AE 598 RL HW 1: Dynamic Programming

Grace E. Calkins*

I. Introduction

This writeup presents the results of Homework 1 for Reinforcement Learning (AE 598 RL), where I implemented policy iteration, value iteration, SARSA, Q-learning, and TD(0) algorithms. Each section of this report details a learning algorithm and presents the results of that algorithm for the required environments. TD(0) implementation is discussed in its own section, but the results of TD(0) are presented in the SARSA and Q-learning sections. Discussion of the results of each algorithm for each environment and comparison of different methods are included in the Results sections of each algorithm.

My code was structured with each algorithm in its own `.py` file which was imported to `train_girdworld.py` and `train_pendulum.py` to train each environment with the different algorithms. Helped functions, like a function to select an action following ϵ-greedy and plotting functions, are also in separate files. The `main` function in the training files has a series of boolean tags at the beginning which can be changed to run different algorithms.

II. Policy Iteration (PI)

A. Implementation

My implementation of policy iteration followed the algorithm in Sutton and Barto. I divided policy evaluation and policy improvement into two separate methods to improve readability.

B. PI Gridworld Results

When implemented in the gridworld environment, this algorithm converged in 4 total PI steps with a tolerance of $1e-10$. Each step of policy evaluation converged in fewer steps than before and resulted in a higher reward function as seen in Figure 1, showing the algorithm is learning the optimal policy by obtaining a higher value function over time. As expected, the optimal policy is to go to state 0 in as few actions as possible and teleport to state 21 as many times as possible per episode.

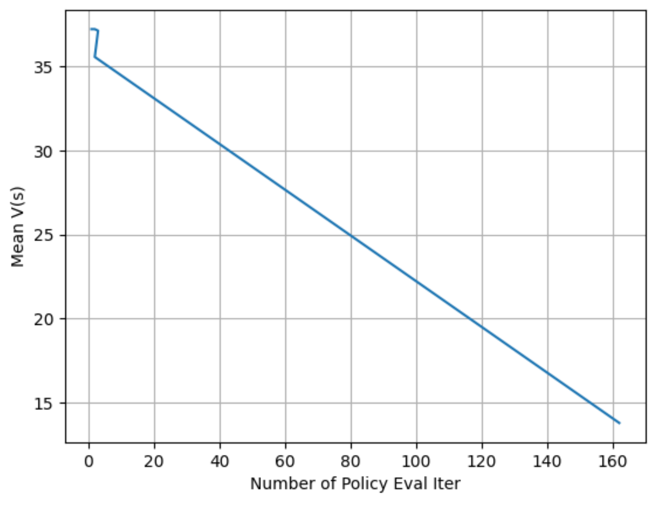


Fig. 1 Learning Rate for Policy Iteration

*Master's Student, Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, 104 S Wright St, Urbana, IL 61801

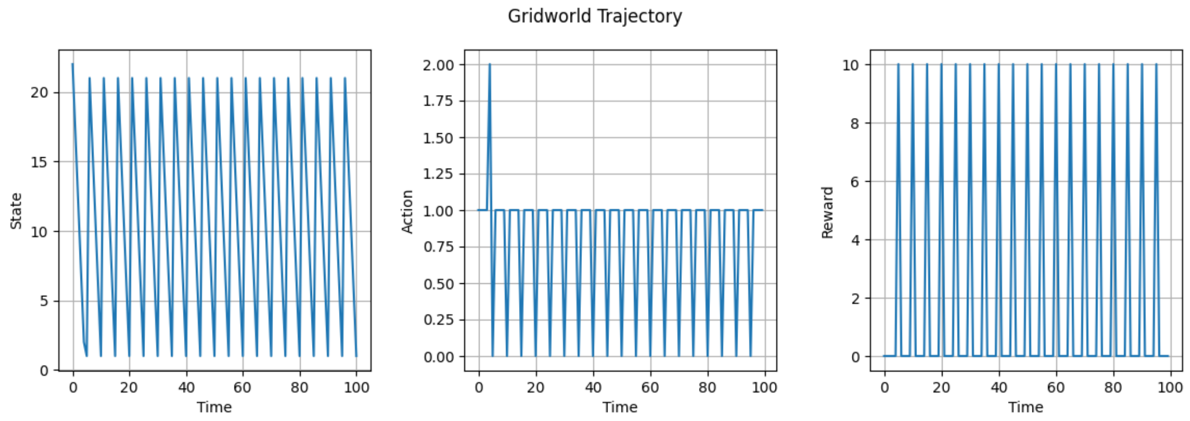
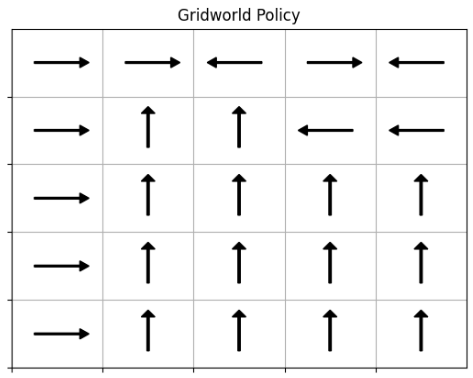
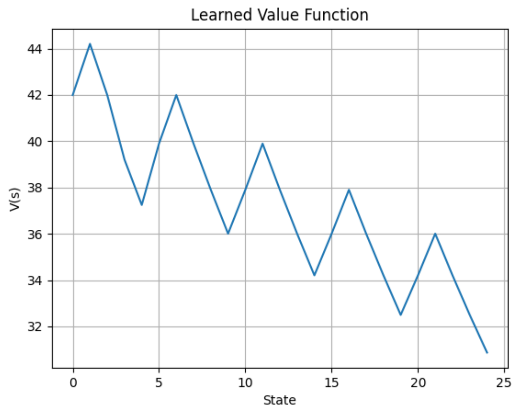


Fig. 2 Example Trajectory for Policy Iteration



(a) Optimal Policy from Policy Improvement



(b) State Value Function from Policy Improvement

Fig. 3

III. Value Iteration (VI)

A. Implementation

My implementation of value iteration followed Sutton and Barto's algorithm. In general, it required fewer episodes that Q-Learning and SARSA to learn the value function. This is likely because it only needs to loop over action space instead of action space and state space like the other model-free algorithms.

B. VI Gridworld Results

The learning rate plot in Figure 4 for value iteration in the gridworld shows the value function improves with the number of value iteration loops. This is expected because the policy shoul improve, resulting in higher value functions. Again, the optimal policy teleports from state 0 to state 21 as many times as possible per episode. The optimal policies and learned value functions from PI and VI agree, which means that both of the model based methods obtained the same optimal policy and the same optimal state value function.

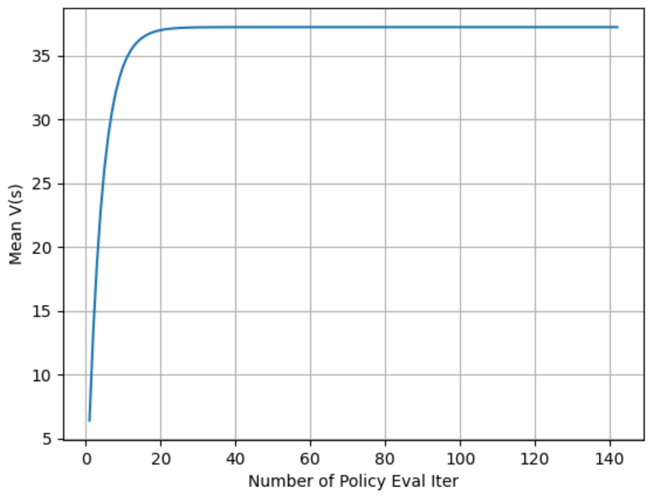


Fig. 4 Learning Rate for Value Iteration

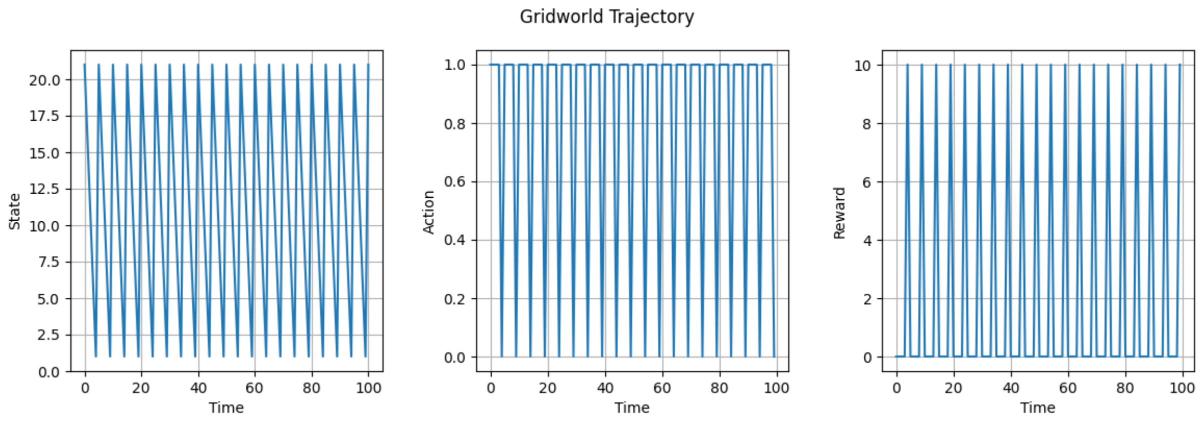
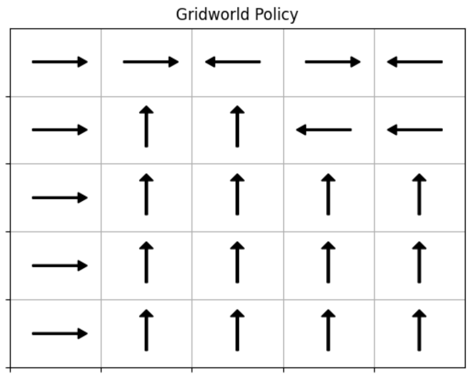
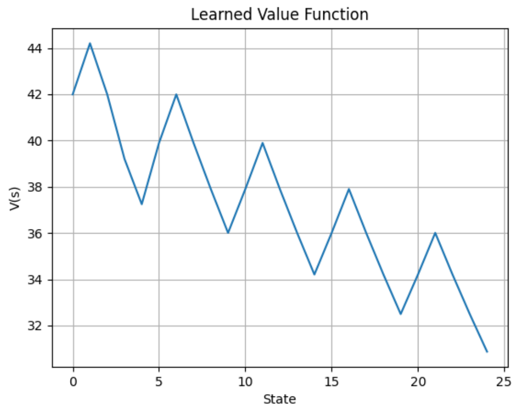


Fig. 5 Example Trajectory for Value Iteration



(a) Optimal Policy from Value Improvement



(b) State Value Function from Value Improvement

Fig. 6

IV. TD(0)

My TD(0) implementation followed the algorithm in Sutton and Barto.

V. SARSA

A. Implementation

My implementation of SARSA followed Stutton and Barto choosing actions from an ϵ-greedy policy. The function outputs an action value function and the optimal policy. The optimal policy is just chosen as the $\texttt{argmax}_a$ of $Q(s, a)$.

B. SARSA Gridworld Results

Table 1 shows the parameters and iterations used for SARSA training in the gridworld environment. The learning rate plot (Figure 7) for SARSA shows that after approximately 1700 episodes the policy is optimal as the maximum return does not increase after that point. The return for each episode has high variance since actions are sampled from an ϵ-greedy policy, which introduces exploration but may result in sub-optimal actions.

While the state and reward trajectories in Figure 8 match the behavior we saw with the PI, VI, and Q-learning policies, the actions and the optimal policy are not the same. However, the value function learned by TD(0) from the action value function generated by SARSA is consistent with the results found from PI, VI, and Q-learning (see Figure 9b). Thus, we can say that this another optimal policy for this environment. This makes sense because there are many possible sequences that can take the agent to the teleporting square from the initial state, and once the agent is in the teleportation loop, they have achieved the maximum rewards for a given scenario. As an example, we can see that it takes 7 actions to move from the bottom right corner (state 24) to the teleporting state one to the right of the top left corner (state 1) following the VI optimal policy (6a), but it also takes 7 actions to move from state 24 to state 1 following the SARSA optimal policy (9a). Since SARSA is learning from episodes where the agent is operating based on the learned policy, the scenario may not visit every value of the state/action space as may times as the model-based methods. Having an ϵ-greedy policy to promote exploration helps with this.

From Figure 10, we see that high step sizes ($\alpha \to 1$) result in a higher percetnage of negative return scenarios, while $0.505 \leq \alpha \leq 0.752$ result in higher overall return and fewer episodes with negative returns. We also see that smaller epsilon ($\epsilon = 0.01$) results in the highest overall reward and few episodes with negative return. This means that having an ϵ-greedy policy with a low probability of choosing an action that does not maximize $Q(s, a)$ leads to the best results.

Table 1 Parameters for training with SARSA in gridworld environment

ϵ	α	Num. Episodes SARSA	Num. Episodes TD(0)
0.2	0.5	10,000	1,000

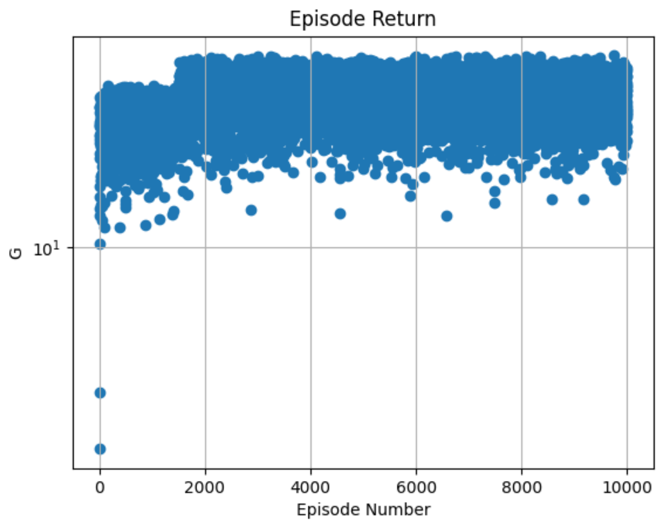


Fig. 7 Learning Rate for SARSA

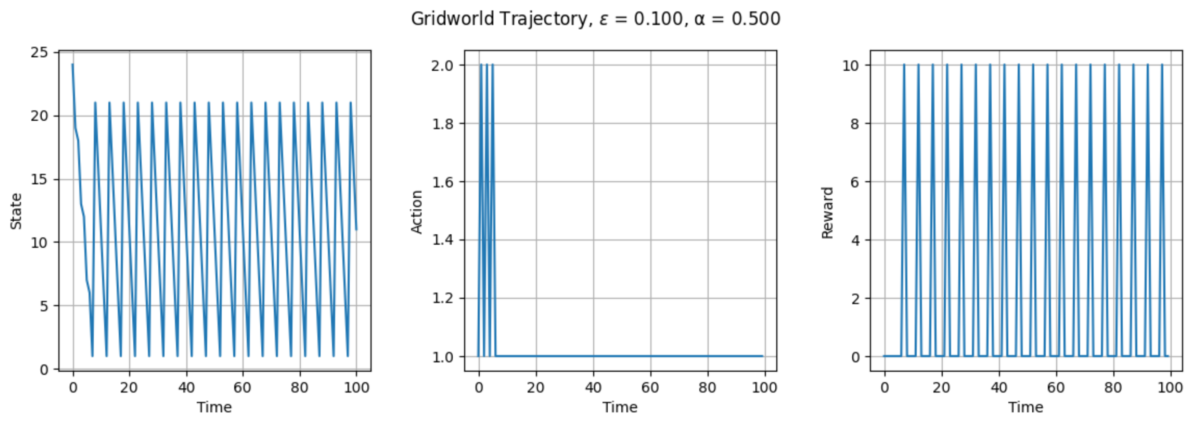
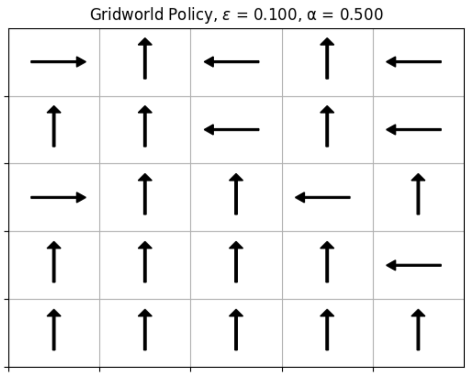
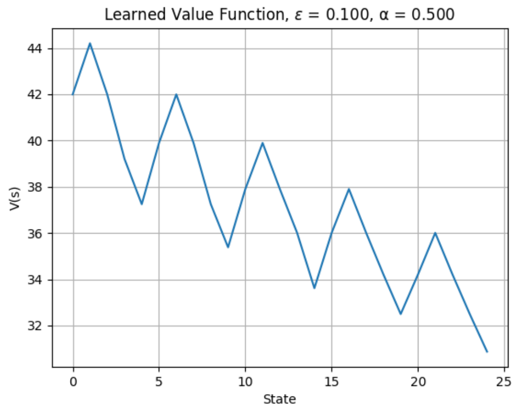


Fig. 8 Example Trajectory for SARSA

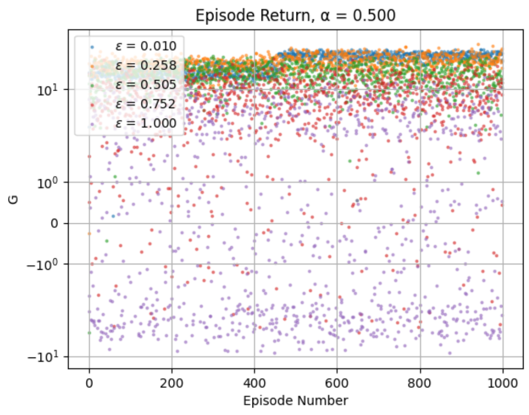


(a) Optimal Policy from SARSA

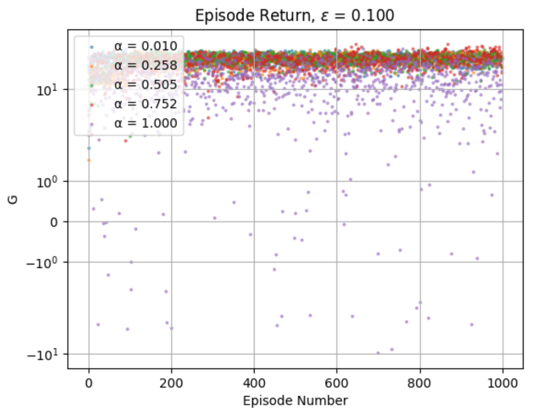


(b) State Value Function from SARSA

Fig. 9



(a) Return varying ϵ for SARSA



(b) Return varying α for SARSA

Fig. 10

C. SARSA Pendulum Results

The parameters in Table 2 shows the parameters used to produce the following results. The trajectory in Figure 12 shows that the pendulum is balanced in the middle of the trajectory for around 10 seconds. It was hard for me to find the correct hyperparameters to use to make SARSA converge to as good of a policy as Q-learning for this scenario. I changed the hyperparameters based on what performed the best in the parameter sweep studies (Figure 14) to $\epsilon = 0.258$ and $\alpha = 0.505$, but even with 50,000 episodes, the pendulum still swung back in forth instead of balancing. I also tried decreasing and increasing the state and action space sizes (by changing the number of θ, $\dot{\theta}$, and τ points, but I did not see any difference in the results (other than a shorter episode run time for smaller state and action spaces). Even when sweeping through the hyperparameter space by trying multiple combinations of $\epsilon \in [0.01, 1]$ and $\alpha \in [0.1, 0.9]$ with 10,000 episodes, the resulting policies were not able to achieve more time balancing than shown in the example trajectory in Figure 12.

The learned state value function from the SARSA policy in Figure 13b achievs a maximum of 8, while the state value function for Q-learning for the pendulum achievs a maximum of 20 (see Figure 21b). With the large state and action space of this environment, on-policy learning via SARSA does not perform as well as off-policy learning with Q-learning. Using an off-policy method allows for more exploration of other policies besides the greedy policy, which helped convergence without as much sensitivity to hyperparameters.

Table 2 Parameters for training with SARSA in pendulum environment

ϵ	α	Num. Episodes SARSA	Num. Episodes TD(0)	Num. θ	Num. $\dot{\theta}$
0.01	0.5	10,000	1,000	15	21

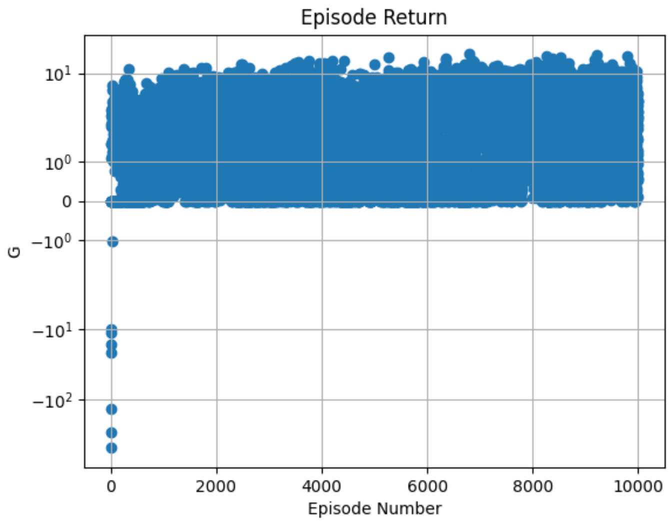


Fig. 11 Learning Rate for SARSA

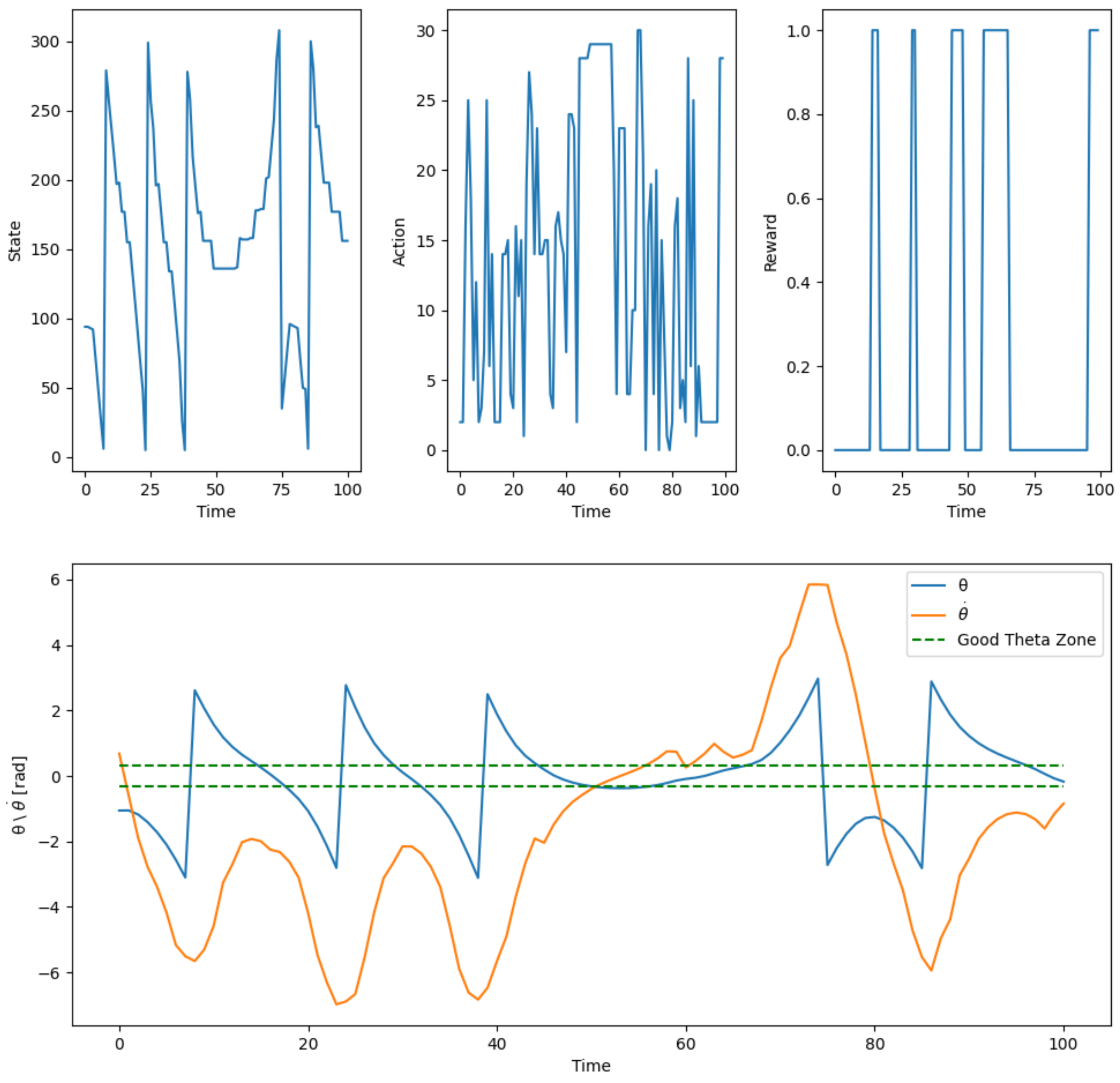
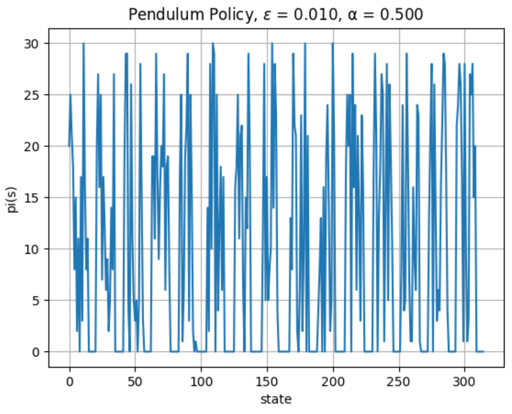
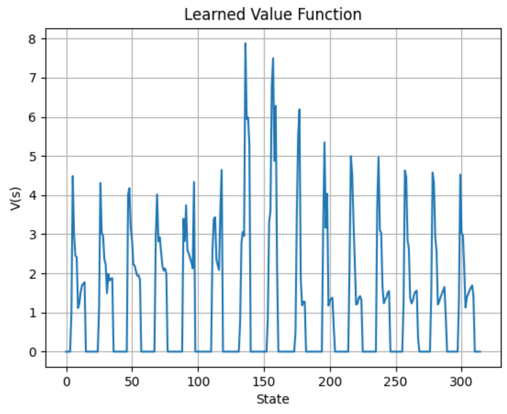


Fig. 12 Example Trajectory for SARSA

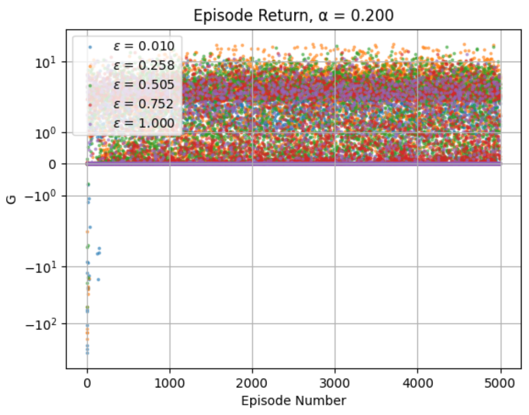


(a) Optimal Policy from SARSA

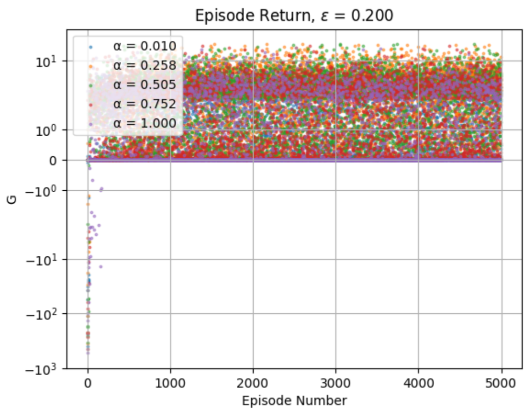


(b) State Value Function from SARSA

Fig. 13



(a) Return varying ϵ for SARSA



(b) Return varying α for SARSA

Fig. 14

VI. Q-Learning

A. Implementation

Q-learning was implemented following Sutton and Barto using an ϵ-greedy policy.

B. Q-Learning Gridworld Results

The following results were obtained using the parameters in Table 3. The learning rate plot shows that the algorithm reaches the maximum episode return after approximately 1000 episodes (Figure 15). The trajectory teleports from state 1 to state 21, and the state value function agrees with the results from other algorithms. In addition, the optimal policy from Q-learning matches the optimal policy from the model-based algorithms. In addition, we see similar trends when varying ϵ and α in Figure 18 and we did for SARSA in Figure 10, namely that $\epsilon = 0.01$ and $0.505 \leq \alpha \leq 0.752$ result in the best returns.

Table 3 Parameters for training with Q-learning in gridworld environment

ϵ	α	Num. Episodes Q-Learning	Num. Episodes TD(0)
0.2	0.5	10,000	1000

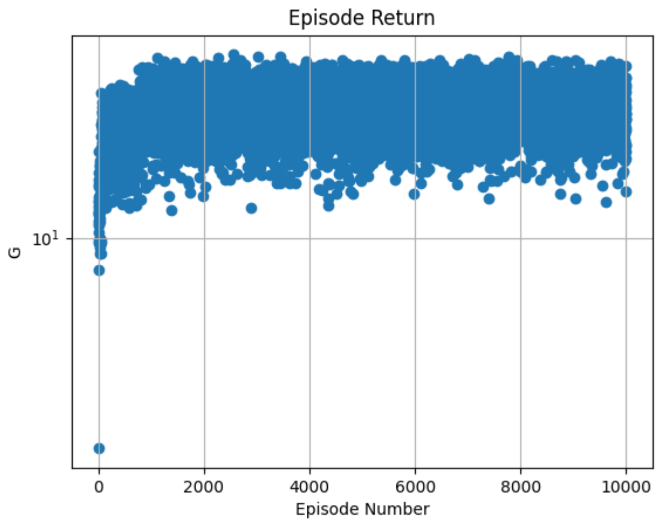


Fig. 15 Learning Rate for Q-Learning

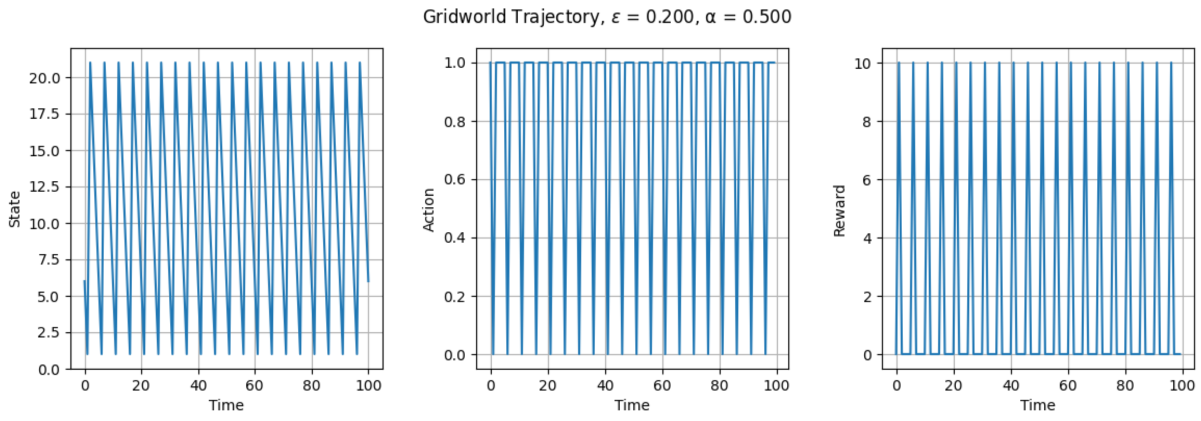
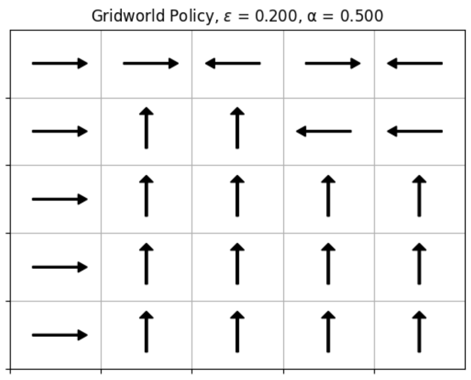
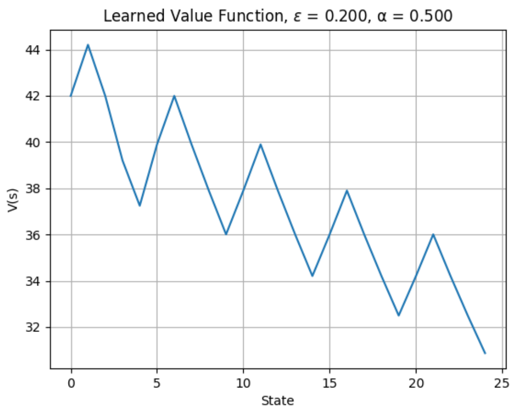


Fig. 16   Example Trajectory for Q-Learning

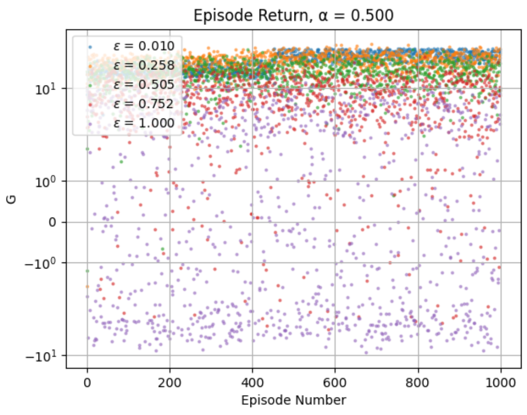


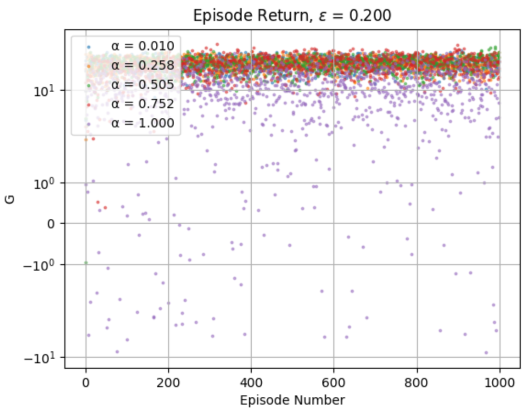


(a) Optimal Policy from Q-Learning

(b) State Value Function from Q-Learning

Fig. 17





(a) Return varying ϵ for Q-Learning

(b) Return varying α for Q-Learning

Fig. 18

C. Q-Learning Pendulum Results

The pendulum results were generated using the parameters in Table 4. The learning rate plot in Figure 19 shows that the highest return was achieved after approximately 10,000 episodes, but that the variance decreases around 40,000 episodes as the results are more tightly spaced. From the trajectory in Figure 20, we see that the optimal policy result in the pendulum being balanced between $\pm 0.1\pi$ after swinging back and forth once, achieving a reward of 1 for the remainder of the trajectory.

Figure 22 shows that $\epsilon = 0.258$ resulted in larger positive returns that other ϵ values. In addition, a large step size ($\alpha = 0.752$) resulted in the highest return over time. The pendulum scenario was more sensitive to hyperparmeters than the gridworld, likely because of the larger state/action space.

Table 4 Parameters for training with Q-learning in pendulum environment

ϵ	α	Num. Episodes Q-Learning	Num. Episodes TD(0)	Num. θ	Num. $\dot{\theta}$
0.01	0.5	50,0000	5,000	20	26

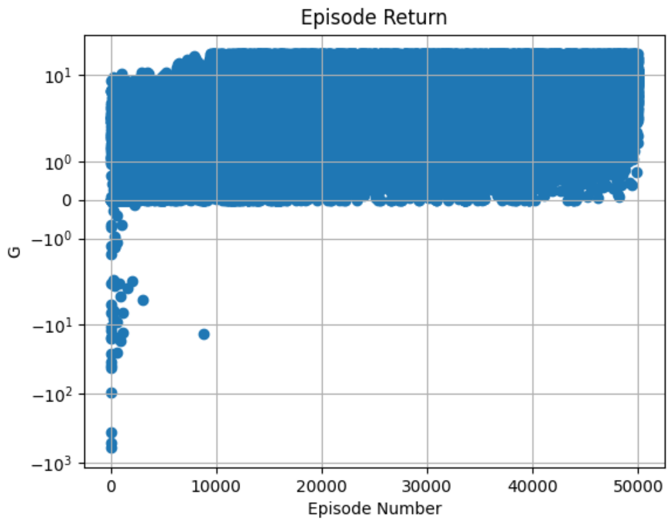


Fig. 19 Learning Rate for Q-Learning

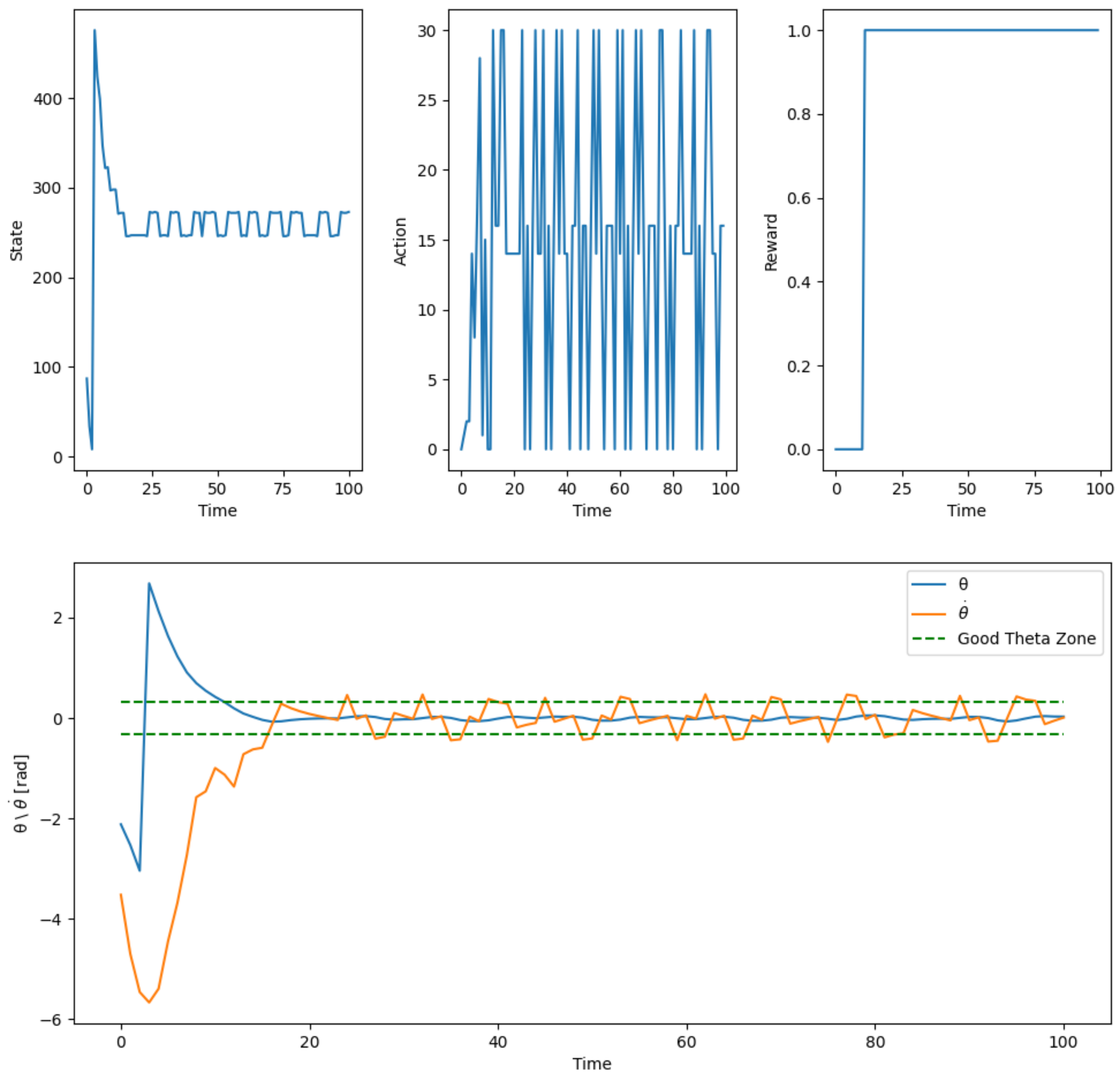
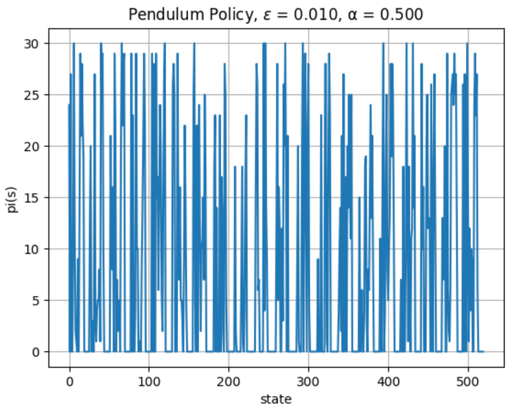
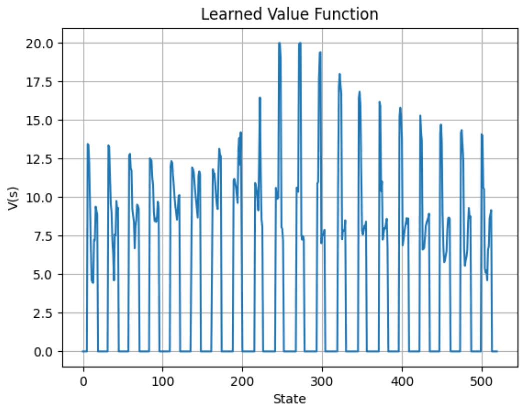


Fig. 20 Example Trajectory for Q-Learning

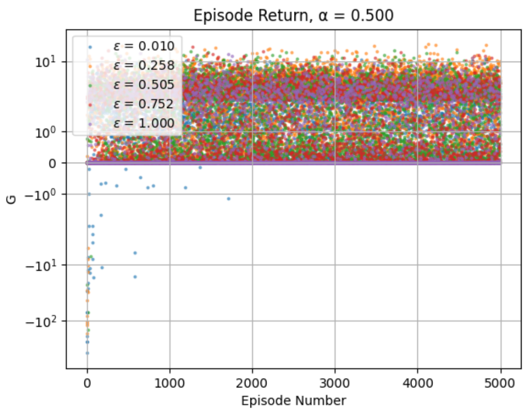


(a) Optimal Policy from Q-Learning

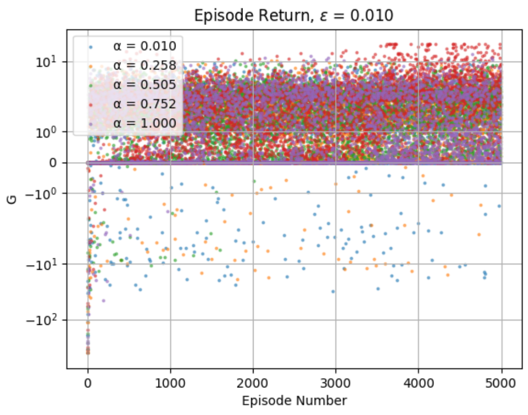


(b) State Value Function from Q-Learning

Fig. 21



(a) Return varying ϵ for Q-Learning



(b) Return varying α for Q-Learning

Fig. 22