



REINFORCEMENT LEARNING

Dynamic Programming, Monte-Carlo, Sarsa, Q-learning



MAY 13, 2021

BEN GURION UNIVERSITY

YOEL BOKOBZA – 312190481 SHAHAF YAMIN - 311530943

Problem definition

Frozen Lake: Given is a penguin on a frozen lake, which is described by a 4x5 grid with holes and a goal state (fish), both defining terminal states. For every transition the penguin gets a reward of

$$r_{t+1}(S_{t+1}) = \begin{cases} 0.96, & S_{t+1} = 17 \\ -1.04, & S_{t+1} = 1, 2, 10, 12, 20 \\ -0.04, & \text{else} \end{cases}$$

The penguin can take four possible $\mathcal{A} = \{N, E, S, W\}$, but the surface is slippery and only with probability 0.8 the penguin will go into the intended direction and with probability 0.1 to the left and 0.1 to the right of it. It is assumed that the boundaries are reflective, i.e., if the penguin hits the edge of the lake it remains in the same grid location. We used a discount factor of $\gamma = 0.9$ throughout this exercise.

Denote by \mathcal{S} the state space and let \mathcal{R} be the set of all valid rewards $\{0.96, -1.04, -0.04\}$.

gridworld

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

Now, we will define some definitions for the rest of this work.

The value function defined as $v_\pi: \mathcal{S} \rightarrow \mathbb{R}$ such that $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$, Intuitively, the value function represents what is the average return starting from state s and following policy π .

The action-value function defined as $q_\pi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ such that $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$, Intuitively, the action value function represents what is the average return starting from state s applying action a and thereafter following policy π .

Where in both cases, $G_t = \sum_{i=1}^T \gamma^{i-1} R_{t+i}$, and T is the terminal state.

We define policy as the mapping $\pi: \mathcal{S} \rightarrow \mathcal{A}$. In addition, we denote the estimation of the value and action-value function by V and Q , respectively.

Q.1 Dynamic Programming

To solve this MDP, we used the Policy-Iteration algorithm as we learned in class.

The Policy-Iteration algorithm is a model-based algorithm, and it is composed of two algorithms: Policy-Evaluation, and Policy-Improvement.

- **Policy-Evaluation** - In the Policy-Evaluation algorithm, given a policy π we estimate the corresponded value function v_π (Here we used the iterative method), where

$$\begin{aligned}
 v_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) \cdot \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} \Pr(s', r | s, a) \cdot (r + \gamma \cdot v_\pi(s')) \\
 &\stackrel{(1)}{=} \sum_{a \in \mathcal{A}} \pi(a|s) \cdot \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} \Pr(r | s, a, s') \cdot \Pr(s' | s, a) \cdot (r + \gamma \cdot v_\pi(s')) \\
 &\stackrel{(2)}{=} \sum_{a \in \mathcal{A}} \pi(a|s) \cdot \sum_{s' \in \mathcal{S}} \Pr(s' | s, a) \cdot (r(s') + \gamma \cdot v_\pi(s'))
 \end{aligned}$$

(1)– Bayes Theorem

(2) - Because r is deterministic given s' , $\Pr(r | s, a, s')$ is equal to 1 for a single $r \in \mathcal{R}$ and 0 otherwise.

In order to determine what is the convergence accuracy, we need to define a new parameter θ . θ represents the maximal difference between values of the value function through 2 consecutive iterations. E.g., when:

$$\max_{s \in \mathcal{S}} |v^{(n+1)}(s) - v^{(n)}(s)| < \theta$$

we determine that the evaluation procedure has converged.

- **Policy-Improvement** – In this algorithm, given v_π we derive an improved policy using greedy strategy.

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s, a) \cdot (r(s') + \gamma \cdot v_\pi(s'))$$

As we learned in lectures, $v_{\pi'}(s) \geq v_\pi(s), \forall s \in \mathcal{S}$, e.g., policy improvement theorem guarantees that the value function improves.

Dynamic Programming Results

As part of this algorithm, we used $\theta = 10^{-12}$ to enforce high accuracy. As requested in the question we've extracted both the policy and the state value function.

gridworld - DP Optimal Policy

	→	→	→	
1	5	9	13	17
2	↑		→	↑
	6	10	14	18
→	→	→	↑	↑
3	7	11	15	19
→	↑		↑	
4	8	12	16	20

gridworld - DP Optimal State Value

0.0	0.408	0.565	0.909	0.0
1	5	9	13	17
0.0	0.054	0.0	0.748	0.909
2	6	10	14	18
-0.089	0.085	0.178	0.581	0.732
3	7	11	15	19
-0.123	-0.089	0.0	0.178	0.0
4	8	12	16	20

As expected, the algorithm has converged and learned a policy which aims to maximize the overall reward. As a first step, we can verify that this makes sense, we can verify that $v_\pi(s = 13) \approx 0.909$.

$$\begin{aligned}
 v_\pi(s = 13) &= \sum_{s' \in \mathcal{S}} \Pr(s'|s = 13, a = E) \cdot (r(s') + \gamma \cdot v_\pi(s')) \\
 &= 0.8 \cdot (0.96 + 0.9 \cdot 0) + 0.1 \cdot (-0.04 + 0.9 \cdot v_\pi(s = 13)) + 0.1 \\
 &\quad \cdot (-0.04 + 0.9 \cdot 0.748) \Rightarrow v_\pi(s = 13) \approx 0.909
 \end{aligned}$$

In addition, although the number of steps using policy π from $s = 19$ and $s = 9$ to the goal is equal, we see that their state values are not equal. The reason for this phenomenon is the stochasticity of the environment, if the environment was deterministic their state values were equal.

For the rest of this work, we will use model-free algorithms. Moreover, to encourage exploration, we are using the $\epsilon - greedy$ policy, in this policy, ϵ is used to determine the exploration and exploitation trade-off. In a mathematically form:

$$\pi(a|s) = \begin{cases} 1 - \epsilon_i + \frac{\epsilon_i}{|\mathcal{A}|}, & \text{if } a = \underset{A \in \mathcal{A}}{\operatorname{argmax}} Q(s, A) \\ \frac{\epsilon_i}{|\mathcal{A}|}, & \text{otherwise} \end{cases}$$

Where i represents the i^{th} episode and $\epsilon_i = \frac{1}{EpsilonDecay \cdot i + 1}$

In addition, in all the algorithms we are using GLIE (Greedy in the Limit of Infinite Exploration) Intuitively speaking, this means that the ϵ_i decays to zero in the limit $i \rightarrow \infty$.

And we can see that indeed $\lim_{i \rightarrow \infty} \epsilon_i = 0, \forall EpsilonDecay \in (0,1)$.

Q.2 Monte Carlo

In this question, we were asked to implement a Monte-Carlo GLIE, first-visit, α -constant control algorithm. Similar to the policy-iteration algorithm, this algorithm composed of two main components evaluation and control.

Monte Carlo Evaluation

Using Monte-Carlo algorithm we can estimate $\mathbb{E}_\pi[G_t | S_t = s]$ by the following unbiased and consistent estimator $V_\pi(s)$, which is the average over the returns of different trajectories from the first appearance of state s . (we count only trajectories that contains state s).

In a mathematically form,
$$V_\pi(s) = \frac{\sum_{\tau \in \#episodes} \sum_{t=0}^{T(\tau)-1} \mathbb{I}(S_t = s \cap s \notin \{S_0, \dots, S_{t-1}\}) G_t}{N(s)}$$

Where, $T(\tau)$ is the number of timesteps in the τ^{th} episode and $N(s) = \sum_{\tau \in \#episodes} \sum_{t=0}^{T(\tau)-1} \mathbb{I}(S_t = s \cap s \notin \{S_0, \dots, S_{t-1}\})$.

An average calculation can be done online using the following formula, we denote $\bar{a}^{(n)}$ as the average over n values $\{a_i\}_{i=1}^n$, and we want to calculate the average over $\{a_i\}_{i=1}^{n+1}$, using $\bar{a}^{(n)}$.

$$\bar{a}^{(n+1)} = \frac{\sum_{i=1}^{n+1} a_i}{n+1} = \frac{\sum_{i=1}^n a_i + a_{n+1}}{n+1} = \frac{n \cdot \bar{a}^{(n)} + a_{n+1}}{n+1} = \bar{a}^{(n)} + \frac{1}{n+1} \cdot (a_{n+1} - \bar{a}^{(n)})$$

Using the same principles, we can update $V_\pi(s)$ online.

$$V^{(n+1)}(s) = V^{(n)}(s) + \frac{1}{N(s)} (G_t - V^{(n)}(s))$$

As we learned in class, in non-stationary environments we can use an exponentially recency-weighted moving average instead of the incremental Monte-Carlo e.g.:

$$V^{(n+1)}(s) = V^{(n)}(s) + \alpha (G_t - V^{(n)}(s)), \alpha > 0$$

Monte Carlo Control

In a similar way to the estimation of $v_\pi(s)$, we can estimate the true action value function $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$ using the following estimator:

$$Q_\pi(s, a) = \frac{\sum_{\tau \in \#episodes} \sum_{t=0}^{T(\tau)} \mathbb{I}(S_t = s \cap A_t = a \cap (s, a) \notin \{(S_0, A_0), \dots, (S_{t-1}, A_{t-1})\}) G_t}{N(s, a)}$$

Where,

$$N(s, a) = \sum_{\tau \in \#episodes} \sum_{t=0}^{T(\tau)} \mathbb{I}(S_t = s \cap A_t = a \cap (s, a) \notin \{(S_0, A_0), \dots, (S_{t-1}, A_{t-1})\})$$

Similarly, we can use exponentially recency-weighted moving average instead of the incremental Monte-Carlo e.g.:

$$Q^{(n+1)}(s, a) = Q^{(n)}(s, a) + \alpha (G_t - Q^{(n)}(s, a))$$

The improvement step follows the aforementioned $\epsilon - greedy$ policy using the current estimation of the action value function Q i.e.

$$\pi(a|s) = \begin{cases} 1 - \epsilon_i + \frac{\epsilon_i}{|\mathcal{A}|}, & \text{if } a = \underset{A \in \mathcal{A}}{\operatorname{argmax}} Q(s, A) \\ \frac{\epsilon_i}{|\mathcal{A}|}, & \text{otherwise} \end{cases}$$





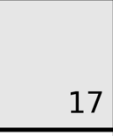
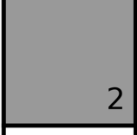

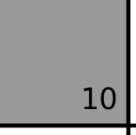









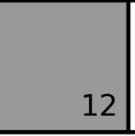

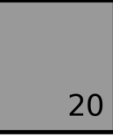
Where i represents the i^{th} episode and $\epsilon_i = \frac{1}{EpsilonDecay \cdot i + 1}$

We initialized the action value function such that our policy may choose each action for all non-terminal states with the same probability.

Monte-Carlo Results

we applied this algorithm with $EpsilonDecay = 10^{-4}$, $\alpha = 0.001$ and $1 \cdot 10^6$ episodes. (This parameters choice will be explained in the Hyper-parameters tuning section).

gridworld - MC-control Policy

 1	 5	 9	 13	 17
 2	 6	 10	 14	 18
 3	 7	 11	 15	 19
 4	 8	 12	 16	 20

gridworld - MC-control State Value

0.0 1	0.348 5	0.54 9	0.899 13	0.0 17
0.0 2	0.0 6	0.0 10	0.732 14	0.904 18
-0.161 3	0.01 7	0.167 11	0.533 15	0.698 19
-0.194 4	-0.199 8	0.0 12	0.059 16	0.0 20

gridworld - MC-control State-Action Value

	<div>-0.016 -0.753 0.348 -0.195</div>	<div>0.303 0.053 0.54 -0.658</div>	<div>0.68 0.423 0.899 0.516</div>	
	<div>0.0 -0.823 0.833 -0.309</div>		<div>0.612 -0.714 0.732 0.26</div>	<div>0.904 0.584 0.748 0.55</div>
<div>-0.839 -0.347 0.161 -0.266</div>	<div>-0.103 -0.222 0.01 -0.258</div>	<div>-0.825 -0.298 0.167 -0.846</div>	<div>0.533 0.069 0.456 0.022</div>	<div>0.698 0.199 0.345 -0.737</div>
<div>-0.194 -0.281 0.293 -0.287</div>	<div>-0.199 -0.265 0.858 -0.335</div>		<div>0.059 -0.482 0.529 -0.222</div>	

We have analyzed all algorithms performance at the end of this assignment.

Q.3 SARSA

SARSA is an on-policy $TD(0)$ control method. it can be applied as an online learning algorithm unlike the MC-control algorithm which requires the whole trajectory for the learning procedure.

In the $TD(0)$ algorithm we are using bootstrapping to estimate the return G_t .

Assume we are in some state S_t , taking an action A_t according to current $\epsilon - greedy$ policy, observing reward R_{t+1} , transitioning to state S_{t+1} and taking the next action A_{t+1} according to the same $\epsilon - greedy$ policy.

Now, we can estimate the return starting from state S_t and taking action A_t using the following:

$$\hat{G}_t \approx R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}).$$

we can use an exponentially recency-weighted moving average to iteratively estimate $q_\pi(s, a)$:

$$Q^{(n+1)}(S_t, A_t) = Q^{(n)}(S_t, A_t) + \alpha (\hat{G}_t - Q^{(n)}(S_t, A_t)), \quad \alpha > 0$$

SARSA Results

$EpsilonDecay = 10^{-4}$, $\alpha = 0.001$ and $1 \cdot 10^6$ episodes (This parameters choice will be explained in the Hyper-parameters tuning section).

gridworld - SARSA Policy

1	→ 5	→ 9	→ 13	17
2	↑ 6	10	→ 14	↑ 18
→ 3	→ 7	→ 11	↑ 15	↑ 19
→ 4	↑ 8	12	↑ 16	20

gridworld - SARSA State Value

0.0 1	0.402 5	0.583 9	0.904 13	0.0 17
0.0 2	0.085 6	0.0 10	0.742 14	0.908 18
-0.092 3	0.077 7	0.183 11	0.569 15	0.733 19
-0.115 4	-0.071 8	0.0 12	0.143 16	0.0 20

gridworld - SARSA State-Action Value

	0.15 -0.793 -0.097	0.459 0.152 -0.732	0.762 0.501 0.631	
	0.085 -0.801 -0.2		0.584 -0.653 0.305	0.908 0.656 0.628
-0.8 -0.254 -0.175	-0.009 -0.121 -0.107	-0.817 -0.203 -0.802	0.569 0.15 0.118	0.733 0.321 -0.674
-0.129 -0.169 -0.169	-0.071 -0.159 -0.237		0.143 -0.681 -0.218	

We have analyzed all algorithms performance at the end of this assignment.

Q.4 Q-Learning

Q-learning is an off-policy control method, similarly to SARSA, this algorithm can be applied as an online learning algorithm.

Assume we are in some state S_t , taking an action A_t according to current $\epsilon - greedy$ policy, observing reward R_{t+1} , and transitioning to state S_{t+1} .

Similarly, to the SARSA algorithm this algorithm also estimates the return according to the following formula:

$$\hat{G}_t \approx R_{t+1} + \gamma \cdot \max_{A \in \mathcal{A}} Q(S_{t+1}, A)$$

we can use an exponentially recency-weighted moving average to iteratively estimate $q_\pi(s, a)$:

$$Q^{(n+1)}(S_t, A_t) = Q^{(n)}(S_t, A_t) + \alpha \left(\hat{G}_t - Q^{(n)}(S_t, A_t) \right), \quad \alpha > 0$$

Q Learning Results

$EpsilonDecay = 10^{-4}$, $\alpha = 0.001$ and $1 \cdot 10^6$ episodes (This parameters choice will be explained in the Hyper-parameters tuning section).

gridworld - Q-Learning control Policy

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

gridworld - Q-Learning State Value

0.0 1	0.425 5	0.573 9	0.911 13	0.0 17
0.0 2	0.064 6	0.0 10	0.751 14	0.91 18
-0.08 3	0.09 7	0.195 11	0.583 15	0.73 19
-0.121 4	-0.092 8	0.0 12	0.167 16	0.0 20

gridworld - Q-Learning State-Action Value

	0.147 -0.555 -0.008	0.344 0.176 -0.452	0.601 0.415 0.515	
	0.064 -0.685 -0.162		0.593 -0.712 0.366	0.91 0.665 0.636
-0.829 -0.211 -0.129	0.005 -0.109 -0.095	-0.759 -0.176 -0.782	0.583 0.175 0.164	0.73 0.352 -0.704
-0.121 -0.147 -0.147	-0.092 -0.129 -0.21		0.167 -0.252 0.003	

Hyper parameters tuning

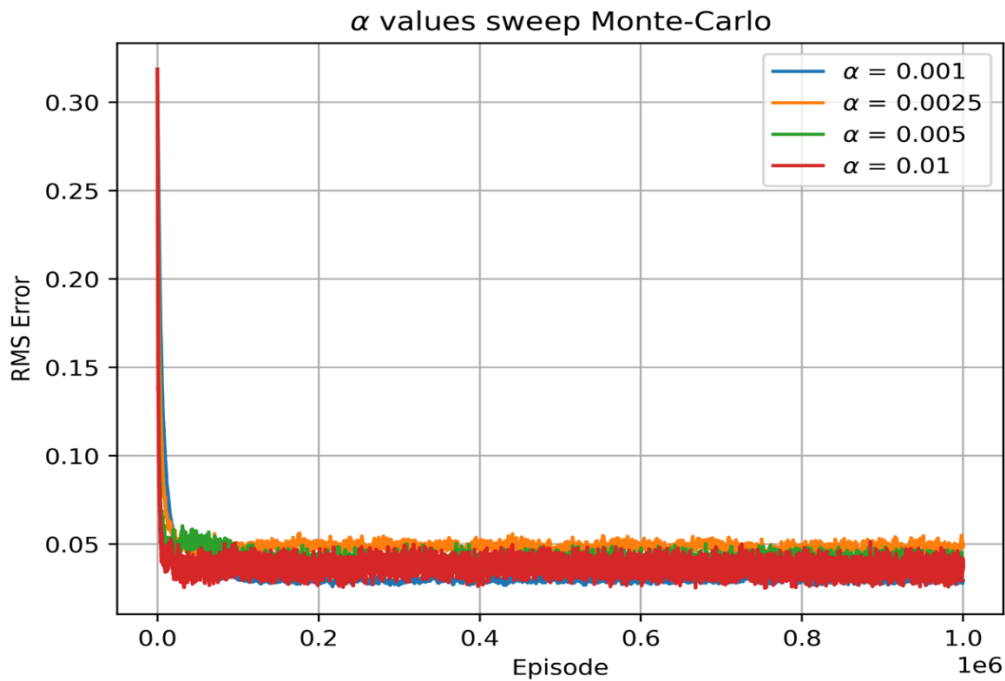
To evaluate performance of this algorithm for different values of α we've measured the average RMS error between the estimated value function V_π and the true value function v_π (We used the DP algorithm with $\theta = 10^{-12}$ value function estimation as the true value function). The average of the RMS is taken over 10 different runs of this algorithm, we denote each run result as V_π^m . V_π^m is a result of the algorithm after 1,000,000 episodes.

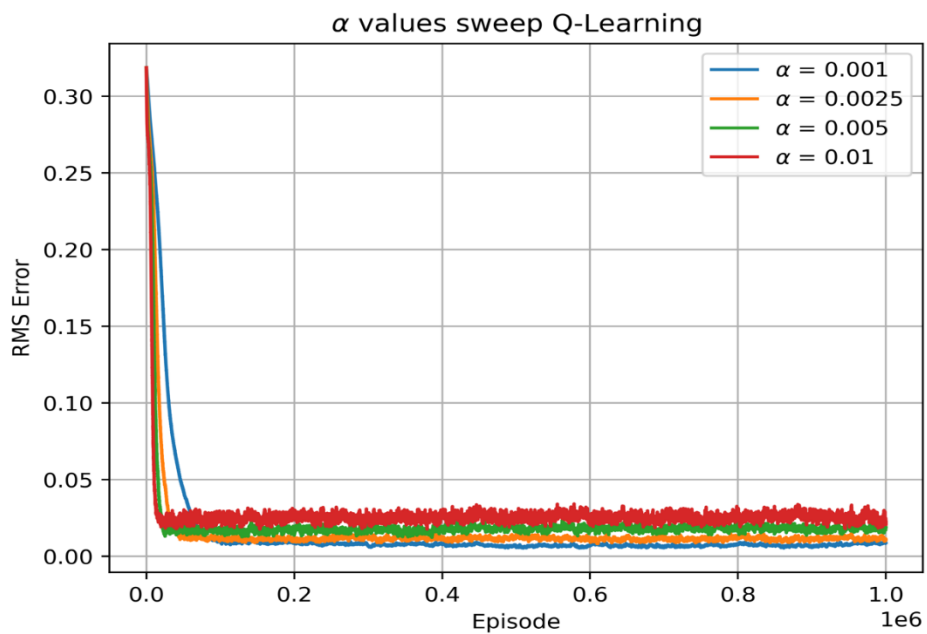
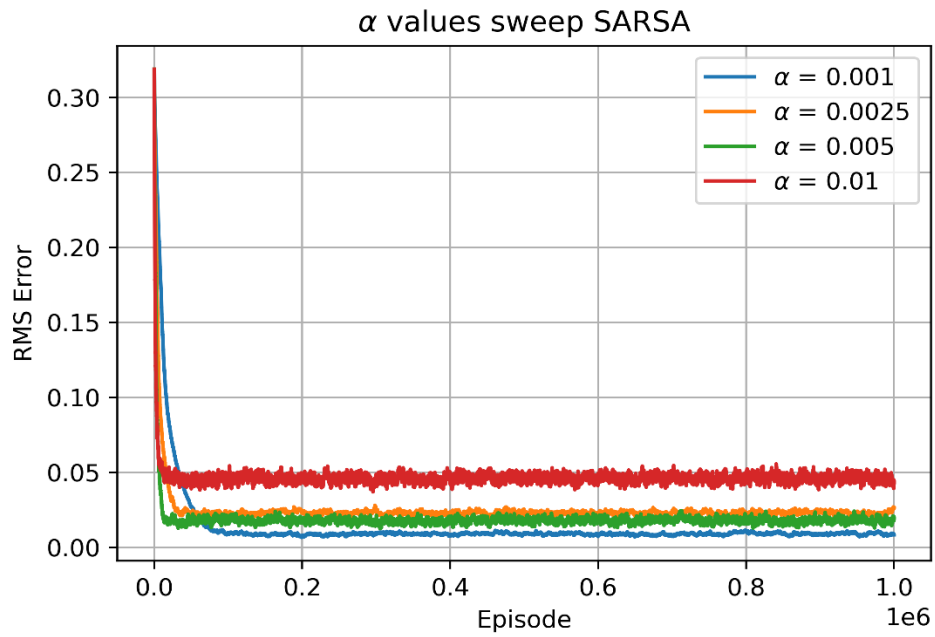
In a mathematically form:

$$RMSError = \frac{1}{10} \sum_{m=1}^{10} \sqrt{\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} (V_\pi^m(s) - v_\pi(s))^2}$$

In all the questions we have found out that $EpsilonDecay = 10^{-4}$ provides a very good balance between exploration and exploitation (when the number of episodes is 10^6).

With the choice of $EpsilonDecay = 10^{-4}$, and for each algorithm, we have plotted the figure of RMS Error VS episode number to determine which is the best α .





We can see from the figures above that decreasing the value of α reduced the fluctuation as we get closer to the optimal value. Furthermore, we see that the error gap between $\alpha = 0.001$ and $\alpha = 0.0025$ is very small and therefore we decided to stop the search optimization procedure at this α . In addition, we can see that decreasing the value of α leads to slower convergence as well.

From these figures, the best choice of α , for $EpsilonDecay = 10^{-4}$ and 10^6 episodes is $\alpha = 0.001$.

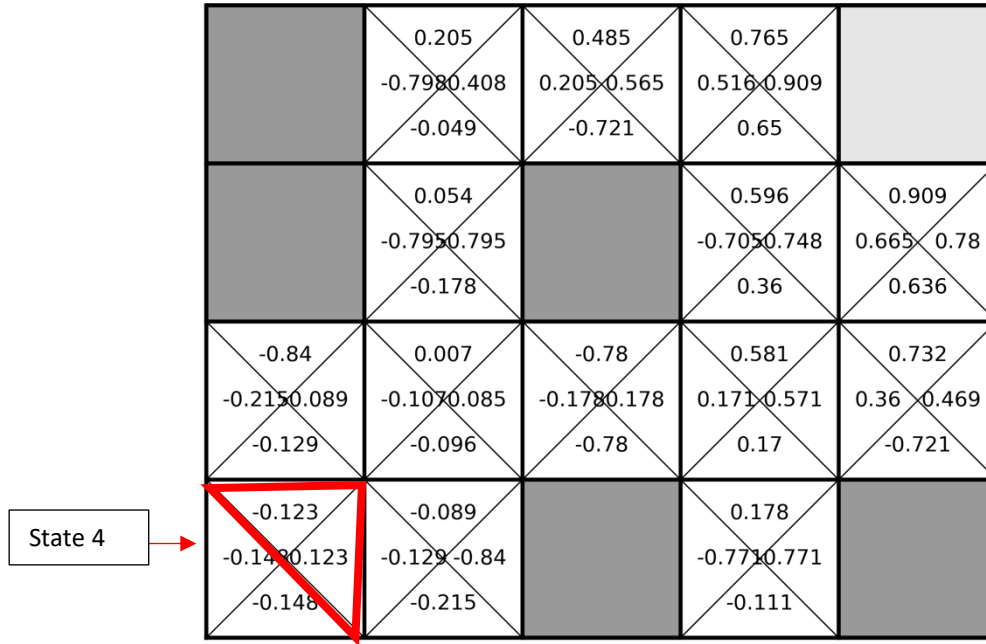
Algorithms Performance Analysis

The policy at state 4 is different from the DP policy (for Q-learning and MC algorithms). Although, their policy is still optimal because as we can see at the DP state-action value function, the following equality holds:

$$q^*(s = 4, a = 'N') = q^*(s = 4, a = 'E') = v^*(s = 4)$$

Thus, $\pi^*(s = 4) = \operatorname{argmax}_{a \in \mathcal{A}} q^*(s = 4, a) = 'N' = 'E'$

gridworld - Optimal State-Action Value

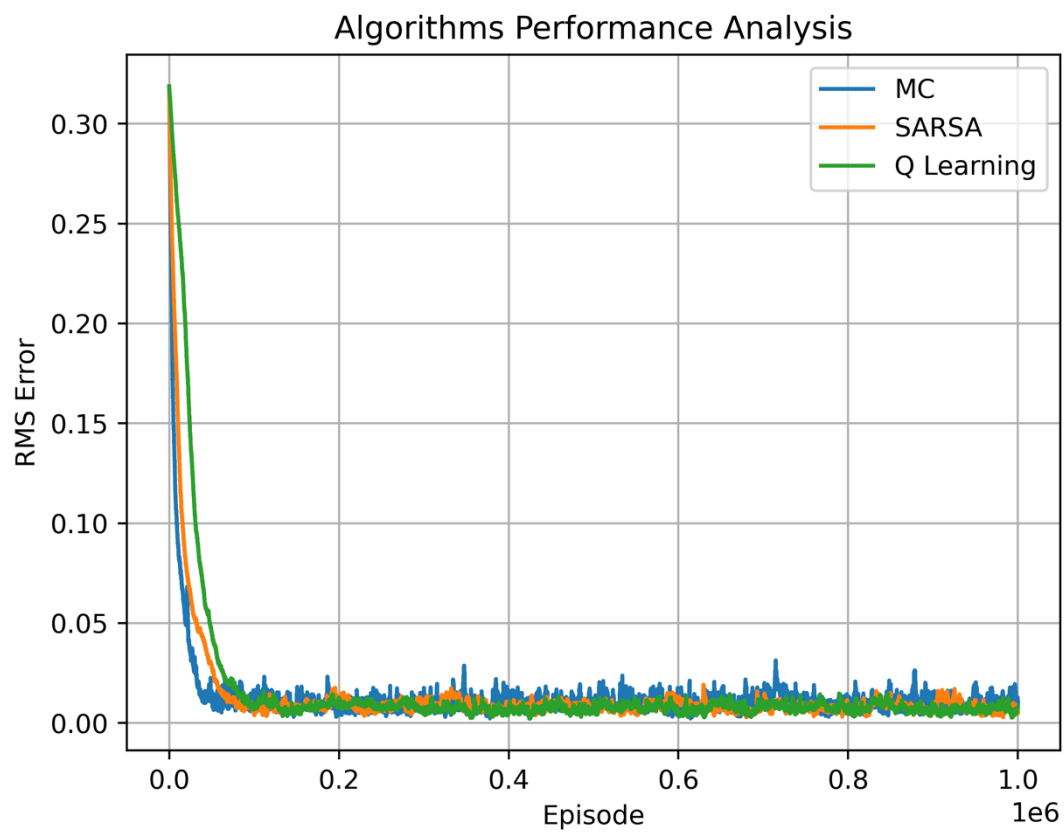


Therefore, we can see that using $EpsilonDecay = 10^{-4}$, and $\alpha = 0.001$, all three algorithms managed to find the optimal policy.

In addition, from the value function and the action value function we can see that for any $s \in \mathcal{S}$, $v_\pi(s) = \max_{a \in \mathcal{A}} (q_\pi(s, a))$ as expected because,

$$v_\pi(s) = q_\pi(s, \pi(s)) = q_\pi\left(s, \operatorname{argmax}_{a \in \mathcal{A}} (q_\pi(s, a))\right) = \max_{a \in \mathcal{A}} (q_\pi(s, a))$$

To further analyze those algorithms results we have run their training performance through time (with $EpsilonDecay = 10^{-4}$, $\alpha = 0.001$).



We can see that compared to both SARSA and Q-Learning, MC convergence trend is noisy. This phenomena match to our expectation, as we've learned during lectures, although MC is an unbiased estimator of the value function, it has high variance.