Comparative Analysis of On and Off Policy Reinforcement Learning

Rong Jian Yee
dept. of Computer Science
University of Nottingham
Nottingham, United Kingdom
psyrjy@nottingham.ac.uk

***Abstract*—Reinforcement learning has made significant progress due to the increased uses of robotics and automation. Some popular real-world application of reinforcement learning includes self-driving cars and gaming robots such as Alpha-Go. This paper investigates two types of reinforcement learning algorithms and presents a performance comparison in solving the Mountain Car problem. In this paper, the two algorithms that will be comparing is Q-Learning and SARSA. The results show that SARSA is the most efficient and effective at solving this problem.**

***Index Terms*—Reinforcement Learning, On-Policy Learning, Off-policy Learning, SARSA, Mountain Car problem**

I. Introduction

Reinforcement learning is one of the important paradigms of artificial intelligence. Reinforcement learning is all about learning what to do in specific situations to maximise a numerical reward[1]. The learner which is the agent in this experiment is not told which actions to take, but it has to explore the environment and discover which actions lead to the most reward. This paper will be investigating two very similar algorithms namely SARSA algorithm and Q-Learning algorithm that are in the subset of Temporal-Difference Learning, the difference between these two algorithms is one is On-policy and the other is Off-policy algorithms. Temporal-Difference learning in Sutton et al. book refers to "a class of model-free reinforcement learning methods which learn by bootstrapping from the current estimate of the value function

By investigating the two methods with the Mountain-Car problem which was first introduced in Andrew Moore's paper[2], this will contributes to real-life application where steps that seem to prevent agent to reach the goal state are necessary to solve the problem. For example in chess, in some situation, one must sacrifice some pieces in order to win the game. In addition, the main question that will be addressed in this paper is what are the differences in performance for the two algorithms to solve the Mountain Car problem.

The remainder of this paper is structured as follows. In the next section, some background and related work on the problem and algorithms will be introduced. In Section 3, the detailed methods will be described. In Section 4, the details of the environment and the parameters settings will be presented. In Section 5, obtained results will be used to discussed and compare the methods. In the last section, a conclusion will be drawn from the experimental results and some further works.

II. Background & Literature

A. Mountain Car Problem

In this experiment, the agent will be trying to solve the Mountain Car Problem that is introduced by Andrew Moore and further defined by Singh and Sutton[3]. Essentially, a car is trying to drive up a steep hill shown in Figure1. However, the main challenge is that the gravity is much stronger than the force of the engine and the car cannot accelerate up the steep hill at full force. The solution to this problem is that the car must accelerate backwards up the hill and then forward with a full thrust in order to gain enough speed and momentum to reach the goal at the top of the hill.

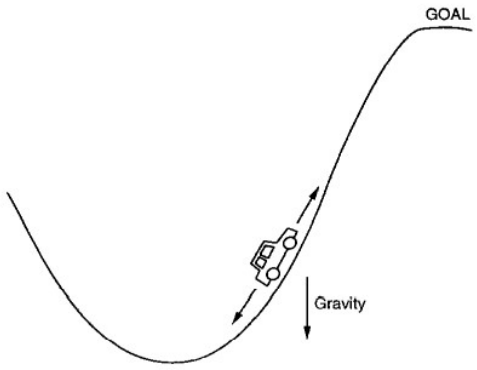

Fig. 1. An illustration of the Mountain Car problem from [3]

Even though this problem might be seen as trivial for human, it is actually an unusual and intriguing task for an agent. Unlike other problems where the agent learns to reach the goal at the very beginning, the agent must learn to first learn to perform actions that prevent it from reaching the goal in order to get better in the long run. This is a simple task where things have to must get worse before they can get better as described by Sutton and Singh[3]. In this paper, this kind of problem will be referred to as the "Sacrificing problem".

B. On and Off-policy Learning

A policy in Reinforcement learning defines how an agent will act in a specific state.

In Sutton and Barto's work[1], they strictly defined that in On-policy learning, the agent will always be exploring and tries to find the best policy that still explores. On the other hand, in Off-policy learning, the agent will also explore but learns a deterministic optimal policy that could be unrelated to the policy that the agent originally follows. In this experiments, Q-Learning and SARSA will be used to represent the two types of algorithm.

1) Q-Learning: Q-Learning algorithm where the Q stands for quality is an example of Off-policy algorithm and it is one of the early breakthroughs in Reinforcement Learning developed by Watkins[4]. Essentially, Q-Learning algorithm is a model-free algorithm where the agent attempts to learn the Q-function or the optimal way in the environment by estimating which actions will lead to the goal in the long run. The aim is to maximise the reward gain from performing certain actions. A detailed explanation will be covered in the next section.

2) SARSA: SARSA algorithm which stands for state-action-reward-state-action was named by Sutton et al. It is essentially a modified version of Q-Learning algorithm but it is an example of On-policy algorithm. SARSA was originally developed by Rummery and Niranjan[5], the authors questioned the effectiveness of Q-Learning algorithm in finding the best estimate of the return for each state. Since in the early stages of learning, the Q-value for actions that have not been explored is likely to be completely wrong. Therefore they proposed SARSA algorithm which ensures the temporal differences errors add up correctly regardless of whether greedy actions are taken or not.

III. Methodology

In both algorithms Q-Learning and SARSA, they have an update policy and behaviour policy. The update policy defines how the agent learns and behaviour policy defines how the agent behaves under a certain state.

A. ϵ-Greedy Strategy

Both of the algorithms defined later will be using the ϵ-greedy strategy to balance exploration and exploitation, where it will determine whether the agent will explore the environment randomly or greedily choose the optimal action that has been learned. Therefore, the agent can take advantage of prior knowledge while exploring new options simultaneously. The decision in which action a is chosen for the agent at a given state is defined by a policy $\pi(s) = a$. At each state, the agent will perform random action with a fixed probability $0 \leq \epsilon \leq 1$, otherwise, it will perform the best action in a given state as illustrated in Figure 2

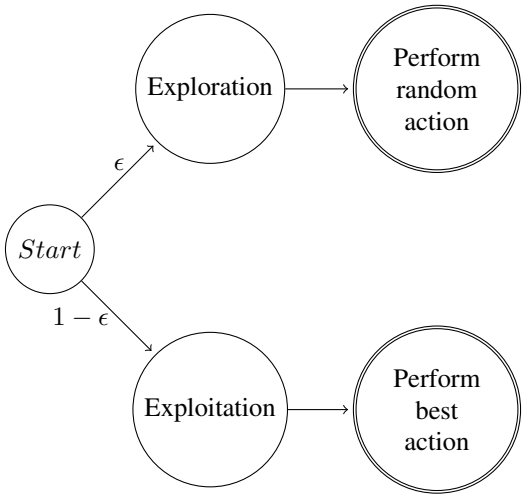

Fig. 2. ϵ-greedy strategy

B. Q-Learning

In Q-Learning algorithm, it uses a look-up table called Q-table and it can be used to calculate the maximum expected future reward or Q-value for each state and action pair (s, a). To calculate the Q-value of each state and action pair the Q-function uses the Bellman equation that takes in two input $s \in S$ and $a \in A$, where S is the state space and A is the action space. The equation is described as follows:

$$Q'(s,a) \leftarrow Q(s,a) + \alpha \overbrace{[\underbrace{R + \gamma \max_a Q(s',a)}_{\text{new value}} - \underbrace{Q(s,a)}_{\text{old value}}]}^{\text{temporal difference error}}$$

Where α, R and γ is the learning rate, the reward for performing action a in state s and the discount rate respectively. Both α and γ are in the range of 0 and 1. The procedural form of Q-learning is shown in Algorithm 1. In Q-Learning, the agent learns the optimal solution by using an absolute greedy policy and behave using ϵ-greedy policy as described in the previous section. Since update and behaviour policy is different, it is therefore considered as an Off-policy algorithm.

Algorithm 1: Q-Learning Algorithm (Off-policy)

Input: $\alpha \in (0, 1]$, $\epsilon > 0$
Initialise $Q(s, a)$ for all state-action pair, $s \in S$, $a \in A$
for *episode* $\in$ *Episodes* **do**
  Initialise s
  while s *is not terminal state* **do**
    a = ϵ-greedy from s
    Take action a, observe R, s'
    $Q'(s,a) =$
     $Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
    s = s'
  end
end

C. SARSA

Similarly to Q-Learning above, SARSA also utilised the Q-table to calculate Q-value for each state-action pair (s, a).

However, the function that is used to update Q-table is slightly different which is described as follows:

$$Q'(s,a) \leftarrow Q(s,a) + \alpha \overbrace{[\underbrace{R + \gamma Q(s',a')}_{\text{new value}} - \underbrace{Q(s,a)}_{\text{old value}}]}^{\text{temporal difference error}}$$

Where the parameters are exactly the same in Q-Learning. Instead of using the weighted max reward that can be obtained in s' , SARSA uses the weighted future reward received from the next state-action observation. The main difference between these two algorithm is how they learn the optimal solution. In SARSA, the agent will learn the optimal policy and behave according to the same policy which is the ϵ-greedy policy defined in the next section which is why it is an On-policy algorithm. Detailed pseudocode of the SARSA algorithm is shown in Algorithm 2.

Algorithm 2: SARSA Algorithm(On-policy)

Input: $\alpha \in (0, 1]$, $\epsilon > 0$
Initialise $Q(s, a)$ for all state-action pair, $s \in S$, $a \in A$
for *episode* $\in$ *Episodes* **do**
  Initialise s
  a = ϵ-greedy from s
  while s *is not terminal state* **do**
    Take action a, observe R, s'
    $Q'(s,a) = Q(s,a) + \alpha[R + \gamma Q(s',a') - Q(s,a)]$
    $s = s'$ $a = a'$
  end
end

Algorithm 3: REINFORCE: Monte-Carlo policy gradient algorithm

procedure REINFORCE(α)
Initialize policy parameters θ arbitrarily
for `each episode`
$(s_1, a_1, r_2, ..., s_{r-1}, a_{r-1}, r_T) \sim \pi_\theta$ **do**
  for $t = 1$ *to* $T - 1$ **do**
    $\theta \leftarrow \theta + \alpha \cdot G_t \nabla_\theta \log_{\pi_\theta}(a_t|s_t)$
  end
end
return θ
end procedure

IV. Experimental Setup

A. Environment

The environment that will be used in the experiment will be the Mountain Car environment provided by OpenAi Gym.[1] There are some necessary pre-processing to the environment in order to fit the use of the experiment. First of all, since both Q-Learning and SARSA algorithms are tabular learning methods, they can only be used in discrete observation space.

The observation space of this environment consists of two elements: 1) Car position which ranges from -1.2 to 0.6 2) Car Velocity which ranges from -0.07 to 0.07. Therefore, the environment has to be converted into discrete space and it is achieved by using the 'numpy' library to split the observation space evenly into 20 discrete states in this experiment. It should be noted that the number of discrete space should be within a reasonable range otherwise it might hinder the learning of the agent.

The action space for this environment is simply 3 discrete values [0, 1, 2] which denotes accelerate to the left, don't accelerate and accelerate to the right respectively. As for the reward space, a reward of 0 will be awarded to the agent if only it reached the goal state which is when car position $\geq$ 0.5 and -1 will be awarded if it is not reached the goal state for every action. Originally, the maximum steps that the environment allowed before it is terminated are 200 moves, however, 200 moves are not enough for the agent to learn and therefore it is adjusted to 1000 moves.

B. Hyper-Parameters

In both algorithms, they share the same parameters and the same values will be used in this experiment so that the results can be used solely for investigating the impact of On and Off-policy.

- α: This is the learning rate or step size for the algorithms. This value should be in the range of $0 \leq \alpha \leq 1$ where setting 0 means that the agent will not learn from new actions and 1 means that the agent will not values prior knowledge. α is set to 0.05 so that the agent will learn at a reasonable rate.
- γ: This is the discount rate of the algorithm where $0 \leq \gamma \leq 1$. This parameter determines how much the agent values rewards in long term relative to short term or immediate rewards. When γ is set to 0, the agent will only perform actions that generate the best immediate result whereas when it is 1 the agent might prevent the model from convergence. γ is set to 0.9 so that the agent focus on long term reward while not neglecting short term results.
- ϵ: This parameter is related to the ϵ-greedy strategy where $0 \leq \epsilon \leq 1$. ϵ will determine how much the agent will be exploring or exploiting. In this experiment, two configurations of ϵ will be used. A constant value 0.5 will be used for the first case, this will balance the probability of the exploration and exploitation. In the second case, ϵ is initially set to 1 and it will decay over time as the agent train. This is because, at the early stage of learning, the values in Q-table are initialised with 0 which is likely to be incorrect for most state-action pair. Therefore, the agent will start learning by exploring the environment and then start exploiting the Q-table once it has sufficient data.

[1] https://gym.openai.com/envs/MountainCar-v0/

C. Baseline

Random learning will be the baseline method for this experiment where the agent will perform random action at any given state. This describes the worst-case scenario where the agent is not learning anything even with a large number of episodes shown in Figure 3.

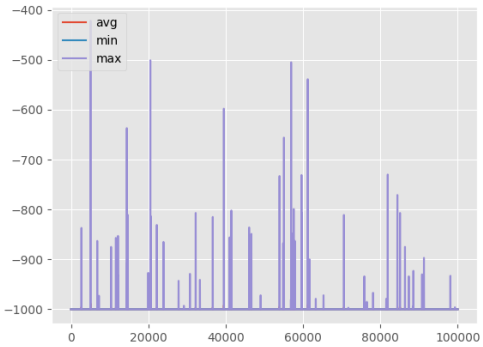

Fig. 3. The cumulative reward of an agent performing only random actions

V. Results and Discussion

In this section, the results will be discussed in the two cases of ϵ configuration: 1) ϵ Decay 2)ϵ Constant. For every algorithm, three videos showing how the agents solve the problem in three stages of learning are recorded for both algorithms and can be found in the repository.

A. ϵ Constant

The results of both algorithms are shown in Figure 4 and Figure 5 respectively. Both algorithms managed to reach convergence within a similar time frame shown in Table I. However, the two algorithms have found the different solution to the problem as the mean cumulative reward of Q-Learning is around -262, while SARSA managed to have a mean cumulative reward of -176 shown in Figure 6 and Table II. By comparing the mean cumulative reward of both algorithms, SARSA clearly outperformed Q-Learning even though both of them managed to converged.

The results obtained in this experiment contradict the results obtained by Sutton and Barto[1] in their comparison of the two algorithms in Cliff Walking problem. This is because Q-Learning algorithm will always try to find the optimal solution to solve the problem while SARSA finds a safer solution.

In ordinary problems such as the Cliff Walking problem where the optimal solution is just taking the best possible action in each state, Q-Learning will perform better because it is committed to finding the optimal solution. However, in a "Sacrificing problem" like the Mountain Car problem, Q-Learning did not learn to take actions that are not optimal initially for solving the problem. In the recorded video, the Q-Learning agent actually accelerates forward in the first few frames while SARSA agent accelerates backwards. Therefore, this provides evidence that Q-Learning is not as effective as SARSA in this case.

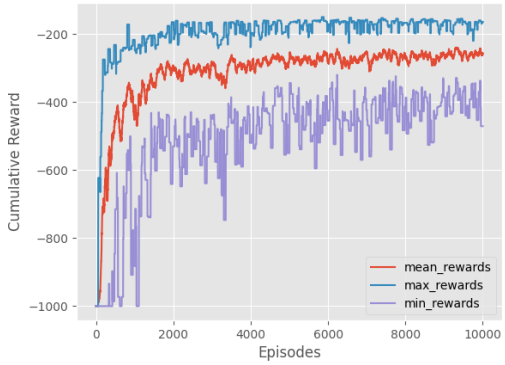

Fig. 4. Cumulative reward of Q-Learning with constant ϵ

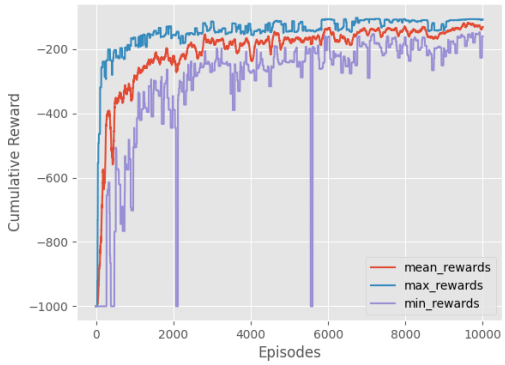

Fig. 5. Cumulative reward of SARSA with constant ϵ

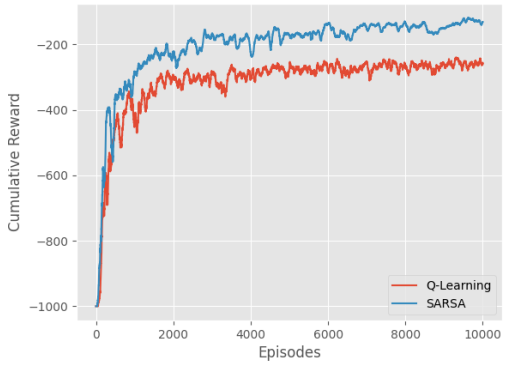

Fig. 6. Mean cumulative reward of algorithms with constant ϵ

B. ϵ Decay

In this case, where the ϵ is decay over time, both algorithms also managed to reach convergence. Unlike the previous

section, where they find a different solution when ϵ gradually decrease, both algorithms managed to find the same optimal solution which the mean cumulative reward is around -174 to -191 shown in Figure 7, Figure 8 and Table II.

Despite both algorithms has found the same optimal solution, the time taken for each agent to solve the problem differ drastically. Table I shows that Q-Learning algorithm takes about 4 times longer than SARSA algorithm. This is clear evidence that SARSA is way more efficient than Q-Learning algorithm in finding the optimal solution in "Sacrificing problem".

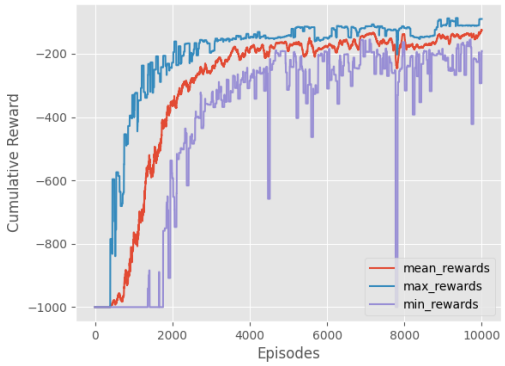

Fig. 7. Cumulative reward of Q-Learning with constant ϵ

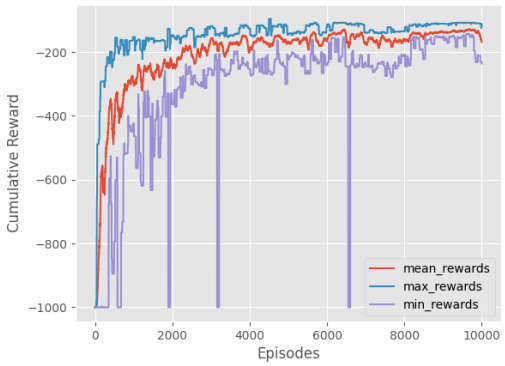

Fig. 8. Cumulative reward of SARSA with constant ϵ

TABLE I
TIME TAKEN IN SECONDS FOR AGENT TO SOLVE THE PROBLEM

	Configuration	
Algorithm	ϵ Decay	ϵ Constant
Q-Learning	41.5	13.8
SARSA	10.7	11.5

VI. Conclusion

In this paper, we have investigated the impact and differences of On and Off-policy learning algorithm in solving the

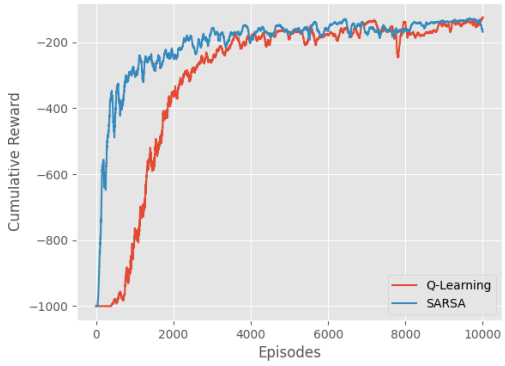

Fig. 9. Mean cumulative reward of algorithms with constant ϵ

TABLE II
AVERAGE CUMULATIVE REWARD FOR BOTH ALGORITHM

	Configuration	
Algorithm	ϵ Decay	ϵ Constant
Q-Learning	-191	-262
SARSA	-174	-176

Mountain Car problem. We have also seen how these two algorithms perform with two different configurations of ϵ. Therefore, by considering the results obtained in this experiment. It is safe to conclude that On-policy learning algorithm performs significantly better than Off-policy algorithm in solving the "Sacrificing problem" both in terms of efficiency and quality of the solution.

VII. Future Work

Due to computation limitations, we were only able to perform comparative analysis in a relatively simple environment. Future studies could fruitfully explore this issue further by comparing the two learning algorithms in a more complex environment, for example playing Atari games or Chess. In addition, further studies could be comparing the differences between deep On and Off-policy learning algorithms.

References

[1] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press Ltd, 2018.

[2] A. W. Moore, "Efficient memory-based learning for robot control," 1990, ISSN: 1476-2986.

[3] R. S. S. Satinder P. Singh, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, pp. 123–158, 1996. DOI: https://doi.org/10.1007/BF00114726. [Online]. Available: https://link.springer.com/article/10.1007/BF00114726#citeas.

[4] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992. DOI: 10.1007/BF00992698.

[5] G. Rummery and M. Niranjan, “On-line q-learning using connectionist systems,” *Technical Report CUED/F-INFENG/TR 166*, Nov. 1994.