autonomous driving system. Its output o, e.g. steering angle and speed, depends on the response r of the ML algorithm that processes the acquired images, and on the outputs (a_i) of on-board sensors. The architecture I envisage foresees three methods by means of which TOS can detect failed tests:

- *Deductive method*: it is based on rules defining both domain-related conditions that should never be violated (e.g., driving rules), and testing assumptions (proper of the chosen testing strategy). They can be derived from an expert, from an ontology, or via deductive processing of other rules. Each output violating such rules is judged as a failure.
  This level is characterized by a deterministic evaluation of the output, detecting a subset of failed tests with 100% accuracy, but incomplete (i.e., only a subset of failures is identified, depending on the set of rules being defined) – hence with false negatives with respect to the set of all possible failures, but with no false positives.
  An implementation of this method is called *Deductive TOS*. A deductive rule to detect failures for the example is "*if (the proximity alarm is ON* $\wedge$ *speed* $>$ 0.1 *Km/h)* $\Rightarrow$ *fail*". Moreover, assuming to apply partition testing, let me consider two partitions, the first one contains all samples with speed lower than 10 Km/h, and the second

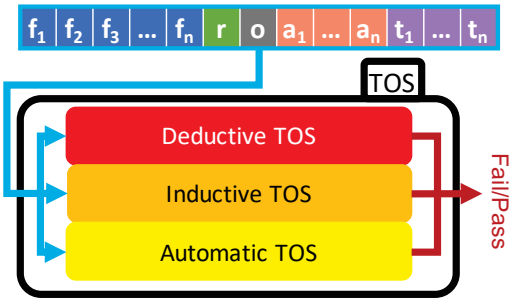

Fig. 2. Architecture of a TOS

one contains all samples with speed greater than 10 Km/h, an example of a rule is: "*if (the sample belongs to the first partition)* $\wedge$ $speed > 10$ *Km/h* $\Rightarrow$ *fail*".

- *Inductive method applied to training data*: a set of rules is inferred from the prior knowledge in the training set. The rules should be verifiable by the tester and able to identify a subset of failures preserving high accuracy (depending on the goodness of the training data). Adding these rules, the completeness can be significantly increased, depending on the representativeness of the training data. An implementation of this method is called *Inductive TOS*. An inductive rule for the example is "*if* $(pixel_{12,35} > 200 \wedge pixel_{14,65} < 56 \wedge speed > 75$ *Km/h)* $\Rightarrow$ *fail*".
- *Inductive method applied to the ML algorithm*: the aim is to mine patterns about how the ML algorithm produces an incorrect response. The assumption is that failures have similar patterns that are not available in the training set, and that are specific to the adopted algorithm. Hence I look for likely invariants in the ML algorithm behavior. This solution relies on the possibility to generate a dataset able to show failures of the system under test, to derive the invariants. Adding these invariants, the expectation is to improve the completeness of TOS detecting further different kinds of failures, potentially decreasing its accuracy (the number of false positives could increase).
  An implementation of this method is called *Automatic TOS*. An inductive rule for the example, assuming that the ML algorithm chosen is a Neural Network, is "*if (a certain subset of neurons has been activated* $\wedge$ $speed >$ 40 *Km/h)* $\Rightarrow$ *fail*".

IV. Experimentation

A. Experimental subject

The ML system considered for this experimentation is LeNet-5 [15], trained on $30,500$ samples ($28,000$ for the training set, and $2,500$ for the test set) of MNIST dataset of handwritten digits [16]. I considered a set of $39,500$ samples, assumed to be unlabeled, as test cases representing the operational environment. The labels removed are used to build a *Perfect Oracle* (PO).

The ML system under test is tested using a *partition testing* strategy. In particular, the test cases are split into two sets: digits lower than 5, and digits greater or equal to 5.

The selection of test cases is performed considering a *testing budget* (number of the executed test) of $20,000$ samples.

B. TOS implementation

The three methods of TOS are implemented for the experimentation as follows.

The *deductive method* is implemented considering a manual definition of two rules derived from the testing strategy:

- *if (input belongs to first partition)* $\wedge$ $label > 5 \Rightarrow$ *fail*;
- *if (input belongs to second partition)* $\wedge$ $label \leq 5 \Rightarrow$ *fail*.

The *inductive method* uses the **C4.5** algorithm [17] to extract the rules from the training set. Then, all the rules with a confidence lower than 0.99 are filtered out.

The *inductive method applied to the ML algorithm* is implemented exploiting Random Forest applied to a custom training set. In particular, the training set of Random Forest is built considering the probability vectors provided in output by the ML system corresponding to each of the $2,500$ instances of the test set. Each sample of the set is labeled as *fail* or *pass*, according to the expected label.

C. Research Questions (RQs)

The following research questions are here considered:

- **RQ1**: How accurate are the estimates obtained with TOS?
- **RQ2**: How does TOS perform compared to a CRO?

CRO is implemented as a majority oracle considering two additional CNNs. In particular, for an arbitrary input, the most voted output of the three CNNs is considered as the correct one; when the three CNNs diverge, assuming that the CNN under test is the most accurate, its output is considered as correct. Each experiment is repeated 5 times.

D. Evaluation metrics

The estimates of *reliability* (R) are computed exploiting the Nelson estimator [18]:

$$R = 1 - \frac{F}{T}, \tag{1}$$

where F is the number of failed tests, and T is the testing budget. The metric considered to answer RQ1 is the *offset%*, defined as the difference in percentage between the reliability value obtained considering PO and the one obtained with the evaluated oracle. The same metric is used to compare TOS to CRO in answering RQ2.

$$offset\% = \frac{R_{PO} - R}{100}. \tag{2}$$

V. Results

A. RQ1: Accuracy

Table I reports in columns 2 and 3 the values reliability R_{TOS} estimated with TOS, and the one R_{PO} obtained with the Perfect Oracle, for the 5 repetitions. The offset% (reported in the last column) between R_{PO} and R_{TOS} is 3.78% on average. The values of reliability obtained with TOS as test oracle are close to the ones obtained with PO, and they represent always an underestimate of the reliability, over the repetitions. This underestimate depends on the presence of false positives in the evaluation of tests outcome.

B. RQ2: Comparison to cross-referencing

Table II reports the reliability estimated exploiting the Cross-Referencing Oracle and the corresponding one with PO. The offset% between the two values is -2.58% on average. The higher estimate of reliability R_{CRO} depends on the presence of false negatives in the evaluation of test outcome.

Comparing Tables I and II, we observe that the reliability values obtained with CRO are closer to the estimates obtained

TABLE I
RELIABILITY ESTIMATES OBTAINED EVALUATING TESTS WITH TOS AND WITH A PERFECT ORACLE

Repetition	R_{TOS}	R_{PO}	*offset%*
1	0.922	0.960	*3.8*
2	0.924	0.960	*3.6*
3	0.920	0.960	*4.0*
4	0.926	0.962	*3.6*
5	0.922	0.961	*3.9*

TABLE II
RELIABILITY ESTIMATES OBTAINED EVALUATING TESTS WITH CRO AND WITH A PERFECT ORACLE

Repetition	R_{CRO}	R_{PO}	*offset%*
1	0.985	0.959	*-2.6*
2	0.986	0.959	*-2.7*
3	0.986	0.960	*-2.6*
4	0.986	0.960	*-2.6*
5	0.985	0.961	*-2.4*

with PO than the ones obtained by TOS. It is typically preferrable to have an underestimate of reliability than an overestimate, given a certain confidence. From this perspective, the values obtained with TOS are better than the ones with CRO.

VI. CONCLUSIONS

The oracle problem represents an important challenge when testing ML systems for reliability evaluation. In this paper, a test oracle surrogate is presented to obtain an automatic evaluation of tests to assess the reliability of the system under test. For this purpose, a three levels architecture is defined. Each level is characterized by a source of knowledge and a strategy to define rules. Although a coarse-grain tuning was performed to select the rules, the reliability estimates obtained with TOS are close to CRO ones (the difference is 0.011 on average). Moreover, the estimates obtained considering the proposed approach are always an underestimate of reliability, that represents a desired property in reliability evaluation. A fine-grain tuning of all parameters can provide better results.

The long-term objective of my research is to provide a technique to support testers assessing the reliability of ML systems. Currently, a tester has to front a strong effort to evaluate the performance of an ML system under test, due to the manual labeling of operational data. The purpose is to reduce this effort minimizing human intervention in test evaluation.

This my research work started during a visiting period at the the Chinese University of Hong Kong (CUHK), under the supervision of Prof. Michael R. Lyu, at the end of 2019. In my PhD thesis (whose defense is scheduled in the first months of 2022), I plan to define a more general and robust formulation of my Test Oracle Surrogate, supported by a deeper experimentation considering more datasets related to various domains (image classification, natural language processing, machine translation), and including comparison with other techniques to deal with the oracle problem in ML systems.

ACKNOWLEDGMENT

I thank my advisors, professor Stefano Russo and professor Roberto Pietranutono. Moreover, I thank professor Michael R. Lyu for his valuable hints on this research topic. This work has been supported by the department project COSMIC.

REFERENCES

[1] M. R. Lyu, Ed., *Handbook of software reliability engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[2] C. Murphy, G. E. Kaiser, and M. Arias, "An approach to software testing of machine learning applications," in *19th Int. Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2007, pp. 167–172.

[3] IEEE, "IEEE recommended practice on software reliability," *IEEE Std 1633-2016 (Revision of IEEE Std 1633-2008)*, pp. 1–261, 2017.

[4] S. Srisakaokul, Z. Wu, A. Astorga, O. Alebiosu, and T. Xie, "Multiple-implementation testing of supervised learning software," in *AAAI Workshops*. Association for the Advancement of Artificial Intelligence, 2018.

[5] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "Boosting Operational DNN Testing Efficiency through Conditioning," in *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 2019, pp. 499–509.

[6] J. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[7] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang, "DeepMutation: Mutation Testing of Deep Learning Systems," in *29th Int. Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 100–111.

[8] C. Murphy, G. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing." in *20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2008, pp. 867–872.

[9] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[10] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars," in *Proceedings of the 40th Int. Conference on Software Engineering (ICSE)*. ACM, 2018, pp. 303–314.

[11] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing," arXiv, 2018.

[12] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated Whitebox Testing of Deep Learning Systems," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*. ACM, 2017, pp. 1–18.

[13] S. Ma, Y. Liu, G. Tao, W.-C. Lee, and X. Zhang, "NIC: Detecting adversarial samples with neural network invariant checking," in *Network and Distributed System Security Symposium (NDSS)*, 2019.

[14] J. Gao, C. Tao, D. Jie, and S. Lu, "Invited paper: What is AI software testing? and why," in *Proc. IEEE Int. Conf. on Service-Oriented System Engineering (SOSE)*. IEEE, 2019, pp. 27–46.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[17] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[18] T. A. Thayer, M. Lipow, and E. C. Nelson, *Software Reliability*. North-Holland Publishing, TRW Series of Software Technology, Amsterdam, 1978.