Which Factors Affect Q-Learning-based Automated Android Testing? – A Study Focusing on Algorithm, Learning Target, and Reward Function –

Yuki Moriguchi Shingo Takada

Keio University, Japan
{yukim, michigan}@doi.ics.keio.ac.jp

Abstract

With the spread of smartphones, the importance of automated testing of mobile applications has increased. However, many current approaches are inadequate, as they are not able to test functions that are available only on hard-to-reach GUI, which is a screen that can be reached only through a specific sequence of input events. To solve this problem, there has been an increase in testing research based on reinforcement learning, specifically Q-learning. Each research uses different learning targets and reward function. Testing research has also been done using Deep Q-Network, which extends reinforcement learning in a "deep" way. Although each work has conducted their own evaluation, it is not clear how the combination of learning algorithm, learning target, and reward function affects the result. To bridge this gap, we have conducted an empirical study comparing eight possible combinations. Our study found that the combination of Deep Q-Network as the learning algorithm, component as the learning target, and GUI change ratio as the reward function had the highest test quality in terms of code coverage.

Keywords: *Software testing, Q-learning, Android application*

1. Introduction

The proliferation of smartphones has led to an increase in mobile applications. As with conventional software, testing is an essential part of the development process of mobile applications. Thus, research on automated testing is being actively conducted. Unfortunately its performance is still not adequate. A major reason for this is that GUI screens which can be reached only through a specific sequence of input events are not easily reached. As a result, functions on those screens are often not tested.

In the past several years, reinforcement learning (specifically Q-learning), as well as a "deep" version of Q-learning (Deep Q-Network), has been applied to testing in order to overcome this issue [6] [4] [1] [5] [8] [9]. Reinforcement learning is a type of machine learning, where an agent adapts to the environment through trial and error. Unlike supervised learning, there is no teacher who explicitly indicates the correct output for the state input. Instead, the agent learns using reward.

Table 1. Summary of Testing Approaches Using Reinforcement Learning

Paper	Learning Algorithm	Learning Target	Reward Function
[1], [4]	Q Learning	Event	Optimistic Initial Value Method
[6], [5], [8]	Q Learning	Event	GUI Change Ratio
[9]	Deep Q Network	Component	GUI Change Ratio

Although each work has their own unique points, the basics are the same. The Q-learning agent interacts with the Android application to gradually build up a model and then creates test cases based on the model. The agent searches for the best input solution that allows it to test as many features as possible. Q-learning based testing has been able to achieve higher code coverage than random testing.

Still, there are some differences that need to be noted. First, we have already stated above that there is Q-learning as well as a "deep" version of Q-learning. Second, what the agent learns (i.e., *learning target*) differs; most work targeted events, while one targeted components. Finally, the function used to calculate the reward differs; some were based on the GUI change ratio, while others were based on "Optimistic Initial Value Method." Table 1 summarizes the differences. Thus, we see at least three factors that need to be considered when applying reinforcement learning to testing, specifically learning algorithm, learning target, and reward function.

Although each work has conducted their own evaluation, it is not clear how the three factors affect testing. In order to bridge this gap, we have conducted an empirical study that compares the eight combinations that can be made from the three factors. Table 2 shows the eight combinations that we target in our study.

In the rest of this paper, we first describe the three fac-

DOI reference number: 10.18293/SEKE2021-046

Table 2. Eight Combinations based on the Three Factors of Learning Algorithms, Learning Targets and Reward Functions

ID	Related Work	Learning Algorithm	Learning Target	Reward Function
1	[1], [4]	Q Learning	Event	Optimistic Initial Value Method
2	[6], [5], [8]	Q Learning	Event	GUI Change Ratio
3		Q Learning	Component	Optimistic Initial Value Method
4		Q Learning	Component	GUI Change Ratio
5		DQN	Event	Optimistic Initial Value Method
6		DQN	Event	GUI Change Ratio
7		DQN	Component	Optimistic Initial Value Method
8	[9]	DQN	Component	GUI Change Ratio

tors that we focus on: learning algorithm, learning target, and reward function. Section 3 describes the design of our empirical study. Section 4 discusses the results of our study. Section 5 discusses threats to validity. Section 6 makes concluding remarks.

2. Factors in Q-learning based testing

This section describes the three factors that we focus on; learning algorithm, learning target, and reward function.

2.1. Learning algorithm

Q Learning. Q-learning is a learning algorithm for reinforcement learning that has been used in the work of Mariani et al. [6], Korogulu et al. [5], Adamo et al. [1], Esparcia et al. [4], and Vuong et al. [8]. There are two methods of reinforcement learning: one based on value functions and the other based on strategy search. Q-learning is an algorithm based on value functions. The value function estimates how good it is for an agent to perform an action in a given state. The criterion for the estimation is the expected future reward, called the cumulative reward. Since the future reward depends on which action the agent will take, the number of values is defined according to a specific strategy. We define the value function $Q^{\pi}(s,a)$ as follows:

$$Q^{\pi}(s,a) = E[R \mid s,a,\pi] \tag{1}$$

The value function returns the cumulative reward that can be achieved by performing a sequence of actions that starts from s with action a, and then following the policy π from the succeeding state.

The optimal Q function $Q*$ returns the maximum cumulative reward that can be obtained from a given pair of state and action.

$$Q^{*}(s_t,a_t) = \max_{\pi} \sum_{t>0}^{T-1} \left(\gamma^{t} r_t \mid s=s_t, a=a_t, \pi\right) \tag{2}$$

If the optimal Q value $Q^{*}(s_{t+1},a_{t+1})$ for the next step is known, then the optimal strategy is to take the action that maximizes $r+\gamma Q^{*}(s_{t+1},a_{t+1})$. r is the immediate reward for the current step. $Q*$ satisfies the Bellman equation.

$$Q^{*}(s_t,a_t) = r(s_t,a_t) + \gamma \max_{a_t+1} Q(s_{t+1},a_{t+1}) \tag{3}$$

γ is the discount rate, which is a value between 0 and 1. The discount rate determines whether to give more weight to immediate or cumulative rewards; closer to 0 means more immediate rewards and closer to 1 means more cumulative rewards.

The optimal strategy $\pi*$ is to take the action with the largest Q value Q*. The Q-learning algorithm iteratively calculates the value of the Q function based on equation (3). First, the Q-function is initialized with a default value. Each time the agent performs an action to go from state s_t to state s_{t+1} and receives a reward r_t, the Q function is updated as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(r_t + \gamma \max_{a} Q(s_{t+1},a) - Q(s_t,a)) \tag{4}$$

α is the learning rate, which is a value between 0 and 1. It determines how much the Q-value is to be updated.

On the surface, choosing the action with the largest Q value seems to be favorable, but there is a need to balance the trade-off between *exploration* and *exploitation* [7]. ϵ-greedy approach is often used in reinforcement learning. Simply, a random action is selected with probability ϵ, or the action with the highest Q-value is selected with probability $1-\epsilon$.

Deep Q-Network. This is the learning algorithm used in the work of Vuong et al. [9]. The strength of neural networks lies in their ability to learn from low-dimensional feature representations and their ability to approximate complex functions. Using the approximation properties of neural networks, it is possible to approximate the optimal strategy $\pi*$ and the optimal value function $Q*$. The extension of reinforcement learning with neural networks is deep reinforcement learning. One of the best known methods for deep reinforcement learning is Deep Q-Network (DQN).

The optimal value function Q can be obtained by a neural network by using the weights θ.

$$Q\left(s_t, a_t, \theta\right) \approx Q^*\left(s_t, a_t\right) \tag{5}$$

Training is done by adjusting the weights θ_i at iteration i so that the mean square error of equation (3) becomes small. The right term in equation (3) is replaced by the following:

$$r\left(s_t, a_t\right) + \gamma \max_{a_t+1} Q\left(s_{t+1}, a_{t+1}, \theta_i^-\right) \tag{6}$$

2.2. Learning target

Event. Mariani et al. [6], Adamo et al. [1], Esparcia et al. [4] and Vuong et al. [8] targeted events for learning. GUI testing tools usually interact with the application under test by sending events to GUI components. For example, a click (event) is sent to a button (component) that transitions to the next page.

Component. Vuong et al. [9] targeted components for learning. The goal of reinforcement learning algorithms is to search for hard-to-reach application features in a way that reveals them. They tried to achieve this goal by considering the semantics of components and by making components as the learning target.

2.3. Reward function

Optimistic Initial Value Method. The optimistic initial value method is a reward function used in the work of Adamo et al. [1] and Esparcia et al. [4]. It is commonly used because it is simple to implement and effective for simple problems. However, so far no theoretical guarantees have been given, and in practice, it is not efficient because many iterations are required before the correct value propagates and overrides the optimistic value. As shown in equation (7), we make the function such that the reward for unexplored actions is maximized, so that all actions are explored exhaustively.

$$R\left(s_t, a_t\right) = \frac{1}{f\left(s_t, a_t\right)} \tag{7}$$

s_t is the state at step t, and a_t is the action at step t. $R\left(s_t, a_t\right)$ is the reward for taking action a_t in state s_t. $f\left(s_t, a_t\right)$ is the number of times action a_t is taken in state s_t.

GUI Change Ratio. GUI Change Ratio is a reward function used in the work of Mariani et al. [6] and Vuong et al. [8] [9]. As shown in equation (8), by considering the percentage of GUI changes, the reward is determined so that new features can be explored.

$$R\left(s_t, a_t, s_{t+1}\right) = \frac{\mid s_{t+1} \backslash s_t \mid}{\mid s_{t+1} \mid} \tag{8}$$

s_t is the state at step t, and a_t is the action at step t. $R\left(s_t, a_t, s_{t+1}\right)$ is the reward for the transition to state s_{t+1} as a result of taking action a_t in state s_t. The right term in Equation (8) is a ratio that indicates how much the number of GUIs is changed when the agent transitions from one state s_t to the next s_{t+1}.

3. Experiment Design

3.1. Overview

We conducted a comparative study of the eight combinations that are shown in Table 2, and aim to answer the following four research questions:

- RQ1: Which reinforcement learning algorithm is better: Q-learning or Deep Q-Network?
- RQ2: Which learning target is better: events or components?
- RQ3: Which reward function is better: optimistic initial value method or GUI change ratio?
- RQ4: Which combination will give the highest test quality?

The implementation of the eight combinations were done by extending existing implementations. For combinations which are based on Q-learning, we extended ClassicQ, which was originally implemented in [8]. For combinations which are based on Deep Q-Network, we extended QDroid which was originally implemented in [9].

We investigated the code coverage for twelve Android applications to evaluate test quality. Although not perfect, code coverage is often used to check the quality of test. We used Androtest [2], an automated test tool evaluation framework, for obtaining code coverage. We measured class coverage, method coverage, block coverage, and line coverage. Two-hour tests were conducted five times for each of the twelve applications under test, and the average code coverage was calculated.

3.2. Parameter Settings

Two important parameters in Q-learning is discount rate γ and learning rate α. We took into account the parameter values used in previous work, and also conducted some trial-and-error executions of our tool. Based on this, we chose the values for these parameters to be $\gamma = 0.9$ and $\alpha = 1.0$.

Another important parameter is ϵ. As with [8], the initial value of ϵ is set to 1 (i.e., always randomly choose an action), and continually decreased it until $\epsilon = 0.5$.

3.3. Target Applications

The applications to be tested as benchmarks are the datasets used in Vuong et al. [9]. These applications are the ones included in Androtest. The dataset consists of twelve Android applications, as shown in Table 3.

Table 3. Twelve Target Applications

App Name	LOC	URL
AnyMemo	8428	https://f-droid.org/en/packages/org.liberty.android.fantastischmemo/
My Expenses	2935	https://f-droid.org/en/packages/org.totschnig.myexpenses/
Who has my stuffs	729	https://f-droid.org/en/packages/de.freewarepoint.whohasmystuff/
Tippy Tipper	1083	https://github.com/mandlar/tippytipper
Munch Life	254	https://github.com/averyada/MunchLife
Mini Note Viewer	3673	https://f-droid.org/en/packages/jp.gr.java_conf.hatalab.mnv/
Mileage	4628	https://f-droid.org/en/packages/com.evancharlton.mileage/
Multi SMS sender	828	https://f-droid.org/packages/com.hectorone.multismssender/
Hot Death	3902	https://f-droid.org/en/packages/com.smorgasbork.hotdeath/
Random Music Player	400	https://f-droid.org/en/packages/com.simplemobiletools.musicplayer/
Dalvik Explorer	1375	https://f-droid.org/en/packages/org.jessies.dalvikexplorer/
Weight Chart	1116	https://f-droid.org/forums/topic/weight-chart/

4. Experiment Results and Discussion

Table 4 shows the average values of class coverage, method coverage, block coverage, and line coverage for each of the eight combinations. Table 5 shows the average method coverage values for each of the combination and each of the application.

For both Tables 4 and 5, the result in the row with the best coverage value is in bold font. So, for example, in Table 4, ID8 had the best class coverage at 62.58%, while ID2 had the best method coverage for the application AnyMemo at 38.8%. For the results of each application, we only show method coverage and not the other three coverages due to space issues, but the tendency was the same.

We now discuss each research question. For each research question, we first discuss based on Table 4 which gives the overall results, and then discuss based on Table 5 at the application level.

4.1. RQ1: Which reinforcement learning algorithm is better: Q-learning or Deep Q-Network?

Since we had learning target and reward function as factors, in order to compare Q-learning and Deep Q-Network, we compared each of the pair (ID1, ID5), (ID2, ID6), (ID3, ID7), and (ID4, ID8). When we look at the results of each of these pairs in Table 4, in **all** cases the combinations using Deep Q-Network had the better results. We also conducted statistical analysis, but we did not obtain a significant difference in each of the pair.

When we look at the results for each application in Table 5, we can see that this depends on each application. ID8 was better than (or the same as) ID4 for all applications. But for the three other pairs, about half had Q-learning better, and about half had Deep Q-Network better.

RQ1 Answer: Overall, Deep Q-Network was found to be better than Q-learning, but the difference was not statistically significant. When looking at each application, there was not a clear cut tendency for one over the other except for ID8, which was better than (or the same as) ID4 for all applications.

4.2. RQ2: Which learning target is better: events or components?

Similar to RQ1, we compared each of the pair (ID1, ID3), (ID2, ID4), (ID5, ID7), and (ID6, ID8). In **most** cases in Table 4, the combination using components had a better result. Only the class coverage and line coverage for the pair (ID5, ID7) had events with the better results. We also conducted statistical analysis, and we found that for the pair (ID6, ID8), ID8 (component) was significantly different (better) than ID6 (event). The difference in other pairs were not statistically significant.

When comparing for each application (Table 5), again there was not a clear cut tendency towards either event or component, except for ID8. ID8 was better than (or the same as) ID6 in eleven out of twelve applications.

RQ2 Answer: Component was found to be better than event in most cases, but there was one pair where the difference was found to be statistically significant; ID8 (component) was found to be better than ID6 (event). This was also seen at the application level.

4.3. RQ3: Which reward function is better: optimistic initial value method or GUI change ratio?

We compared each of the pairs (ID1, ID2), (ID3, ID4), (ID5, ID6), and (ID7, ID8). Looking at the overall results in Table 4, for the pairs using Q-learning, i.e., (ID1, ID2) and

Table 4. Overall Results: Average

ID	1	2	3	4	5	6	7	8
Class	54.10	53.24	57.38	57.05	58.42	60.03	57.89	**62.58**
Method	44.80	44.37	47.02	46.41	47.83	48.45	47.88	**52.98**
Block	40.82	39.90	43.08	41.75	43.25	43.75	43.84	**47.95**
Line	39.98	39.25	42.33	41.40	43.18	43.48	42.98	**47.75**

Table 5. Results of each Application: Method Coverage

App Name	ID1	ID2	ID3	ID4	ID5	ID6	ID7	ID8
AnyMemo	36.0	**38.8**	26.8	26.0	28.2	28.2	31.4	31.8
My Expenses	29.0	35.2	35.0	39.8	45.6	50.4	30.6	**64.4**
Who has my stuffs	**81.4**	69.8	75.2	76.2	76.4	76.4	79.8	80.2
Tippy Tipper	52.8	**54.8**	54.4	53.8	54.2	54.0	50.4	54.6
Munch Life	**51.2**	48.0	48.0	48.0	48.0	48.0	48.0	48.0
Mini Note Viewer	42.2	39.6	32.6	37.4	38.0	38.0	41.0	**48.2**
Mileage	34.4	35.3	27.2	26.4	26.8	27.6	33.6	**35.5**
Multi SMS sender	37.4	37.2	**38.0**	36.8	37.0	37.0	35.0	37.2
Hot Death	16.4	17.0	59.2	60.6	59.8	61.2	59.0	**64.2**
Random Music Player	54.0	54.0	54.0	54.0	54.0	54.0	54.0	54.0
Dalvik Explorer	**80.6**	73.8	76.6	64.2	65.4	65.4	77.5	78.6
Weight Chart	22.2	29.0	37.2	33.8	40.6	**41.2**	34.2	39.0

(ID3, ID4), optimistic initial value method had better results, but it was not statistically significant. But for the pairs using Deep Q-Network, i.e., (ID5, ID6) and (ID7, ID8), it was the opposite, i.e., GUI change ratio had better results. However, in these cases also, the difference was not statistically significant.

When we look more closely at the application level in Table 5, we can see that for the pairs using Deep Q-Network, change ratio had the same or better results in all but one case, where in Tippy Tipper ID5 was better than ID6 by just 0.2%, which should be considered as negligible. For the pairs using Q-learning, there was little difference between the two reward functions.

RQ3 Answer: There was no statistically significant difference. But there was a tendency for optimistic initial value method to be better for Q-learning, and change ratio to be better for Deep Q-Network.

4.4. RQ4: Which combination will give the highest test quality?

From Table 4, we can see that ID8 (Deep Q-Network, component, GUI change ratio) was found to have the best results for all types of coverage. We also checked the statistical difference between ID8 and each of the other combinations. Except for the pair (ID1, ID8), the difference was statistically significant for all other combinations.

This can also be seen in Table 5. ID1 had three apps with the best results, while ID8 had four. ID8 was better than ID1, in terms of number of apps with the best results, although not by much.

We also note that ID2 had two apps and ID3 and ID6 had one app each with the best result. However, for these four apps, we can see that the difference with the other combinations were not that large. When compared with ID8, the difference in these four apps ranged from 0.2% (Tippy Tipper) to 7.0% (AnyMemo).

RQ4 Answer: The combination of Deep Q-Network, component and GUI change ratio was found to be the best combination.

4.5. Further Discussion

In Table 5, when comparing the results for ID1 and ID8, we can also see that the results for ID8 was more stable. The lowest and second lowest results for ID8 were 31.8% (AnyMemo) and 35.5% (Mileage), while the two lowest results for ID1 was 16.4% (Hot Death) and 22.2% (Weight Chart). Note though that although Mileage was the second worst result for ID8, it was still the best result among all eight combinations for Mileage.

Although we focused on the three factors of learning algorithm, learning target and reward function, we must not forget that other parts still need work, especially being able to generate "meaningful" strings. For example, in Table 5, the results of Random Music Player for all combinations

was 54%. We manually checked the results, and found that it wasn't just the method coverage value itself that was the same; the methods that were covered were also the same. This was because one of the functions in Random Music Player requires the input of a URL. However, none of the eight combinations were able to generate a meaningful URL, and thus all functions (methods) that can be used after entering a URL could not be tested.

5. Threats to Validity

Internal Validity. We limited each execution to two hours. Since reinforcement learning is an approach that learns while executing, there are two possible issues. First, it may be possible that the coverage would continue to increase if the execution time was longer. Second, the shape of the coverage curve may differ between execution, i.e., some executions may cover more code quickly while others may not be as quick. Third, there may be differences between each execution. Although we cannot completely negate these possibilities, we tried to minimize these as much as possible by taking the average of five executions for each combination and application.

Another threat to internal validity is the parameters of reinforcement learning. The execution results will vary depending on the parameters values, specifically discount rate γ, learning rate α, and ϵ-Greedy value ϵ. To mitigate this threat, we selected parameter values such as discount rate and learning rate based on empirical analysis reported in previous studies, as well as some trial-and-error execution of the combinations.

External Validity. In this study, we targeted twelve Android applications. In terms of sampling bias, it is possible that completely different results could be obtained if the current test were conducted on different applications.

Construct Validity. We used code coverage to assess how good an app was tested. Using code coverage for this purpose has long been considered to be controversial [3]. Thus, code coverage may not be perfect for comparison. But code coverage is used in many testing papers, and we believe that it is adequate enough as one way to compare testing approaches.

6 Conclusions and Future Work

We conducted an empirical comparative study of Android application testing focusing on the three factors of learning algorithms, learning targets, and reward functions. We implemented eight combinations based on these three factors and executed them on twelve applications, and measured code coverage. We compared and discussed the eight combinations based on four research questions. We found that the combination of Deep Q-Network, component and GUI change ratio was the best combination (RQ4). For the other research questions, there was little statistical significance, although we did discuss some trends.

Future work includes investigating other Android applications to eliminate external validity. Also, executing for more than two hours needs to be considered. Finally, as was discussed in subsection 4.5, one major issue that needs to be solved regardless of the three factors is being able to generate "meaningful" strings when necessary.

References

[1] D. Adamo, M. K. Khan, S. Koppula, and R. Bryce. Reinforcement Learning for Android GUI Testing. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, A-TEST 2018, pages 2–8, 2018.

[2] Androtest. http://bear.cc.gatech.edu/ shauvik/androtest/. Accessed: 2021-1-18.

[3] X. Cai and M. R. Lyu. The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the 1st International Workshop on Advances in Model-Based Testing*, A-MOST '05, pages 1–7, 2005.

[4] A. I. Esparcia-Alcazar, F. Almenar, U. R. M. Martinez, and T. E.J. Vos. Q-learning strategies for action selection in the TESTAR automated testing tool. In *Proceedings of META 2016 6th International Conference on Meta heuristics and Nature Inspired Computing*, pages 174–180, 2016.

[5] Y. Koroglu, A. Sen, O. Muslu, Y. Mete, C. Ulker, T. Tanriverdi, and Y. Donmez. Qbe: Qlearning-based exploration of android applications. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pages 105–115, 2018.

[6] L. Mariani, M. Pezze, O. Riganelli, and M. Santoro. AutoBlackTest: Automatic Black-Box Testing of Interactive Applications. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 81–90, 2012.

[7] Y. Shen and C. Zeng. An adaptive approach for the exploration-exploitation dilemma in non-stationary environment. In *2008 International Conference on Computer Science and Software Engineering*, volume 1, pages 497–500, 2008.

[8] T. Vuong and S. Takada. A Reinforcement Learning Based Approach to Automated Testing of Android Applications. In *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, A-TEST 2018, pages 31–37, 2018.

[9] T. Vuong and S. Takada. Semantic analysis for deep q-network in android gui testing. In *Proceedings of 32nd International Conference on Software Engineering and Knowledge Engineering*, pages 123–128, 2019.