HW1 - Dynamic Programming

Alen E. Golpashin[1]
University of Illinois, Urbana, Illinois, 61801, USA

The goal of this report is to implement five reinforcement learning algorithms in a "tabular" setting (i.e., assuming small finite state and action spaces). The author has implemented 5 different Reinforcement algorithms in two different environments. One to find an optimal trajectory to maximize reward, and another to balance a pendulum. The explored algorithms are value and policy iterations, SARSA, Q-Learning, and TD(0). The results of each respective algorithm are presented and discussed below.

I. Nomenclature

s	=	state
a	=	action
r	=	reward
γ	=	discount factor
n	=	number of episodes
G	=	return
Q	=	control value function
V	=	value function
P	=	transaction probabilities

II. Introduction

In this report, we will use model-based and non-modeled based reinforcement learning algorithms to approximate the value function and the policy of two problems formulated as Markov Decision processes. We define the parameters γ as the discount factor, ε as the ε-greedy parameter, n as the number of episodes, and α as the learning rate. A brief detail of the algorithms that are applied here are listed in the next section.
Section IV presents and discusses the results of different algorithms applied to the Gridworld environment [2]. Then, Section V presents and discusses the results obtained for the discrete pendulum environment.

III. Algorithms

We would like to briefly point out that both the value iterations and policy iterations are based on fixed-point map iterations of the Bellman operator [1]. While the value improves the policy at every iteration step of the value function, policy iteration allows the value function to converge and then performs a policy improvement step, This distinction can be seen later on, for example in Figure 2 when looking at the learning curve generated through a policy iteration algorithm.
On the other hand, when comparing the SARSA [1] and Q-learning [1] algorithms, we can see that Q-learning's update target $r + \gamma \max_a Q(s_1, a)$ is maximized (off-policy) with respect to the action a, while for SARSA, this maximization is not carried out (on-policy).

$$\mathrm{Q(s,a)} = \mathrm{Q(s,a)} + \alpha\left[r + \gamma \max_a Q(s_1, a) - Q(s, a)\right]$$

Further details on on-policy and off-policy algorithms are provided in [1].

[1] Graduate Student, Department of Aerospace Engineering.

IV. Gridworld Results

For this environment, we have applied the policy iteration [1], value iteration [1], SARSA [1], and Q-learning [1] to either estimate the value function, or to estimate the value function as well as the optimal policy. Additionally, the TD(0) algorithm [1] is applied to learn the value function associated with the optimal policy produced by SARSA and Q-learning. For this experiment, the following parameter values were chosen:

Table. 1 The chosen Gridworld parameters.

Parameters	Values
γ	0.95
ε	0.05
n	5000
α	$1/n$

For this environment, the state s of the Gridworld was discretized as follows:

Table. 2 The discretized Gridworld state-space.

$s = 0$	$s = 1$	$s = 2$	$s = 3$	$s = 4$
$s = 5$	$s = 6$	$s = 7$	$s = 8$	$s = 9$
$s = 10$	$s = 11$	$s = 12$	$s = 13$	$s = 14$
$s = 15$	$s = 16$	$s = 17$	$s = 18$	$s = 19$
$s = 20$	$s = 21$	$s = 22$	$s = 23$	$s = 24$

A. Value and Policy Iterations

For both the policy and value iterations, we set the value function convergence error threshold as 1e-3. That is, we declare the fixed-point mapping of the value function to have converged if the error at the iteration was equal or lower than the specified threshold. The value function resulting from the value iteration algorithm was computed to be:

Table. 3 The value iteration converged value function at each respective grid-state.

41.99151532	44.20235168	41.99223409	39.20235168	37.24223409
39.89193956	41.99223409	39.89262239	37.89799127	36.0030917
37.89734258	39.89262239	37.89799127	36.0030917	34.20293712
36.00247545	37.89799127	36.0030917	34.20293712	32.49279026
34.20235168	36.0030917	34.20293712	32.49279026	30.86815075

Note that each entry of Table 3 corresponds to an entry of a Table 2. For example, the value function at state $s = 1$ was found to be 44.20235168 using the value iteration algorithm. The residual error of the fixed-point mapping was 0.00093, which is smaller than the specified threshold. On the other hand, using the policy iteration algorithm, the value function converged to the following values at each state s:

Table. 4 The policy iteration converged value function at each respective grid-state.

41.99336527	44.20385847	41.99366555	39.20385847	37.24366555
39.89369701	41.99366555	39.89398227	37.89928316	36.004319
36.004319	39.89398227	37.89928316	36.004319	34.20410305
36.00406155	37.89928316	36.004319	34.20410305	32.4938979
34.20385847	36.004319	34.20410305	32.4938979	30.869203

Next, we look at the learning curves resulting from each algorithm. For policy iteration and value iteration, we plot the mean of the value function $\frac{1}{25}\sum_{s=1}^{25} V(s)$ versus the number of value iterations. More specifically for policy iteration, "the number of value iterations" is the number of iterations spent on policy evaluation (which dominates the computation time). For value iteration, "the number of value iterations" is synonymous with the number of iterations of the algorithm. Below, the value iteration learning can be seen to stabilize slightly above mean value of 35.

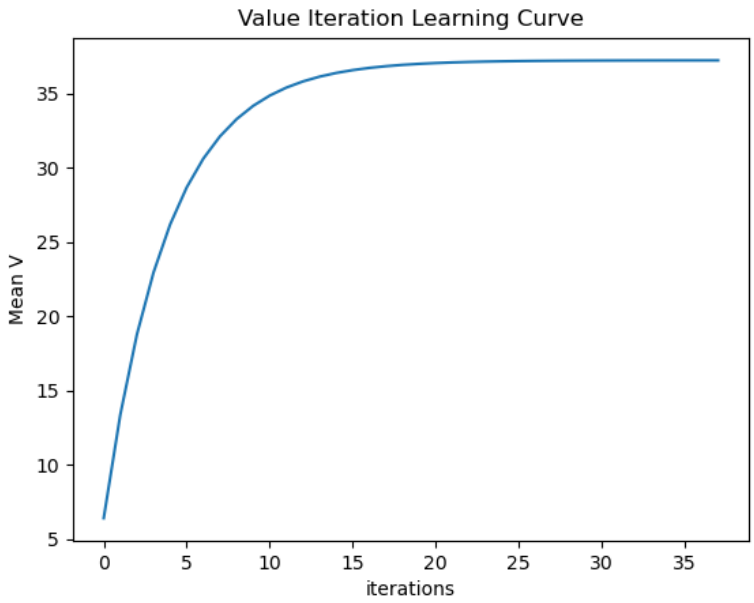

Fig. 2 Value iteration learning curve.

Looking at the policy iteration learning curve, we can see where each policy improvement step was performed. This step corresponds to where the curve starts to increase after each time it had initially stabilized. Moreover, the initial negative slope of the learning curve corresponds to the bad initial guess for both the value function V and the optimal policy π.

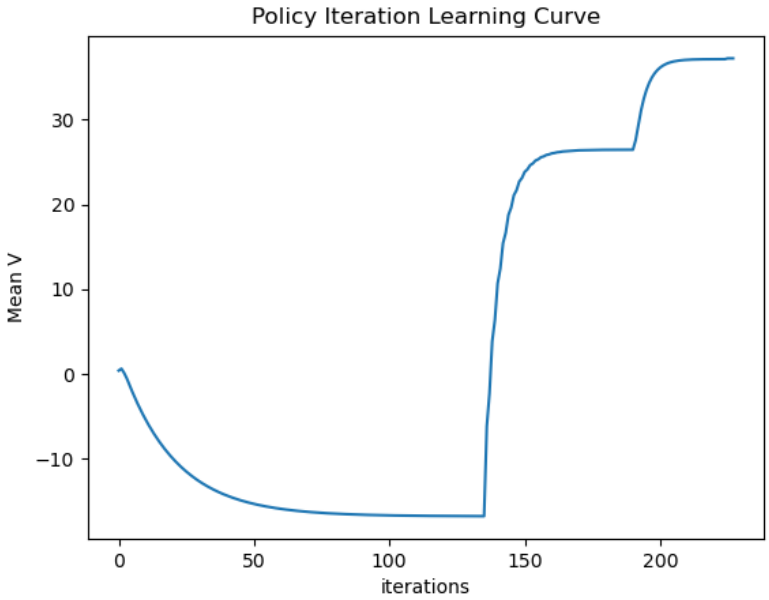

Fig. 2 Policy iteration learning curve.

But it is important to note that both the policy and value iterations learning curves have stabilized at a similar final value. This is rather reassuring since it means that both algorithms have converged to the vicinity of the actual value function.

Next, we produce a plot of an example trajectory for each trained agent. That is, we plot the state, action, and reward tuples (s, a, r) over every grid-state s. For the value iteration, the example trajectory is shown in Figure 3.

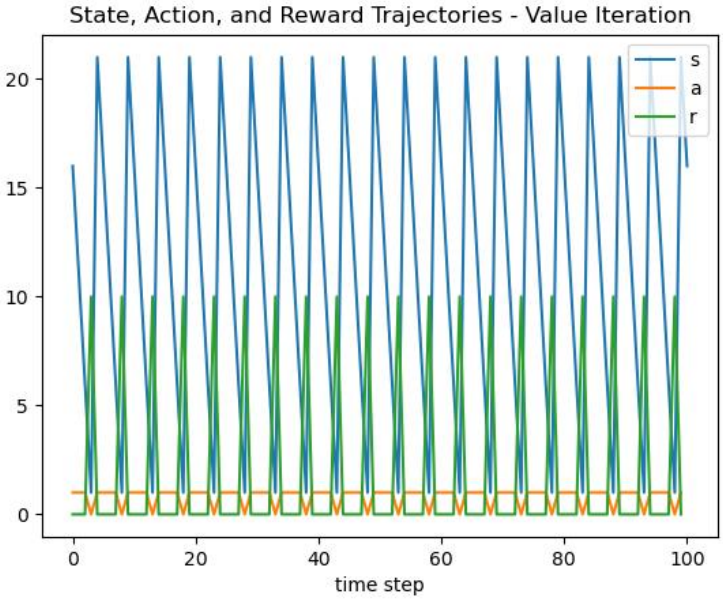

Fig. 3 Value Iteration example trajectory.

Similarly, we visualize the example trajectory resulting from the policy iteration algorithm in Figure 4. It is notable to mention that in both of these example trajectories, the state s is oscillating between the states $s = 1$ and $s = 21$

because the optimal policy is to always go to the state $s = 1$ which yields the highest reward. The reward trajectory also exhibits this behavior.

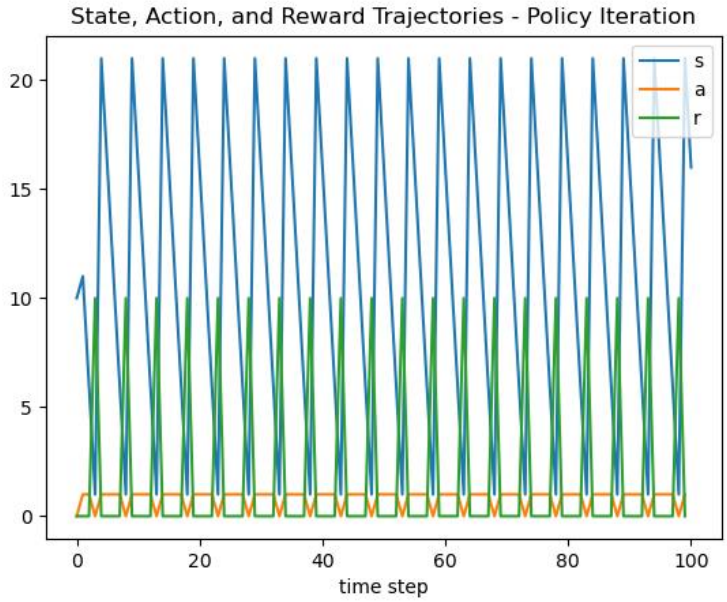

Fig. 4 Policy Iteration example trajectory.

Next, we will investigate the same problem, but using the SARSA algorithm to estimate the optimal policy.

B. SARSA

Here, we outline the results of the SARSA algorithm which is an on-policy algorithm. In table 5, we show the obtained optimal policy π using the $n = 5000$ iterations. Below, action to go right is noted as *R*, up is *U*, Down is *D*, and Left is *L*. More specifically, $\pi = [R\ R\ R\ R\ L\ R\ U\ R\ U\ L\ R\ U\ R\ U\ L\ R\ R\ R\ U\ L\ R\ U\ U\ U\ U]$

Table. 5 Optimal policy obtained through the SARSA algorithm

R	***R***	***R***	***R***	***L***
R	***U***	***R***	***U***	***L***
R	***U***	***R***	***U***	***L***
R	***R***	***R***	***U***	***L***
R	***U***	***U***	***U***	***U***

Next, we look at the learning curve generated through the SARSA algorithm. This curve, shown in Figure 5, is defined as the return G versus the number of episodes. Specifically, G is the discounted sum of rewards r over the infinite horizon.

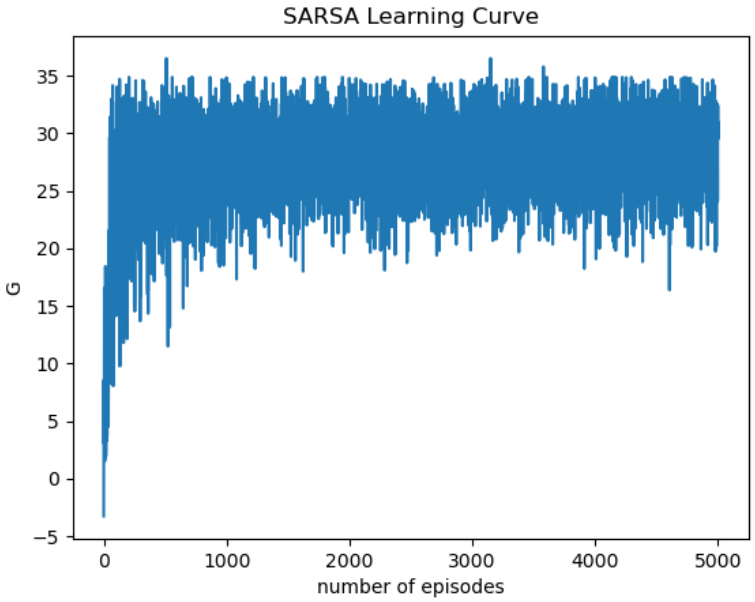

Fig. 5 Gridworld SARSA learning curve.

Inspecting this plot, we see that it has leveled off around the average value of $G = 28$. This is consistent with the previous form of the learning curve generated through the value and policy iterations. An example trajectory is also generated and displayed below:

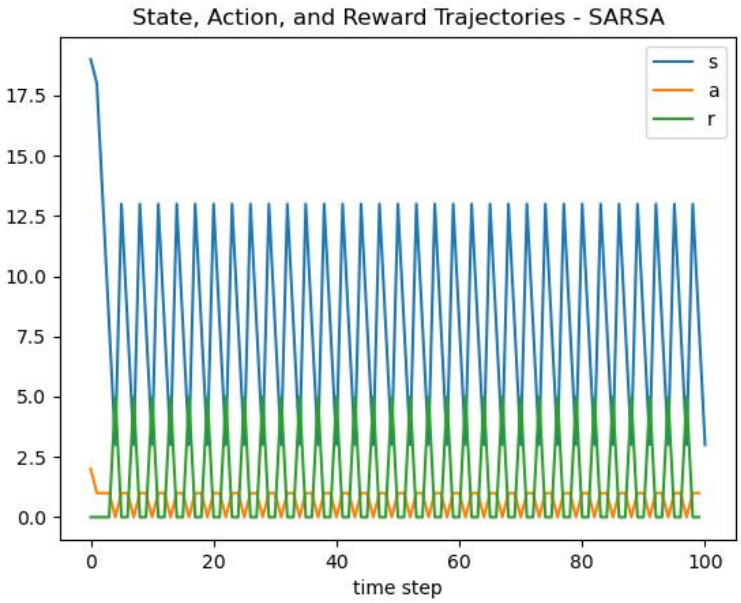

Fig. 5 SARSA example trajectory.

C. Q-learning

The Q-learning optimal policy is shown below over the grid. More specifically, the value of π is found to have converged to $\pi = [R\ R\ L\ R\ L\ R\ U\ R\ U\ L\ R\ U\ R\ U\ L\ R\ R\ R\ U\ L\ R\ U\ U\ U\ L]$ using the Q-learning algorithm.

Table. 6 Optimal policy obtained through the Q-learning algorithm

R	***R***	***L***	***R***	***L***
R	***U***	***R***	***U***	***L***
R	***U***	***R***	***U***	***L***
R	***R***	***R***	***U***	***L***
R	***U***	***U***	***U***	***L***

Comparing the optimal policies of the Tables 5 and 6, we see that the Q-learning algorithm has converged to a better (more optimal) policy. This is evident by looking at state $s = 2$, where Q-learning has correctly specified a $a = L$ action, while SARSA has converted to a suboptimal policy where $a = R$ at $s = 2$.

The learning curve resulting from the Q-learning algorithm seems similar to the SARSA-generated learning curve. However, it can be seen that the average of the asymptote of the curve is closer to $G = 29$, which indicates that Q-learning may have maximized the return better over the infinite horizon. We could relate this and the better performance in finding a more optimal policy to the fact that Q-learning algorithm is an off-policy algorithm. This means that the target Q is maximized with respect to the action a at every iteration.

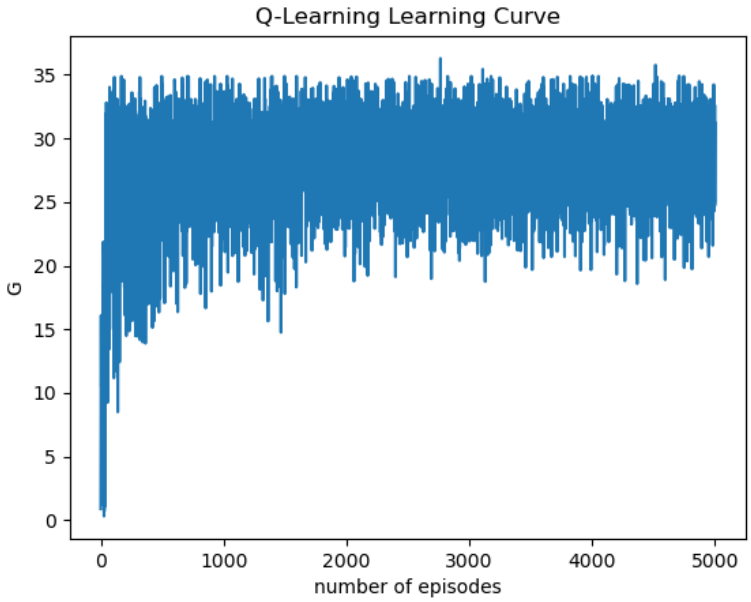

Fig. 6 Gridworld Q-learning learning curve.

Next, generating the sample trajectory, we see the familiar oscillatory behavior:

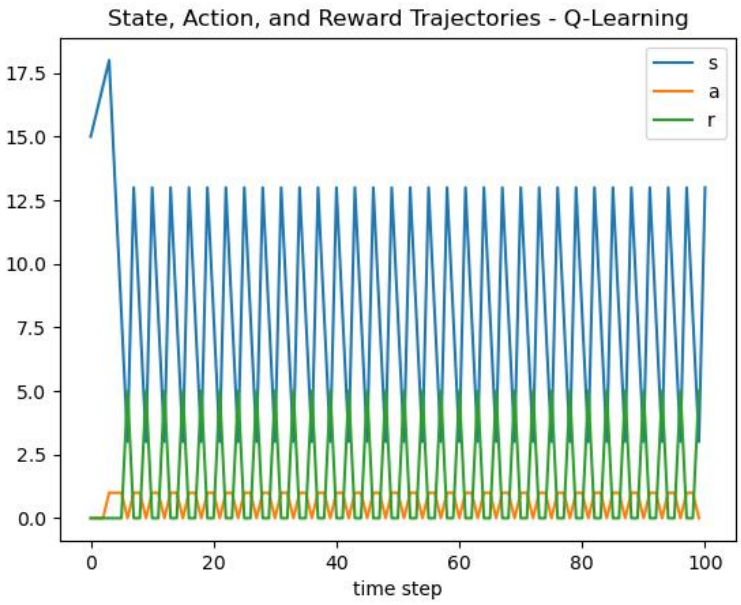

Fig. 7 Q-learning example trajectory.

D. TD(0) Function Estimation

In this section, we will look at the learned (estimated) value function for both the SARSA and Q-learning algorithms.

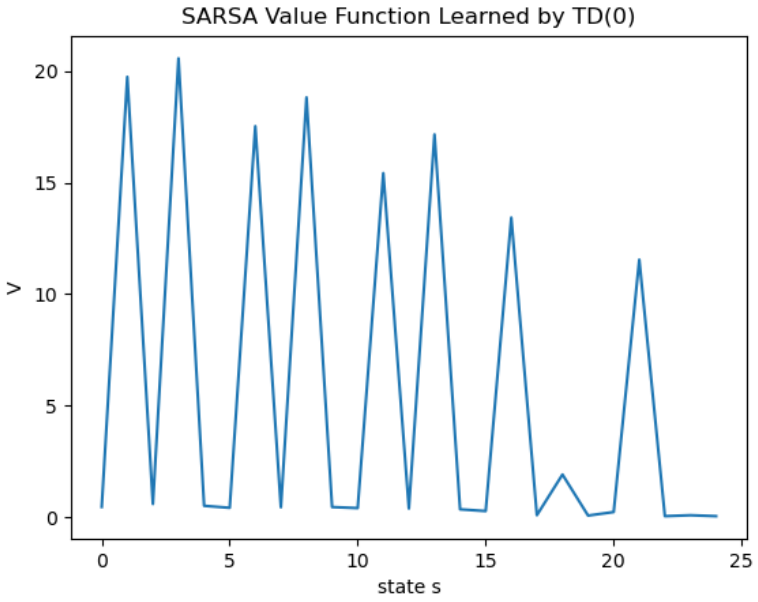

Fig. 8 SARSA TD(0) learned value function over the states.

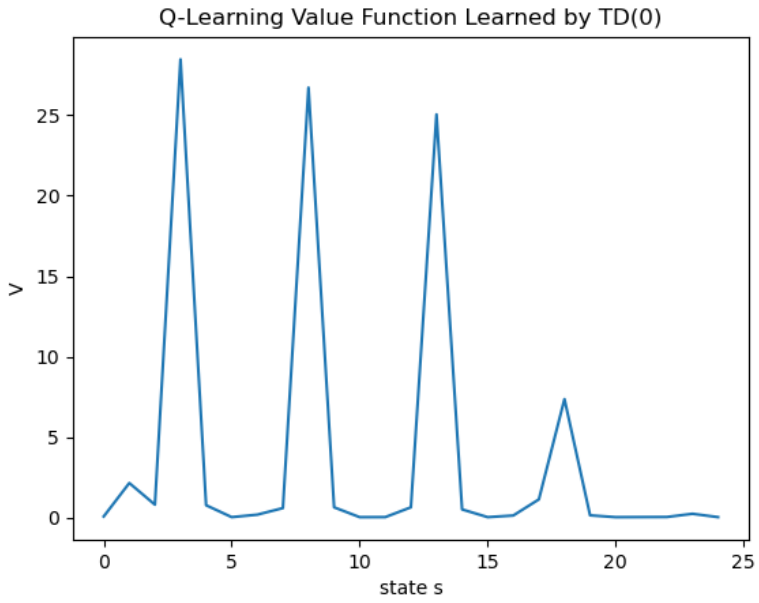

Fig. 9 Q-learning TD(0) learned value function over the states.

The next to figures show the corresponding converged optimal policy for each algorithm (SARSA and Q-learning value functions learned through TD(0)):

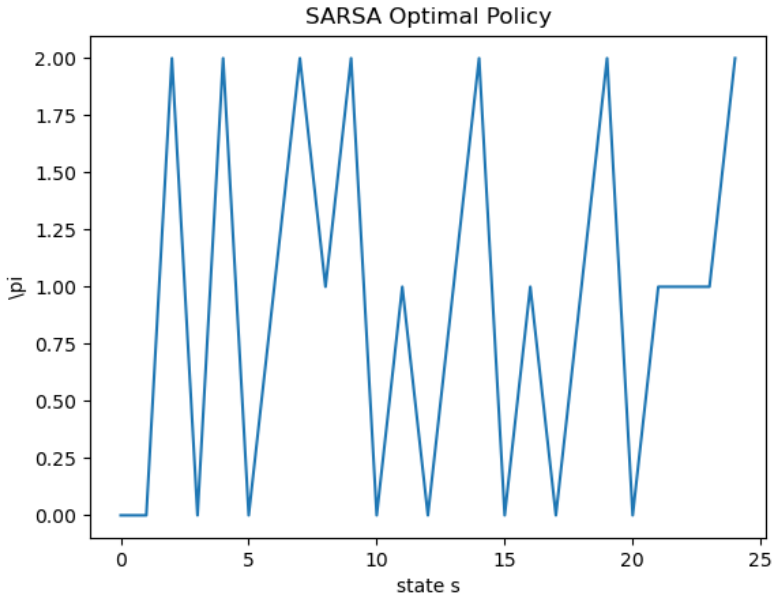

Fig. 10 SARSA optimal policy over the states.

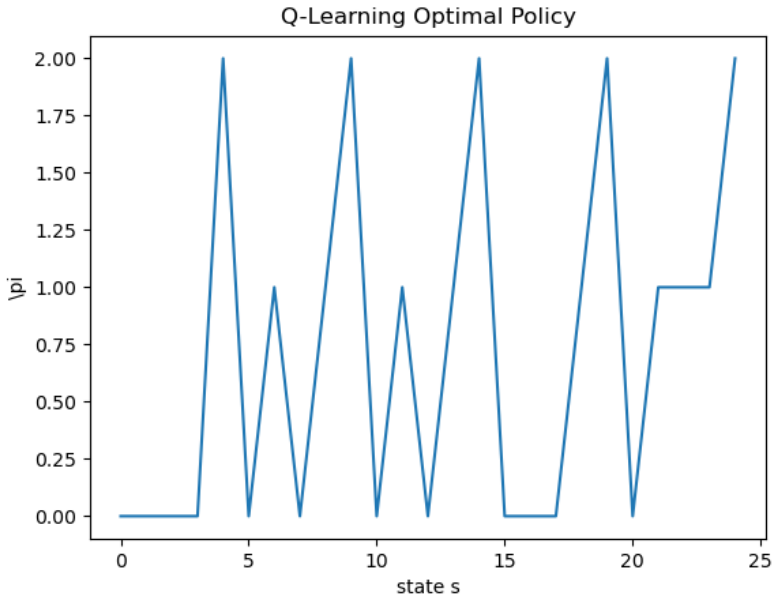

Fig. 11 Q-learning optimal policy over the states.

E. Experiment Varying ε and α

In this section, we will focus on the learning curves of SARSA and Q-learning. However, here we will also vary the ε and the α parameters. Specifically, in this experiment, we have chosen a minimum α value of 0.05 and a maximum value of 0.8. We have generated 5 equidistant αs within this range. Similarly, we have chosen a minimum value of 0.001 for the ε with a maximum value of 0.7. There are 5 ε values which their learning curves we have generated here. Below, show the learning curves with the varying parameters.

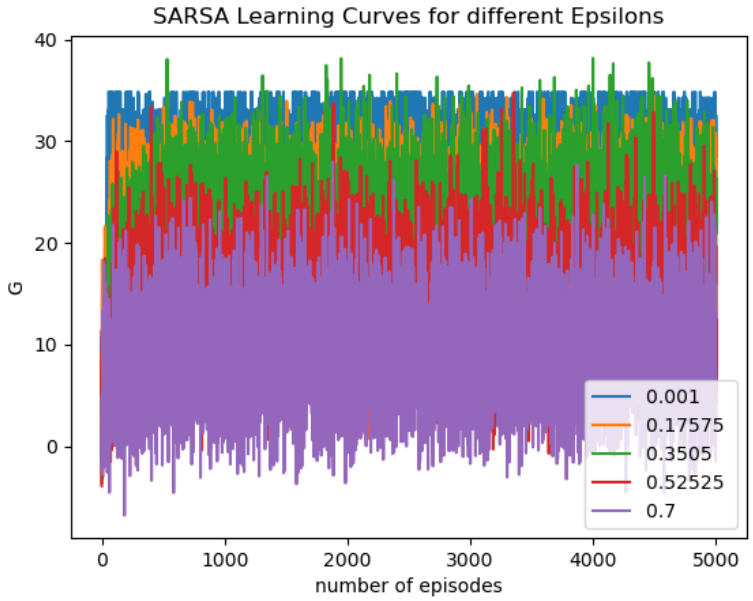

Fig. 12 SARSA value function varying ε.

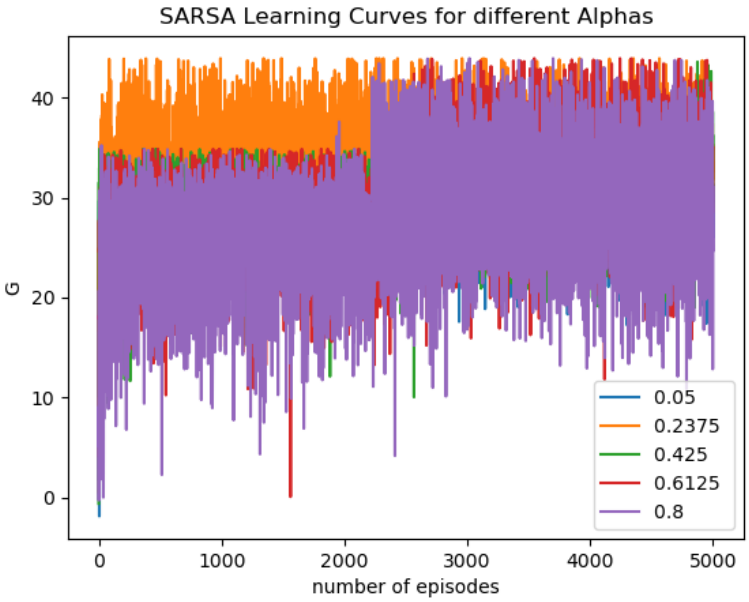

Fig. 13 SARSA value function varying α.

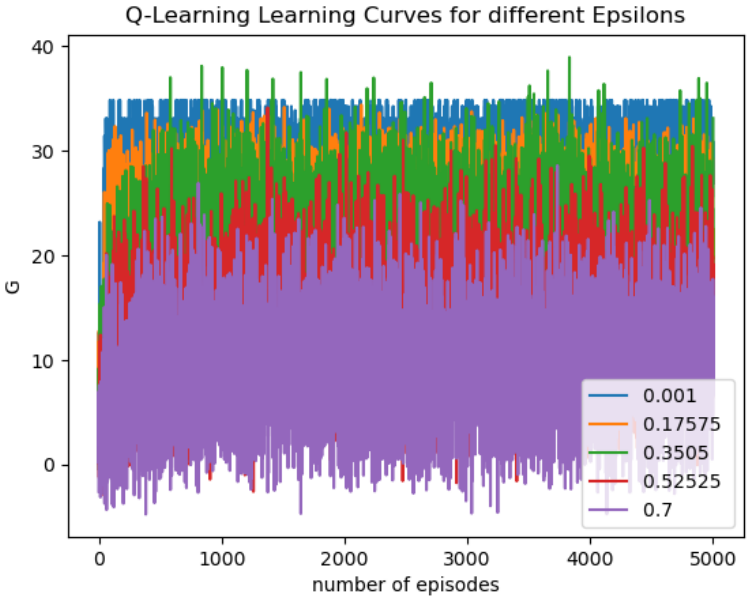

Fig. 14 Q-learning value function varying ε.

From Figure 12 and 14 we can see that a smaller ε leads to a higher learning curve. This may be because a lower ε leads to the random policy being selected fewer times, thus, for the same number of iterations, we may end up with a higher learning curve.

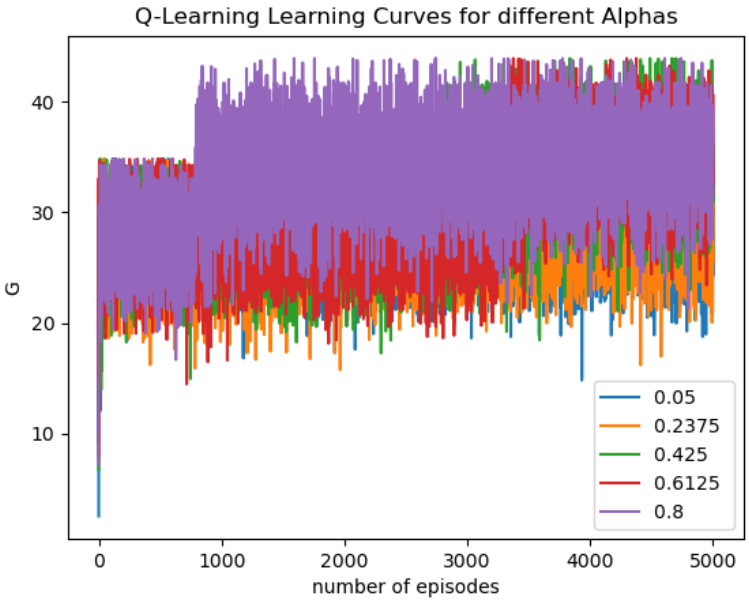

Fig. 15 Q-learning value function varying α.

Also, we may conclude that a higher alpha leads to a higher variance in the learning curve, Looking at Figures 13 and 15, we can see that higher α, i.e $\alpha = 0.8$ leads to a higher bandwidth of the learning curve. Thus, a smaller learning α may give more consistent results.

V. Discrete Pendulum

The goal of this section is to balance a simple pendulum with discretized state and action spaces, for which an explicit model is not available. In contrary to the Gridworld environment, the transition probabilities p or the state rewards are not available. To use the pendulum environment, both the state space (θ, $\dot{\theta}$) and the action space are discretized with 31 grid points in each dimension, for a total of 31 x 31 states and 31 actions. To speed up computation, we choose coarser grids such as 15 x 21 states and 31 actions or 11 x 51 states and 21 actions.

A. SARSA and TD(0)

We run the TD(0) algorithm in conjunction with SARSA. That is, we call the SARSA algorithm once to yield an optimal π, and then using the policy, we compute the value function approximation. Once the value function is obtained, we plot it over the transformed state-space s.

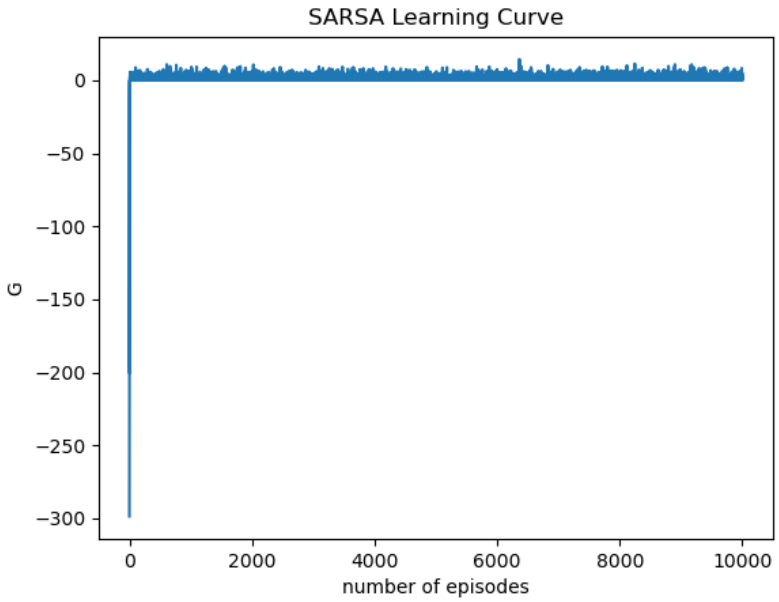

Fig. 16 Pendulum SARSA learning curve – first 3000 episodes.

To start with, we plotted the learning curve and the trajectories of a simulation instance with 15 θ-nodes and 21 $\dot{\theta}$-nodes. The other parameters were $\varepsilon = 0.05$, $n = 3000$, and $\alpha = 1/n$. After the simulation was done, it was obvious that the coarse grid chosen, and the lower number of iterations weren't able to balance the pendulum. In Figure 17 it can be seen that θ periodically visits the boundaries of 3.14 and -3.14. For this reason, we begin a new simulation instance, where we select the nodes as 11 θ-nodes and 51 $\dot{\theta}$-nodes, and 21 action space nodes. Moreover, we save the final Q and the evaluated policy π vector into a file, and then call that file every time when add more iterations to be run. Thus, the cumulative number of iterations was possible to be added to. However, because of a larger number of nodes, the processing speed of the code was very slow.

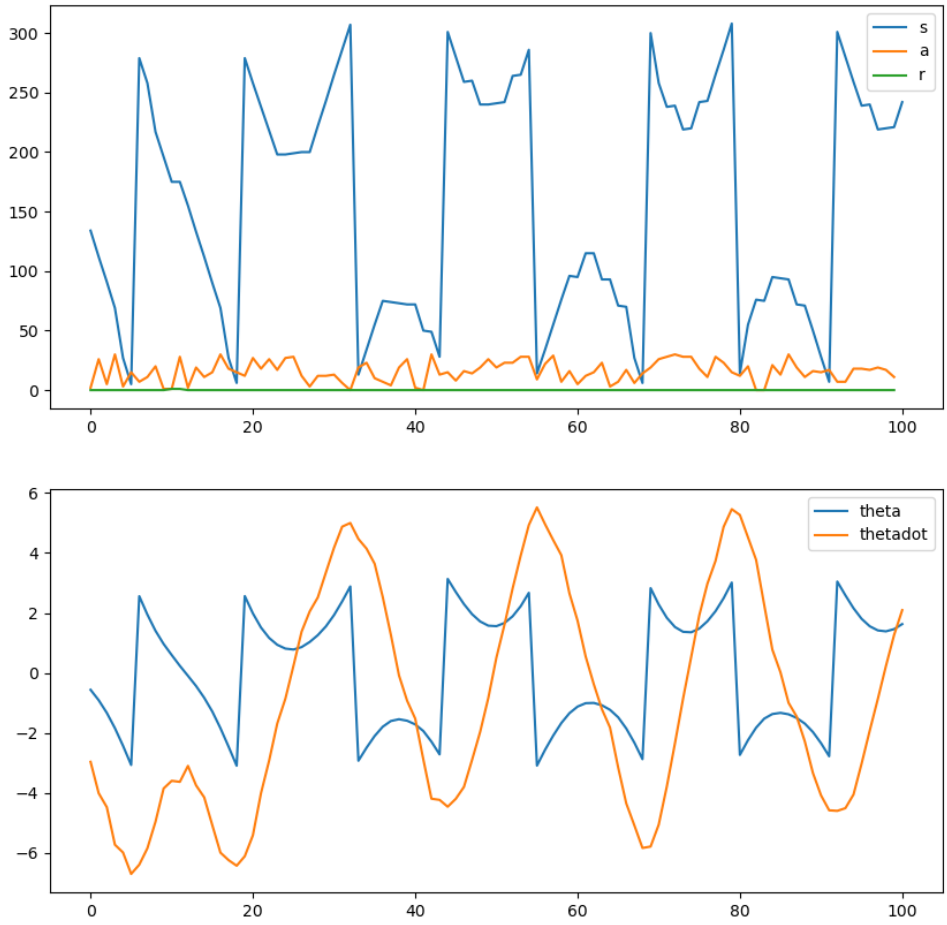

Fig. 17 SARSA pendulum trajectory – first 3000 episodes.

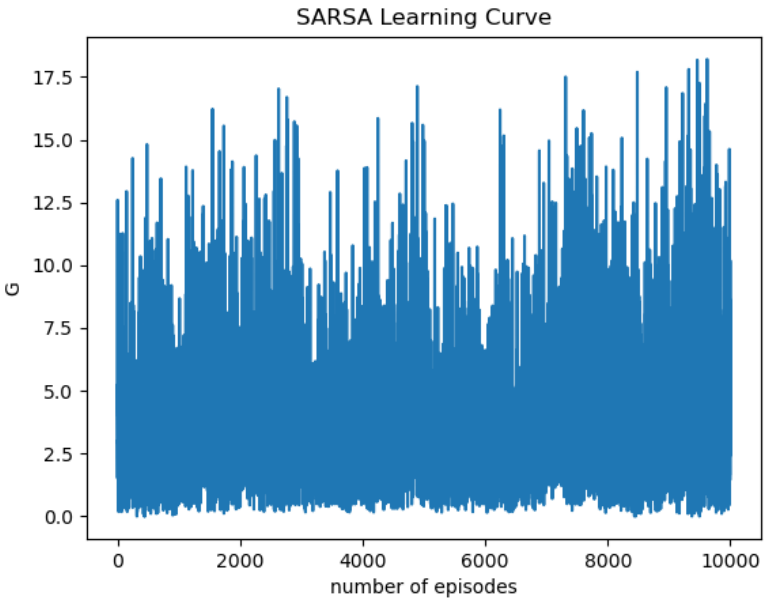

Fig. 18 Pendulum SARSA learning curve.

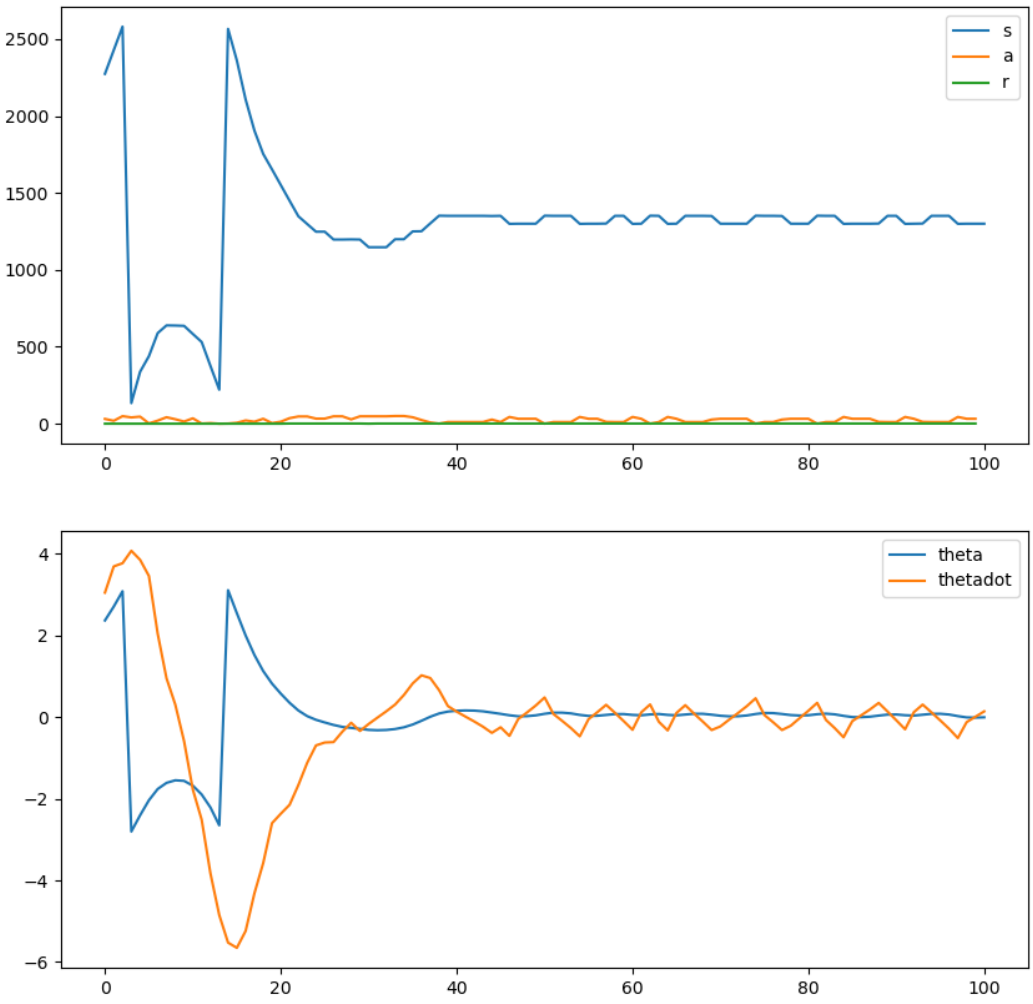

Fig. 19 SARSA pendulum trajectory.

B. Q-learning and TD(0)

The learning curves for the Q-learning algorithm are presented here. The steep learning slope can be seen here again.

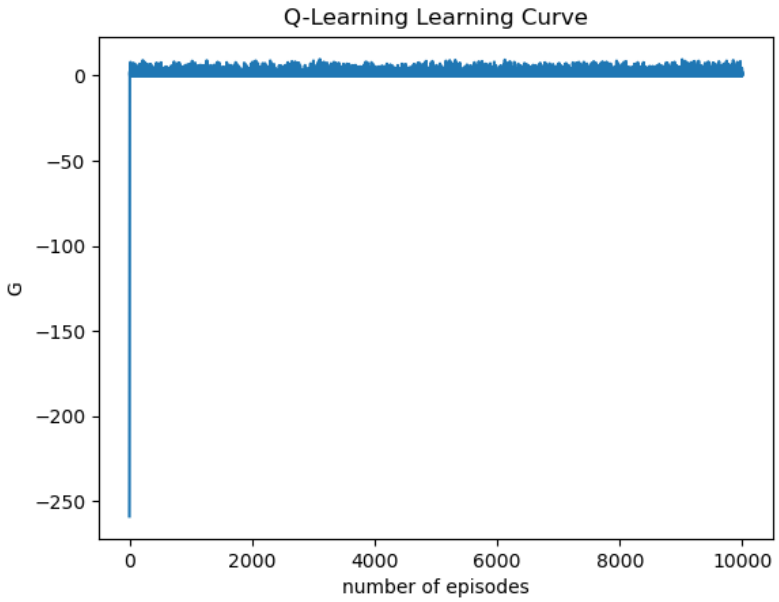

Fig. 20 Pendulum Q-learning learning curve - first 3000 episodes.

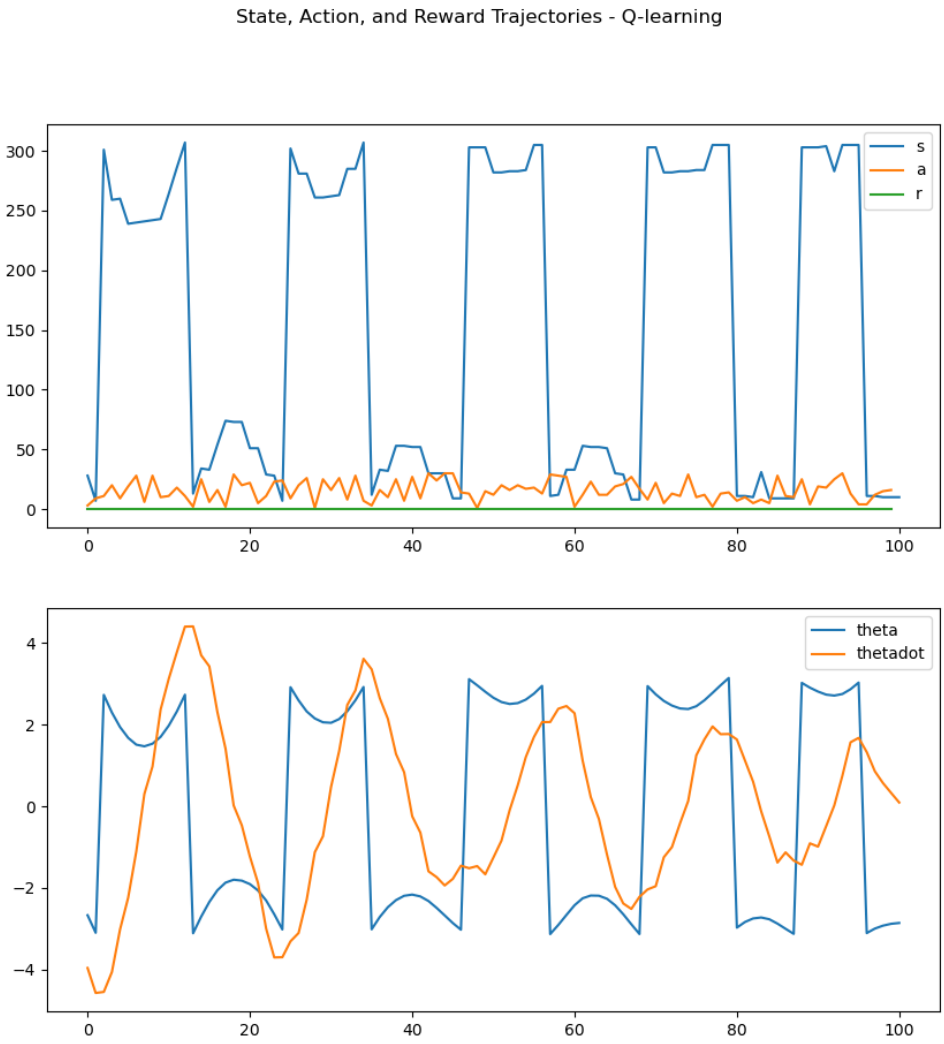

Fig. 21 Q-learning pendulum trajectory - first 3000 episodes.

To improve the results, we decided to increase the grid structure to a 51x51x51 number of nodes. We also changed the ε to 0.5 and α to 0.57 (based on the results of Section V.D) and increased the number of episodes to 5000. This helped the pendulum to be stabilized at the origin. Both Figures 19 and 23 show this stabilization of θ.

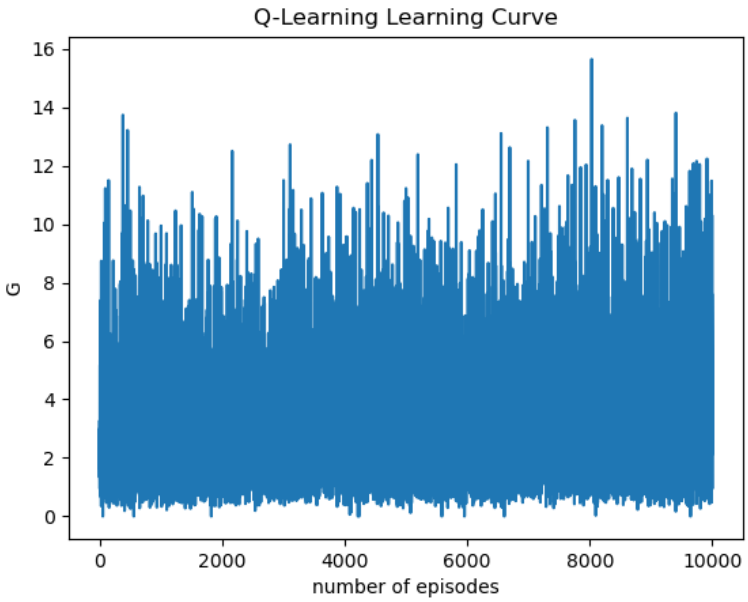

Fig. 22 Pendulum Q-learning learning curve.

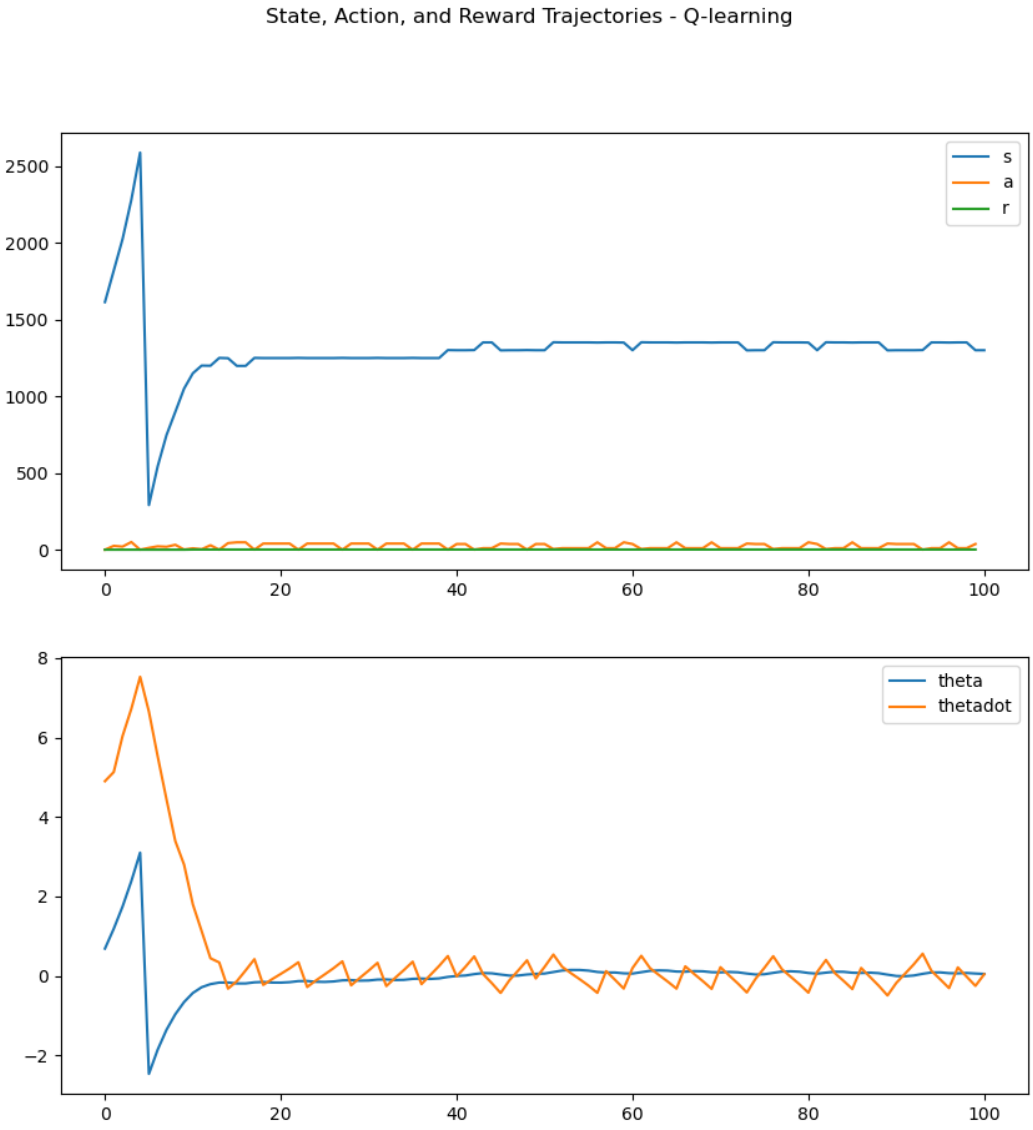

Fig. 23 Q-learning pendulum trajectory.

Aside from the coding typos, the increase in the number of nodes and episodes seems to have helped the SARSA and Q-learning algorithms to converge to an optimal policy. Although, the convergence of the Q-learning seems to be more rapidly. There is an initial delay of 20 seconds before θ stabilization when SARSA algorithm was used; see Figure 19.

From this, we conclude that a higher number of trajectories may help stabilize the pendulum using both the Q-learning and the SARSA algorithms. Empirical data suggest that increasing the number of nodes may also help in finding a stabilizing policy, but this will increase the computation time significantly. The increase in node numbers directly increases the dimensionality of the problem.

C. TD(0) Function Estimation

Here, we plot the learned/estimated value functions by TD(0) algorithm over the transformed states s.

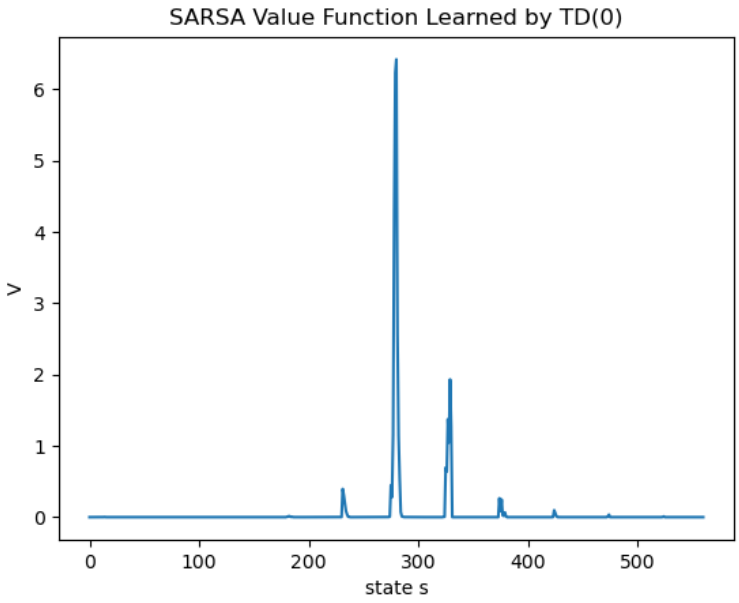

Fig. 24 SARSA TD(0) learned value function over the states.

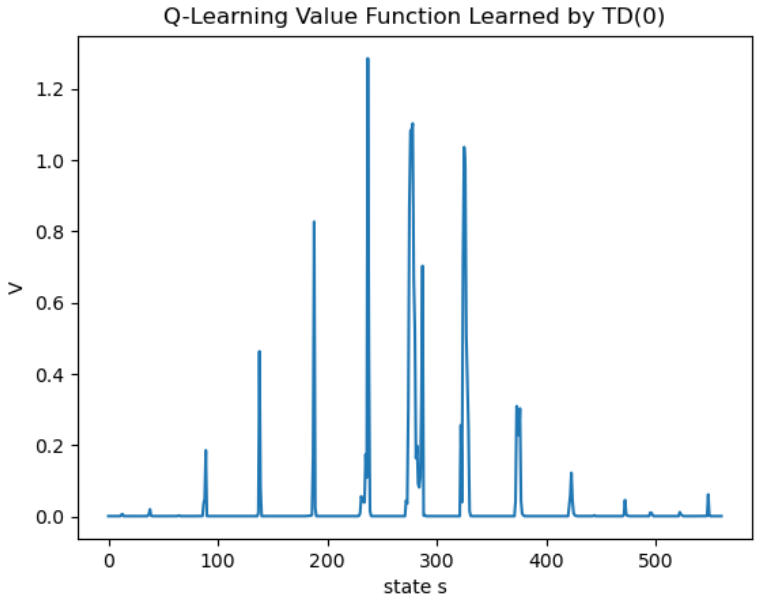

Fig. 25 Q-learning TD(0) learned value function over the states.

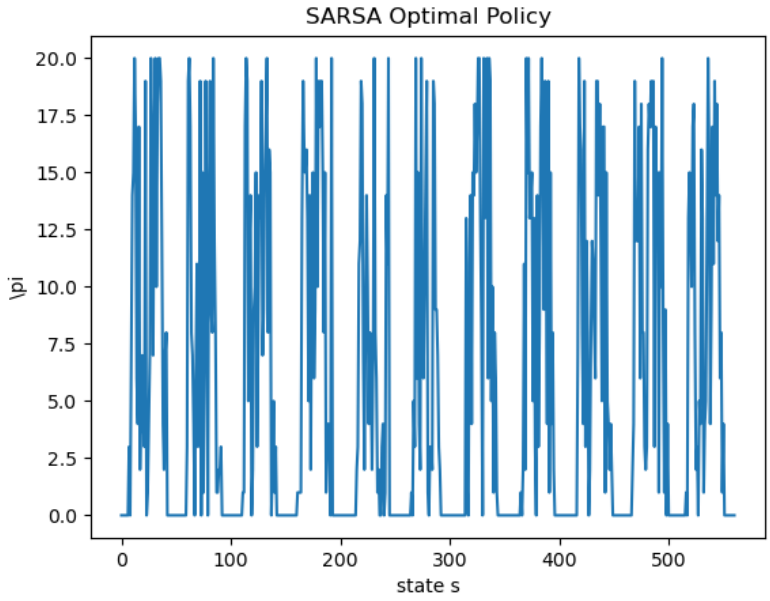

Fig. 26 SARSA optimal policy over the states.

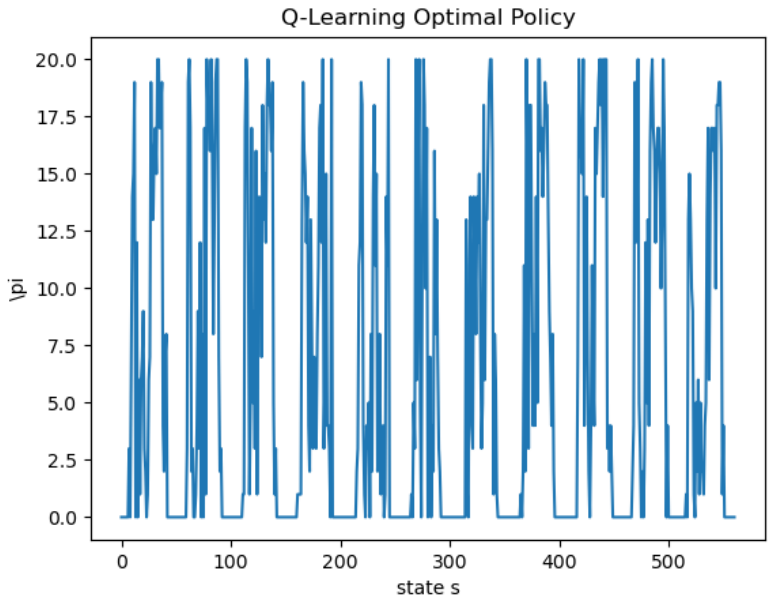

Fig. 27 Q-learning optimal policy over the states.

We should note that comparing Figures 24 and 25, we see that the Q-learning algorithm is relatively doing a better job in maximizing the value function over every state s.

D. Experiment Varying ε and α

Below, we plot the learning curves for the two algorithms of SARSA and Q-learning, while varying one of the ε or α parameters at a time. The ε is varied from 0.001 to 0.2, and α is varied from 0.1 to 0.8.

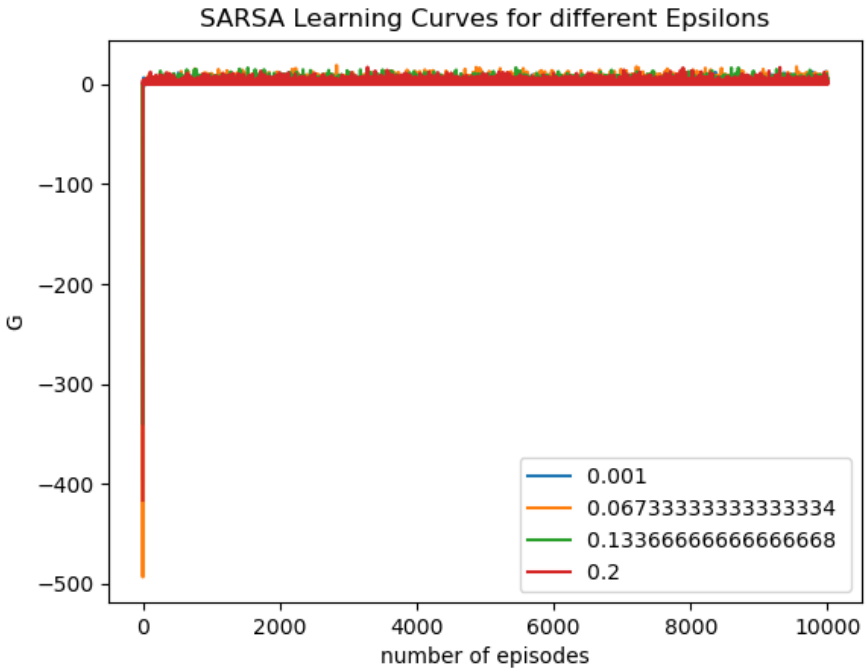

Fig. 28 SARSA value function varying ε.

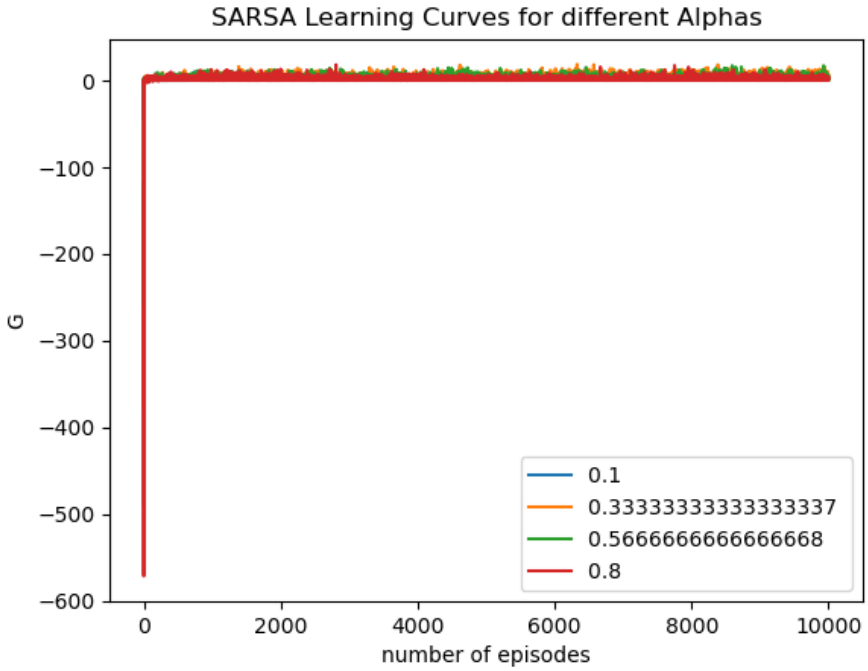

Fig. 29 SARSA value function varying α.

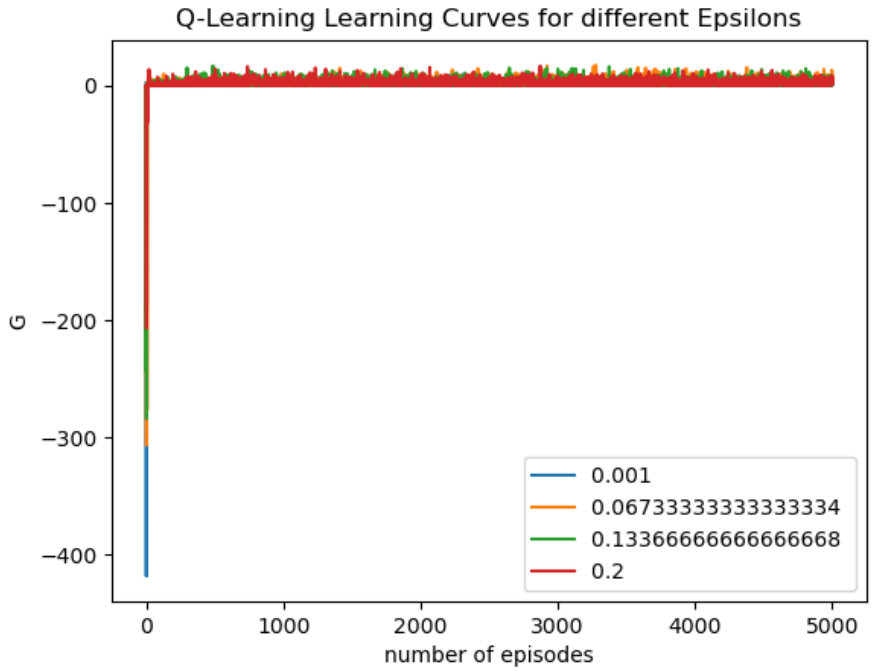

Fig. 30 Q-learning value function varying ε.

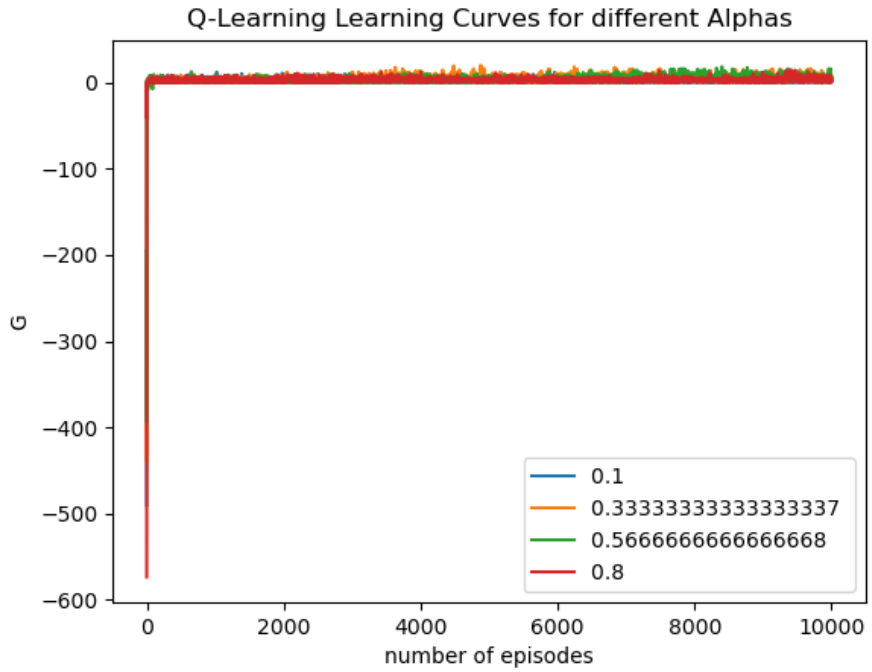

Fig. 31 Q-learning value function varying α.

The learning curve have a very steep learning curve slope. We also notice that the value functions start in the negative thousands which means that the initially the evaluated policies were very poor. From Figures 28 and 30, we see that an ε close to 0.13 has the highest outliers which means that it could lead to a higher learning curve. Also, we see that the higher the ε goes (i.e. $\varepsilon = 0.8$), the higher the variance in the learning curve becomes.

In addition, from Figures 29 and 31, we see that an alpha of $\alpha = 0.56$ leads to a higher learning as the number of iterations get larger. However, it also can be seen that a higher α may lead to a higher variance in the learning curve.

VI. Conclusion

In this report, we trained agents in two environment of the Gridworld and the discrete pendulum. In the Gridworld, we saw that higher number of episodes lead to better results, but this improvement starts to decrease as the number of episodes goes up. In general, we saw that the Q-learning algorithm had a better performance. Although, using any of the algorithms, the agents did converge to the optimal policy. However, this was not as easy in the discrete pendulum environment. The initial experiments with 3000 iterations and lower grid points showed that a lower number of episodes do not lead to a policy that can stabilize the pendulum. We also saw that the SARSA and the Q-learning algorithms were both sensitive to the ε and α choices. For this reason, these values were chosen based on observation from Section V D. Moreover, because the pendulum was model-free, the dynamics of the pendulum were discretized on a grid. The greater number of nodes would lead to lower error and a policy that may be more optimal, but this was increasing the computation time from minutes to hours. In the second experiment with a larger number of episodes (5000 episodes), we saw that the pendulum was successfully stabilized. The Q-learning algorithm did obtain the optimal policy earlier than the SARSA algorithm and was able to stabilizes pendulum faster. Our method of inputting previous saved value functions and policies as initial guesses did also seem to be making a big difference. Starting at the previous converged optimal policy and approximated Q, allowed us to modify parameters but not be required to run a larger number of episodes. The author acknowledges that the typo in using the optimal policy for plotting did contribute to the initial report lacking the convergence of the pendulum. This code typo was corrected for the results presented in this version of the report.

Acknowledgments

The author acknowledges the help of Prof. Huy Tran, and other classmates in completion of this report.

References

[1] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[2] Golpashin, A. (2023) train_gridworld.py. https://github.com/uiuc-ae598-rl-2023-spring/hw1-dp-golpashin
[3] Golpashin, A. (2023) train_pendulum.py. https://github.com/uiuc-ae598-rl-2023-spring/hw1-dp-golpashin