ECE372-Dynamic Programming & Optimal Control **Name:** *Mojtaba AlShams*
HW-02
Submission Due Date: 18 Nov. 2024

---

1. In this exercise, we will apply reinforcement learning methods to solve a maze example (maze-sample-10x10-v0) provided by the TA.
   Hyper-parameters were swept on both SARSA and Q-Learning algorithms for testing purposes to choose the same parameters for both algorithms to have a fair comparison and a learning convergence (i.e. an optimal solution is found). The parameters are maximum number of episodes, ϵ (for ϵ-greedy strategy), learning step-size and decaying factor γ. Hyper-parameters applied on both SARSA and Q-Learning algorithms are 500 as maximum number of episodes, ϵ swept from 0.001 to 0.5, constant learning step-size swept from 0.001 to 1.0 along with an adaptive step-size of $\frac{1}{t}$ where t is the step number in the episode and decaying factor γ=1. γ=0.9 was also experimented but its results are not included as SARSA did not converge on this decaying factor value.

   Kindly refer to the python codes for implementation details. It is possible to run run_QLearning.py and run_QLearning.py through the terminal. Args library enables changing algorithm's hyper-parameters defaults.

   (a) Use the SARSA algorithm and experiment with different step-size parameters.
   Fig. 1 shows the algorithm followed to implement SARSA algorithm. γ was set to 1 and both of α and ϵ were swept across varied values resulting in Fig. 2 and Fig. 3.
   ϵ was set to 0.1 in Fig. 2 where it shows that small values of α leads the algorithm to diverge which explains why adaptive step-size diverge as well. As number of steps increases, the step size decreases preventing the algorithm from convergence. Nevertheless, adaptive step-size's performance is better than small α values. α=0.1 gave the best performance and yielded the optimal solution.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 1: SARSA algorithm taken from chapter 6 of Sutton & Barto.

α was set to 0.1 in Fig. 3, although, SARSA converged for small ϵ values, Fig. 3 shows that very small values or too large values lead to less efficient performance as the

agent will not be able to properly explore the environment with small randomness or on the other hand too much randomness also causes the agent not to learn properly. ϵ=0.1 gave a good performance and an optimal solution.

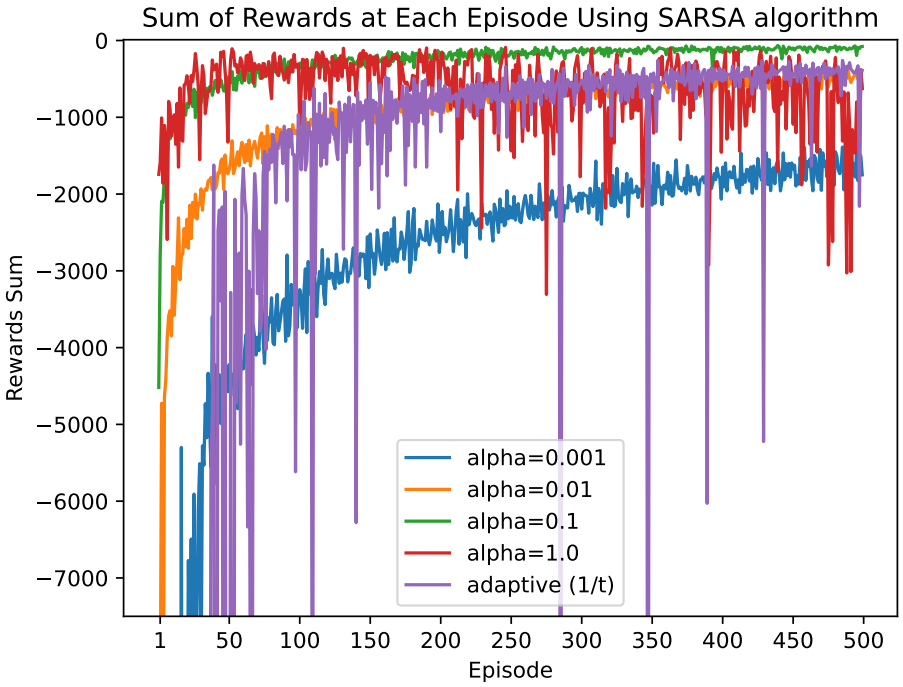

Figure 2: Sum of Rewards at Each Episode When α values are swept from 0.001 to 1.0 Using SARSA algorithm.

(b) Use the Q-learning algorithm and experiment with different step-size parameters.

Fig. 4 shows the algorithm followed to implement Q-Learning algorithm. γ was set to 1 and both of α and ϵ were swept across varied values resulting in Fig. 5 and Fig. 6.

ϵ was set to 0.1 in Fig. 5 where it shows very similar results to SARSA, however, Q-Learning is more stable. SARSA was suffering with many spikes while Q-Learning shows smoother learning curves with stable performance. Similar to SARSA smaller α values may not enable the agent to quickly learn and as Q-Learning is stable it was safely to increase α to 1.0 which provided the best solution in terms of performance and it also provided an optimal path in the maze.

α was set to 0.1 in Fig. 6 where it showed very similar results to SARSA's, hence, to avoid the redundancy kindly refer to Fig. 3 discussion.

Fig. 7 complement the discussion of these experimentation. A direct comparison between the two algorithms using the same exact hyper-parameters show that Q-Learning is more stable in performance as shown on the figure. Another aspect of comparison, not shown in this document, the computation complexity. When running both of the algorithms using the same hyper-parameters, SARSA always consumes more time compared to Q-Learning.

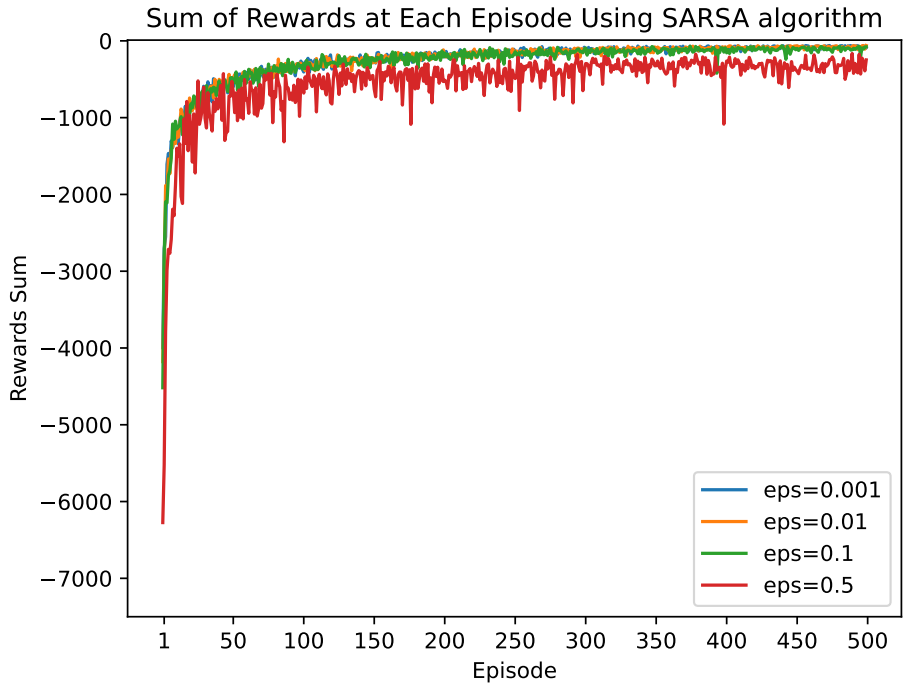

Figure 3: Sum of Rewards at Each Episode When ϵ values are swept from 0.001 to 0.5 Using SARSA algorithm.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 4: Q-Learning algorithm taken from chapter 6 of Sutton & Barto.

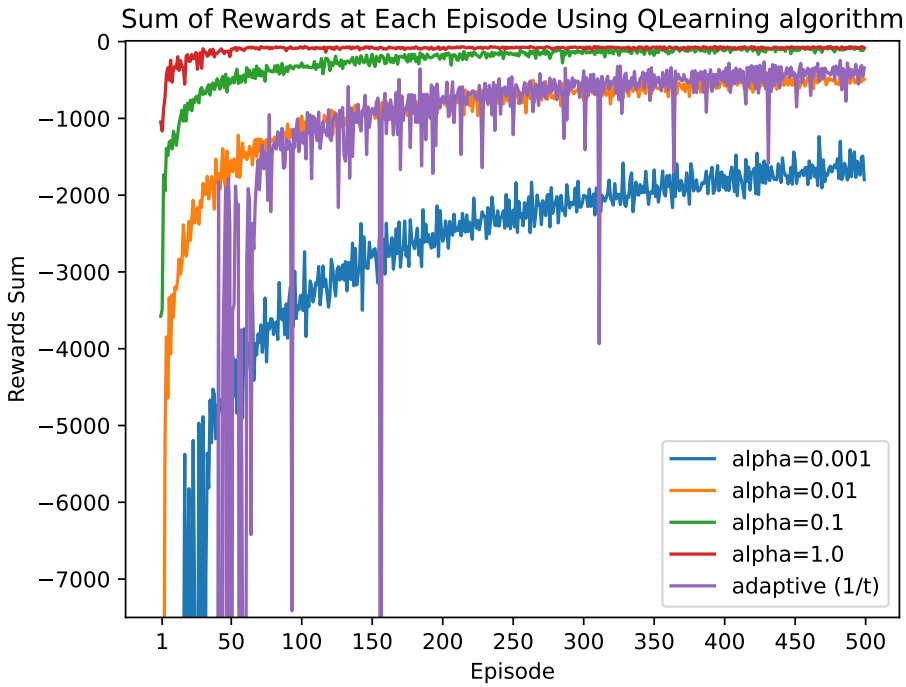

Figure 5: Sum of Rewards at Each Episode When α values are swept from 0.001 to 1.0 Using Q-Learning algorithm.

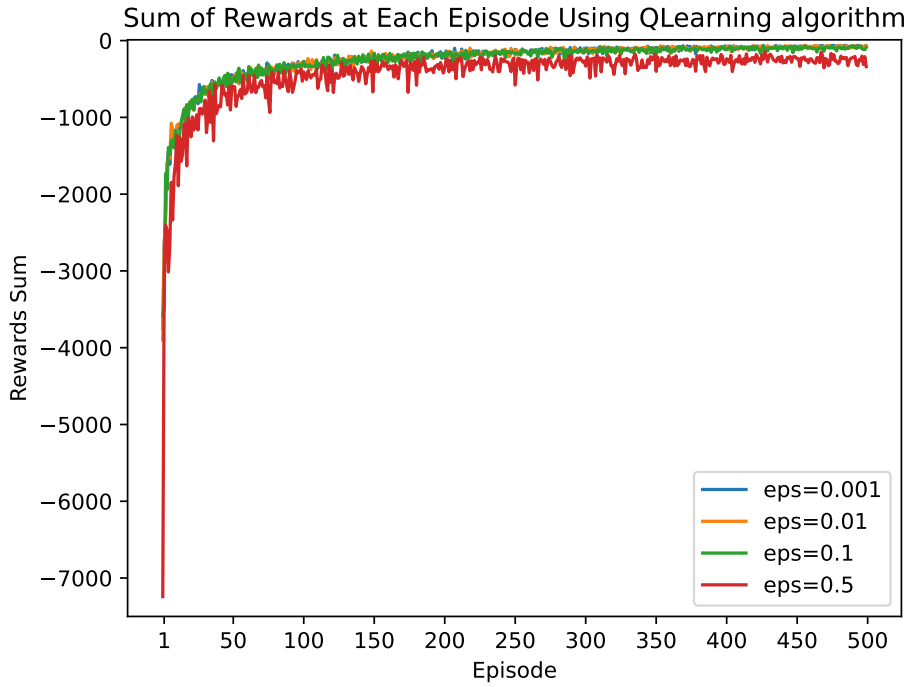

Figure 6: Sum of Rewards at Each Episode When ϵ values are swept from 0.001 to 0.5 Using Q-Learning algorithm.

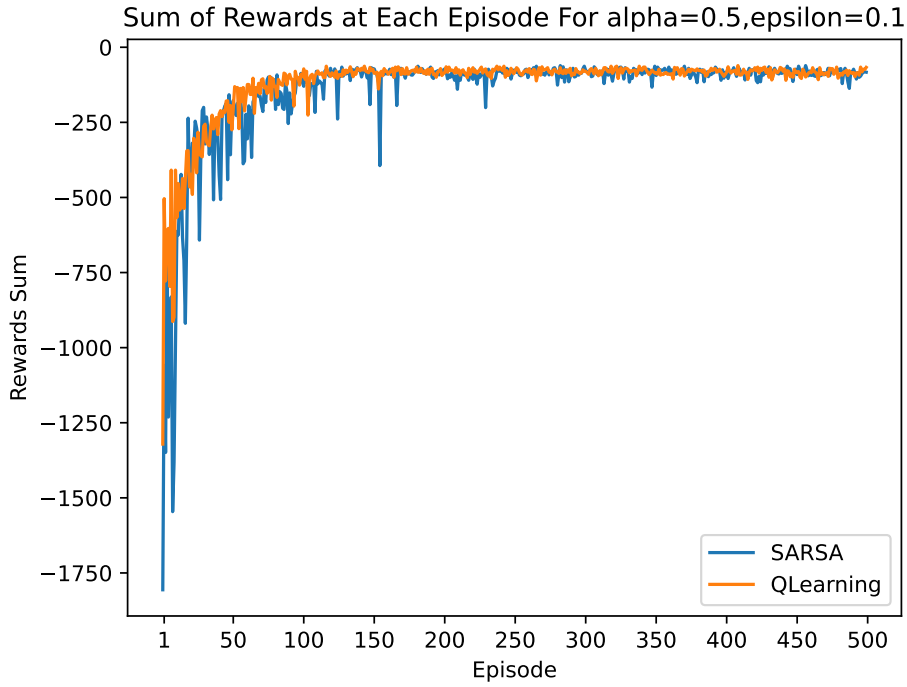

Figure 7: Sum of Rewards at Each Episode When comparing Q-Learning and SARSA algorithms.