Evaluation of Q-learning and SARSA Algorithms in Grid Based Game

Ani Sanikidze

***Abstract*—The following report explores the applications of reinforcement learning algorithms in a custom, randomized grid environment. In particular, model-free algorithms, Q-learning and were applied for training the agent. The challenge of randomization was solved by storing all the possible states of the grid.**

Based on the provided results, both Q-learning and SARSA were able to learn the optimal-policy. The Q-learning outperformed SARSA at the beginning of the training process, however, ultimately their performances were almost the same. As a result, the agent was able to play the grid game successfully.

I. Introduction

The following paper presents the implementation of model-free algorithms, namely Q-learning and SARSA on the randomized grid game. The paper introduces how proposed algorithms work, followed by a hypothesis about the efficiency of the aforementioned algorithms. The environment description and implementation details are given as well. Both of the algorithms were evaluated in terms of convergence speed and obtained cumulative rewards. The conclusions are drawn and supported by corresponding graphs.

II. Proposed Algorithms

Q-learning and SARSA are model-free, TD learning algorithms; therefore, the updates of Q function are based on TD error. The TD error is obtained by summing the immediate reward and the difference between the new Q value and the already estimated one. At each iteration the value of Q value is moved to the direction of the calculated TD error. Both of them utilize the Q table for storing and updating the Q values of the corresponding state-action pairs and are designed to maximize cumulative reward by choosing the most optimal actions. The optimal Q values are obtained using the Bellman optimality equation.

The main difference between the aforementioned algorithms lies in the fact that the Q-learning is an off-policy algorithm, while the SARSA – on-policy. The term off-policy refers to the fact that at each time-step the agent obtains the optimal policy by utilizing absolute greedy policy and behaves using other policies, such as e-greedy. In SARSA, the agent learns optimal policy and behaves using the same policy, therefore the SARSA is referred to as an on-policy algorithm [3].

It is worth noting that when applying $\epsilon - greedy$ policy, the hyperparameter ϵ determines the exploration-exploitation trade-off in both algorithms. In other words, at each time step, the action with the highest Q value is chosen with the probability of $(1-\epsilon)$ and a random action with the probability of ϵ. Therefore, $\epsilon - greedy$ is a more balanced action selection algorithm than a fully greedy or fully random policy.

III. Hypothesis

Q-learning and SARSA are generally suitable for solving the most of the tabular problems, i.e., the environments where the state and action space are a collection of discrete values. According to Szepesvari C., if all state-action pairs are visited infinitely often and suitable hyperparameters are applied, then the computed Q values are proved to converge to Q^* with the probability of 1 [1].

For both algorithms e-greedy policy will be applied. It is assumed that this approach will ease the process of obtaining optimal Q values for both Q-learning and SARSA algorithms.

The difference between the expected results of Q-learning and SARSA is difficult to assume as they are very similar to each other. However, based on the difference between the update rules, during exploration the Q-learning will directly learn the optimal policy, whilst SARSA – near-optimal policy. Based on this, it is assumed that SARSA will take longer but safer paths during learning process, whereas the Q-learning will take shortest path with the risk of loosing the game. Nevertheless, with applying e-greedy policy, ϵ will be gradually decreased and SARSA will achieve the optimal policy as well.

Performance of the algorithms is also dependent on the grid size. It is expected that algorithms will converge to optimal Q value even in relatively large environment. However, if the state space is extremely large, then it might not be feasible to establish lookup table to train the agent.

IV. Environment

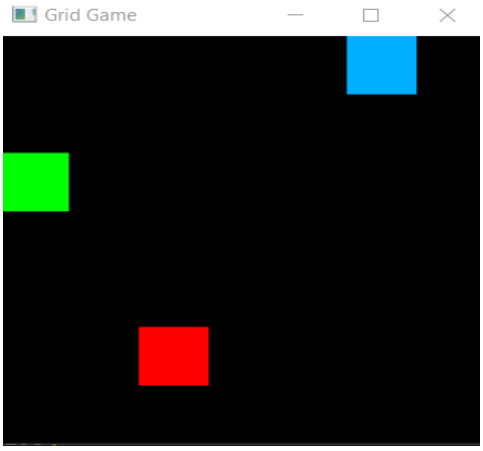

Fig. 1. Grid Game GUI

In order to demonstrate and compare different algorithms to each other, a custom grid environment was created. The implemented environment is similar to OpenAI GYM environments

[2]. The minimum size of the grid is 2 and it consists of the following components: agent (blue square), food (green square) and bomb (red square). The positions of each square are selected randomly at the beginning of each episode. The main goal of the agent is to maximize the cumulative reward by finding the shortest path between the starting position and the main reward – food. The implementation of the prototype can be accessed on github - https://github.com/AniSanikidze/Grid-Game.

A. Rewards

The agent receives -1 reward for every step it takes. This encourages the agent to take the shortest path to the end goal by minimizing punishments (i.e., negative rewards). There are two terminating points in the environment: bomb and food. Bomb is associated with the minimum negative reward of -100, while food is associated with the maximum positive reward of +100. At the end of each episode, the agent wins by reaching the food or loses an episode by encountering the bomb. Besides, the agent receives negative reward of – 10 for taking off-grid action.

B. State space

Since the environment is random, there are a large amount of possible combinations of placing the grid components. This challenge of randomization was solved by encoding each possible state of the grid with unique integer values. The states are stored as tuples and each of them contains information about the positions of the agent, food and bomb. The different squares are represented in the following way:

- Empty - 0
- Agent – 1
- Bomb - 2
- Food - 3

This way of storing the state space enables the agent to identify the current state of the grid and take actions based on Q values stored in the lookup table.

C. Actions

The 4 actions that the agent can take under each state are the following: move north, south, west or east. The actions are encoded with corresponding integer values : 0, 1, 2 and 3.

V. Training Results and Evaluation

The implemented algorithms were applied to the 7 by 7 grid and the hyperparameters were tuned as follows: $\epsilon = 1$ (gradually decreased using linear function), $\alpha = 0.5$, $\gamma = 0.9$. Figure 2 and 3 show the results of training agent for 150 000 episodes.

As it can be seen, the average rewards gradually approach the maximum rewards and minimum rewards are also decreased through the training process, meaning that both algorithms converged to the optimal policy. The graphs also contain information about the average number of steps the agent took during each episode. It is clearly presented that number of steps the agent takes decreases as the training process proceeds.

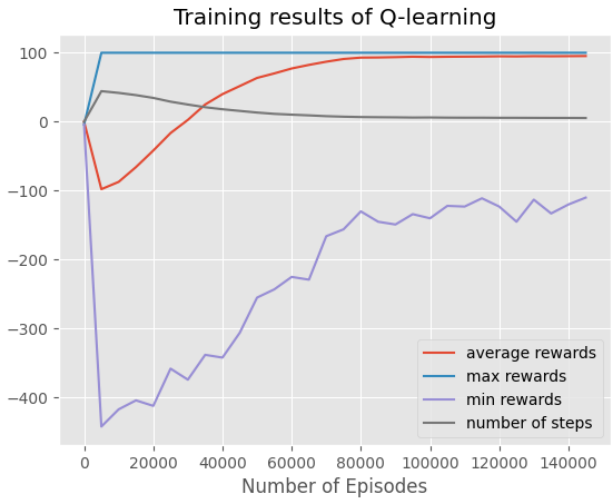

Fig. 2. Training Results of Q-learning on 7x7 grid

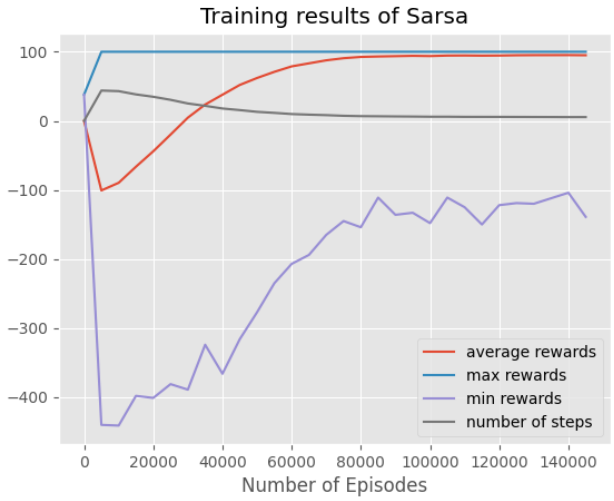

Fig. 3. Training Results of SARSA on 7x7 grid

VI. Convergence Comparison of Q-learning and SARSA

In order to draw a conclusion of which algorithm works better in the given environment, the results were plotted on the same graph. Figure 4 shows that both algorithms work similarly. The main difference can be noticed at the beginning of the training process, when the Q-learning displays slightly better performance than SARSA. This fact confirms the above given hypothesis that SARSA takes longer paths during the learning process than Q-learning. Nevertheless, it is clear that both algorithms converge to Q*. Comparing the applied algorithms with the random agent aimed to show how the performance of the trained agent is gradually improved, while the average episode rewards of the random agent stays almost the same and is always negative.

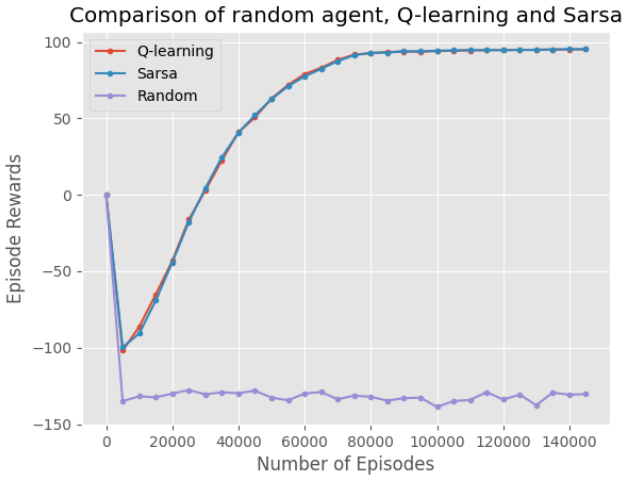

Fig. 4. Convergence Comparison of Q-learning,SARSA and Random Agent on 7x7 grid

VII. Experimental Results and Evaluation

Apart from convergence comparison of Q-learning and SARSA, both of them were experimented on different sized grids and with different policies. Since the performance of the given algorithms are extremely similar, results of only Q-learning are presented in this section.

A. Q-learning on different sized grids

The following section evaluates the Q-learnign performance on 3 by 3 and 7 by 7 grids.

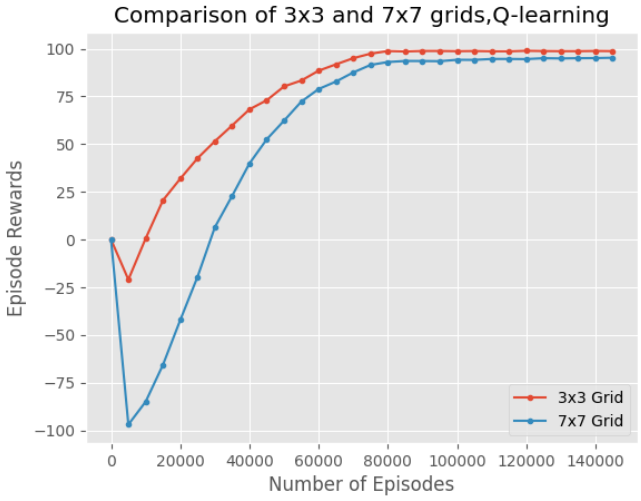

Fig. 5. Comparison of 3x3 and 7x7 grids,Q-learning

As Figure 5 shows, Q-learning finds an optimal policy regardless of the environment size. The average episode rewards in a 7x7 grid are slightly less even when the algorithm converges. This is explained by the fact that the agent receives -1 reward at each time step; therefore in the bigger grids the agent will always end up with less average rewards than in the smaller one.

B. Exploration/Exploitation

Both algorithms were implemented using $\epsilon - greedy$ policy. In order to find out, the best policy for Q-learning, different ϵ values were experimented. As a baseline, $\epsilon - greedy$ policy is used that decays the starting epsilon value of 1 using the linear function and it is compared to the fixed epsilons with the values of 0.5 and 0.1. The following figure displays the performance of Q-learning using three different policies.

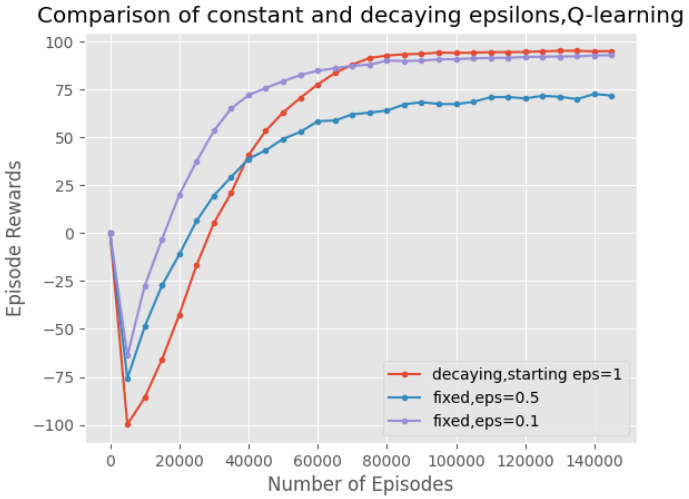

Fig. 6. Comparison of Q-learning using different policies

The $\epsilon - greedy$ shows the worst performance at the beginning of the training, as the agent is heavily exploring the environment, however, it is clear that decaying epsilon outperforms both of the fixed ones in the end. This proves the idea presented in the hypothesis that using $\epsilon - greedy$ would help the algorithms to achieve the best results.

The remaining hyperparameters, namely γ and α, were kept constant throughout the training process. Experimenting with γ and α values did not show drastic changes in the performance of implemented algorithms, therefore they are not examined.

VIII. Conclusion

In conclusion, the paper introduced application of Q-learning and SARSA on the grid game. Based on the provided results and evaluations, Q-learning and SARSA proved to be working well in randomized grid enviornments. Besides, the Q-learning showed great performance in both small and larger grids. Finally, e-greedy policy proved to be the best approach for the aforementioned algorithms.

References

[1] Szepesvari, C., *Algorithms for Reinforcement Learning* , Online available: https://sites.ualberta.ca/szepesva/papers/RLAlgsInMDPs.pdf [Accessed 10 January 2022]

[2] *OpenAI GYM* , Online available: https://gym.openai.com/

[3] R. S. Sutton and A. G. Barto *Reinforcement Learning* ,published by the MIT Press. London,2018. Online available: http://www.incompleteideas.net/book/RLbook2020.pdf [Accessed 12 January 2022]