Fault Localization for Reinforcement Learning

Jesús Morán
Department of Computing
University Of Oviedo
Gijón, Spain
moranjesus@uniovi.es

Antonia Bertolino
ISTI-CNR
Consiglio Nazionale delle Ricerche
Pisa, Italy
antonia.bertolino@isti.cnr.it

Claudio de la Riva
Department of Computing
University Of Oviedo
Gijón, Spain
claudio@uniovi.es

Javier Tuya
Department of Computing
University Of Oviedo
Gijón, Spain
tuya@uniovi.es

***Abstract*—Reinforcement Learning is widely adopted in industry to approach control tasks in intelligent way. The quality of these programs is important especially when they are used for critical tasks like autonomous driving. Testing and debugging these programs are complex because they behave autonomously without providing insights about the reasons of the decisions taken. Even these decisions could be wrong if they learned from faults. In this paper, we present the first approach to automatically locate faults in Reinforcement Learning programs. This approach called SBFL4RL analyses several executions to extract those internal states that commonly reduce the performance of the program when they are covered. Locating these states can help testers to understand a known fault, or even detect an unknown fault. SBFL4RL is validated in 2 case studies locating correctly an injected fault. Initial results suggest that the faults of reinforcement learning programs can be automatically located, and there is room for further research.**

Keywords—software testing, debugging, fault localization, reinforcement learning

I. Introduction

Reinforcement learning (*RL*) is a type of machine learning concerned to solve decision-making problems developing *agents* that learn from experience through trial-and-error decisions in an *environment*. Despite *RL* is the type of machine learning more searched in both *Google Scholar* and *Google*, the majority of research contributions in testing machine learning programs are focused on supervised and unsupervised learning [1]. According to the survey of Zhang et al. [1] more research is needed in testing *RL* programs.

Testing and debugging *RL* programs is challenging due to their opacity and stochasticity, among others. Some research works are focused on testing *RL* programs with mutation [2] and with adversarial evaluation [3]. Research is also active to explain humans why the *agent* makes one *action* [4], and Deshpande et al. [5] create a tool for visualizing how the *agent* performs. To the best of our knowledge, there is no previous research about locating faults in *RL* applications.

This paper proposes a fault localization approach to locate automatically which group of *states* is the root cause of faults -or low performance- for an *agent*.

II. Background

A. Reinforcement Learning

The *RL* problem can be modeled with *agent*, *states*, *actions*, and *rewards*. The *agent* observes the *state* from the *environment*, and based on that it makes an *action* that moves it to a new *state*. This transition of *states* takes 1 *timestep* and the *agent* receives a *reward* indicating good or bad behavior.

Each execution of the *agent* from the initial *state* until the end -if any- is called an *episode*, and the sequence of all *state-action-reward-state* of the *episode* forms a *trajectory*. The total *reward* obtained by the *agent* in the *trajectory* is called *return*, and can be calculated in different ways depending on the problem to solve. The simplest *return* is the *finite-horizon undiscounted return* that is the sum of all *rewards* from the *trajectory*. The *agent* is trained to learn which *action* is the best in a given *state* in order to maximize the *return*.

Consider a robotic cart called CartPole [8] that aims to balance a rigid pole hinged to it. The *actions* that the *agent* can take are moving the cart to the right or left. The *states* are the position and velocity of the cart, the angle of the pole, and the rate of change of the angle. The *rewards* are +1 per each *timestep* taken but the *episode* finishes when the angle passes ± 0.2 radians. During the training the *agent* only requires few *timesteps* to learn that bigger *returns* are obtained when its *actions* put the pole in vertical angle and with low velocity.

B. Spectrum-based Fault Localization (SBFL)

SBFL is a widely used technique for fault localization [6]. Given some test executions, *SBFL* obtains the so-called suspiciousness ranking that contains the most suspicious code lines to cause failures. This ranking is obtained calculating how much suspicious is each code line based on the number of executions that both fail/succeed and cover/uncover the line. Different metrics exist that calculate the suspiciousness of each code line in different ways. In general, the ranking metrics consider that the most suspicious lines are those that are usually both covered in the failed executions and not covered in the successful ones. Despite *SBFL* is commonly used to locate faulty code lines, it can also be used to locate other parts of the software that could cause failures.

III. SBFL for Reinforcement Learning (SBFL4RL)

This paper proposes an *SBFL* approach (SBFL4RL) that analyzes the *agent* executions with the goal to automatically locate those group of *states* in which the *agent* does not perform well or even fails. To this end, SBFL4RL considers that a group of *states* is more suspicious when the *episodes* cover it with low *returns* or do not cover with high *returns*. As Fig. 1 summarizes, SBFL4RL receives the *agent* and automatically obtains a ranking of the most suspicious group of *states*. First, the *agent* is tested several times to collect *trajectories*, from which both the group of *states* covered in each *episode* and their *returns* are obtained. Next, SBFL4RL uses the *returns* as test oracle to classify the *episodes* into two, those that perform well and those that do not. Finally, a

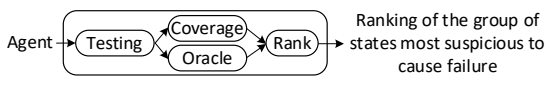

Fig. 1. SBFL approach for Reinforcement Learning applications

ranking metric obtains the suspiciousness ranking of group of *states* that are more suspicious to either cause failures or reduce the performance. Each phase is detailed below.

Testing: the *agent* is executed in an instrumented *environment* during several *episodes* to collect the *trajectories* of the *episodes*. The number of *episodes*, denoted as *E*, is determined by the tester and each one starts in a random *state*.

Coverage: SBFL4RL obtains per each *episode* the *group of states* covered and not covered. To this end, it groups the *states* with their neighborhood creating *G groups of states*, each one denoted as gs_i. The tester could vary the number of groups to obtain more/less fine-grain localization.

Oracle: SBFL4RL obtains per each *episode* the *finite-horizon undiscounted return*. Some *episodes* have higher *returns* and others lower. The tester wants to locate the *groups of states* that reduces the performance of the *agent*. Therefore, SBFL4RL divides the *episodes* in two depending on the *returns*: highest and lowest. This division is done with the percentile *P* established by the tester, i.e. if *P* = 50 means that half of the *episodes* are classified as lowest *return*.

Rank: the previous phases obtain per each *episode* (1) the *group of states* covered, and (2) if it has highest/lowest *returns*. Finally, SBFL4RL obtains a suspiciousness ranking that contains in the first positions the *groups of states* that are the root cause of the lowest *returns*. This ranking is obtained using Ochiai2 [7] that is a state-of-the-art ranking metric.

Suppose a faulty implementation of CartPole that receives wrong *rewards* when the angle of the pole is -0.02 radians receiving -25 as *reward*. During the training, the *agent* erroneously learns to avoid the pole in this angle. SBFL4RL instruments the *environment*, and the tester executes 100 *episodes* (*E* = 100) to collect their *trajectories*. The tester wants to analyze in which angles of the pole the *agent* does not perform well, and selects *G* = 20 for the angle. This means that the *sates* are grouped in ranges of 0.22 radians e.g. gs_9 is the interval of angles (-0.022, 0]. This *group of states* gs_9 is the faulty one because the *state* with -0.02 radians is in this group. Next, SBFL4RL obtains which *group of states* are covered in each *episode*, for example gs_9 is covered in 48 *episodes* and not covered in 51. The *returns* of the *episodes* vary between 10 and -68, and the tester classifies them in highest/lowest with percentile 50 (*P* = 50). In total, 51 *episodes* with *returns* between 10 and 8 are classified as the highest *returns* and the remainder as the lowest. Finally, the ranking metric Ochiai2 indicates correctly that the *group of states* most suspicious is gs_9 with a suspiciousness of 0.33.

IV. Case Studies

We evaluate SBFL4RL using two different *RL* programs as case studies: the already mentioned CartPole and LunarLander, a rocket trajectory optimization problem [8]. The *agents* are trained using the PPO algorithm until there is no improvement in 3 evaluations, each one after 1000 *timesteps*. For each *agent*, 100 *episodes* are executed (*E* = 100), and half of them are classified as highest *returns* (*P* = 50). The localization is done grouping the *states* in 20 groups (*G* = 20), in the case of CartPole we are focused on the angle of the pole, and in LunarLander in the horizontal position of the rocket. We have injected a fault in the programs to evaluate if SBFL4RL locates it: for both programs, the *reward* of gs_9 is mutated to -25. The fault localization is repeated 30 times, this means that 30 *agents* are trained, tested and debugged in each program. For each one, we check if SBFL4RL is able to locate the faulty *group of states* in the top of the suspiciousness ranking. Note that the ranking could have ties when at least two *group of states* have the same suspiciousness, but the tester can only analyze one at time and the ties must be broken. It is common to analyze the best, average and worst scenarios.

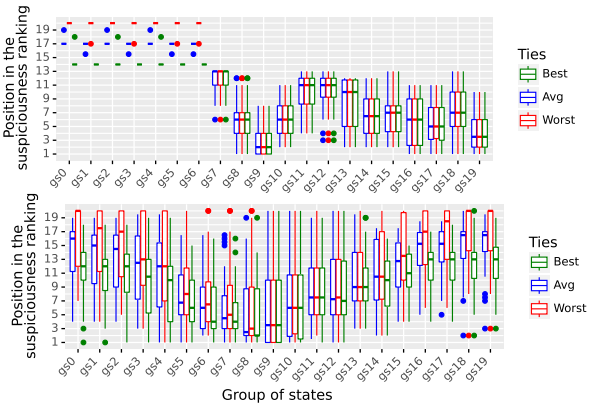

Fig. 2. Positions in suspciciousness ranking: CartPole and

Fig. 2 depicts the distribution of the position of each *group of states* in the suspiciousness ranking per each *RL* program, CartPole (top) and LunarLander (bottom). The X-axis is the *group of states*, Y-axis the position, and the color indicates the tie-breaking strategies. We can observe that SBFL4RL locates automatically the fault (gs_9) in the first positions of the ranking. SBFL4RL also indicates that the neighborhood *states* like gs_8 and gs_{10} are still suspicious. Note that these *states* are not faulty, but they are usually covered just before or after the faulty one. The ranking obtained by SBFL4RL gives hints about a possible fault around the *group of states* gs_8 - gs_{10}. Once the tester analyzes these *states*, it could detect a new unknown fault or understand better a known fault.

V. Conclusions and Future Work

Fault localization can be used to help tester during the testing/debugging of *Reinforcement Learning* applications. On one hand, it can help tester to understand better a known fault -or performance issue- providing the location of the *states* that cause low performance. On the other hand, it can help tester to discover new unknown faults because it automatically locates those *states* in which the *agent* does not work well. *RL* is one of the artificial intelligence areas that receive more attention in research, and there are opportunities to further research in testing/debugging these applications.

As future work, we plan to improve SBFL4RL to test/localize faults considering also which pairs of *state-action* are covered. Another improvement could be analyzing how much times each *state* is covered because SBFL4RL now only considers if it is covered or not. The same happen for the *returns* because SBFL4RL only considers two classes (lowest and highest *returns*), but analyzing the whole distribution of *returns* could improve the fault localization.

Acknowledgment

This work was supported by the project PID2019-105455GB-C32 funded by MCIN/AEI/10.13039/501100011033 (Spain), and the Italian MIUR PRIN 2017 Project: SISMA (Contract 201752ENYB), and ERDF funds.

References

[1] J. M. Zhang et al., "Machine Learning Testing: Survey, Landscapes and

Horizons," *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, Jan. 2022.
[2] Y. Lu, W. Sun, and M. Sun, "Towards mutation testing of Reinforcement Learning systems," *J. Syst. Archit.*, vol. 131, p. 102701.
[3] J. Uesato *et al.*, "Rigorous Agent Evaluation: An Adversarial Approach to Uncover Catastrophic Failures," *7th Int. Conf. ICLR 2019*,
[4] R. Dazeley et al. "Explainable Reinforcement Learning for Broad-XAI: A Conceptual Framework and Survey," arXiv:2108.09003, 2021.
[5] S. Deshpande, J. Schneider, "Vizarel: A system to help better understand rl agents," arXiv preprint arXiv:2007.05577, 2020.
[6] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A Survey on Software Fault Localization," *IEEE Trans. Softw. Eng.*, vol. 99, 2016.
[7] L. Naish et al., "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–32, 2011.
[8] G. Brockman *et al.*, "OpenAI Gym," arXiv:1606.01540 , Jun. 2016.