Dynamic Programming: Implementing Reinforcement Learning Algorithms

Pallavi Ravada*
University of Illinois Urbana-Champaign, Urbana-Champaign, Illinois, 61820

I. Introduction

This report details the implementation of five reinforcement learning algorithms in a "tabular" setting assuming small finite state and action spaces.

Two of the algorithms are model based:
1) Policy iteration (Chapter 4.3, Sutton and Barto)
2) Value iteration (Chapter 4.4, Sutton and Barto)

Two of the algorithms are model free:
1) SARSA with an epsilon-greedy policy, i.e., on-policy TD(0) to estimate Q (Chapter 6.4, Sutton and Barto)
2) Q-learning with an epsilon-greedy policy, i.e., off-policy TD(0) to estimate Q (Chapter 6.5, Sutton and Barto)

The final algorithm is also model free and computes the value function associated with a given policy:
1) TD(0) (Chapter 6.1, Sutton and Barto)

Brief algorithm descriptions, implementation notes, and results will be given in the following sections. For detailed algorithm pseudo-code please refer to Sutton and Barto.

II. Algorithm Description and Implementation

A. Environment

The algorithms are tested in the following provided environments:
1) A simple grid-world, for which an explicit model is available (Example 3.5, Chapter 3.5, Sutton and Barto).
2) A simple pendulum with discretized state and action spaces, for which an explicit model is not available ("`http://underactuated.mit.edu/pend.html`")

For both environments the reinforcement learning problem is expressed as a Markov Decision Process with an infinite time horizon and a discount factor of $\gamma = 0.95$.

B. Policy Iteration

Policy iteration is the process of repeatedly evaluating and improving a policy where each successive policy is guaranteed to be a strict improvement over the previous policy unless the policy is optimal. As mentioned before this process is composed of an evaluation step and an improvement step. The evaluation step itself is an iterative computation that utilizes the value function of the previous policy. In policy iteration the model must provide the state transition probabilities and the expected reward from any state-action pair, hence the algorithm being model based.

To initialize this algorithm an arbitrary initial value is used for the state value function V. In this code implementation the value function was an array of size [states x actions] with value 0.25 (from 1/number of actions). Additionally a helper function was utilized to "look one step ahead" to easily determine the expected value of the next possible actions from the current state. From this the best next action could be determined. Once the policy is stable the value function and policy were returned. Additionally the mean value function for every state was computed for plotting purposes.

*Graduate Student, Department of Aerospace Engineering.

C. Value Iteration

Value iteration is the other model-based method implemented and is a special case of when policy evaluation is stopped after just one back up of each state. Value iteration functions as completing one sweep of policy evaluation, then one sweep of policy improvement in each of its iterations. After the optimal value function is determined, then the deterministic policy is outputted. Again since value iteration is model based it uses the model's distributions of the next state and reward in order to calculate optimal actions to take.

This algorithm is initialized with a state value function array of all zeros of size [states]. To determine the next best action from the current state the helper function "look one step ahead" is again utilized. Once the value function is determined to within a certain tolerance, the deterministic policy is outputted by determining the best action from the value function.

D. SARSA with ϵ-Greedy Policy

SARSA is the first of the model free algorithms implemented. As a model free method this algorithm purely samples from experience, not the model's distribution of transition probabilities. In SARSA the goal is to learn the action value function Q, as opposed to the state value function determined previously. Transitions from one state-action pair to another are considered, denoted by $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$, giving rise to the algorithm's name.

In this implementation SARSA is written using an ϵ-greedy policy. The hyper-parameter $\epsilon \in [0, 1]$ denotes the balance between exploration and exploitation. The ϵ-greedy algorithm either selects a random action (exploration) or the best action (exploitation) based on the given ϵ value.

This algorithm also has another hyper-parameter $\alpha \in [0, 1]$, which dictates the learning step size. This means this parameter dictates to what extent the new information overrides the old information. A value of 0 means all new information is disregarded, and a value of 1 means the agent only would consider new information. The hyper-parameter γ is as given in the introduction.

To initialize the SARSA algorithm Q is initialized arbitrarily as an array of zeros of size [states x actions]. Then for each episode, using the ϵ-greedy policy an action is chosen, a step is taken, and the Q value is computed. This is done for all of the episodes given as the input parameter.

E. Q-LEARNING with ϵ-Greedy Policy

Q-learning is an off policy control algorithm, as opposed to SARSA which as on policy. This means Q-learning is able to estimate the optimal action-value function independent of policy. The policy only effects the state-action pairs that are visited and updated. Q-learning has the same hyper-parameters as SARSA, ϵ and α that require tuning. γ is as given in the introduction.

The implementation is very similar to SARSA, the difference being Q-learning chooses a greedy action as opposed to the action dictated by the policy. The greedy action gives the maximum Q-value for the state. For each episode, stepping through the environment the Q value is computed and returned at the algorithm termination once all of the episodes have been completed.

F. TD(0)

The TD(0) algorithm is used to learn the value function associated with the optimal policy produced by SARSA and Q-learning. This is the simplest form of the TD learning algorithm. This algorithm uses experience to solve the prediction problem, updating their estimate of V to the optimal V. TD(0) is also model free as the agent does not know the state MDP transitions, and the agent learns fron sampled experience.

This algorithm is initialized with V being an array of zeros with size [states]. Then for every episode, the best action is determined according to the given policy, a step is take, and the value function is updated.

III. Results

The results of the algorithms for the gridworld and pendulum environments are given below.

A. Gridworld

1. Policy Iteration

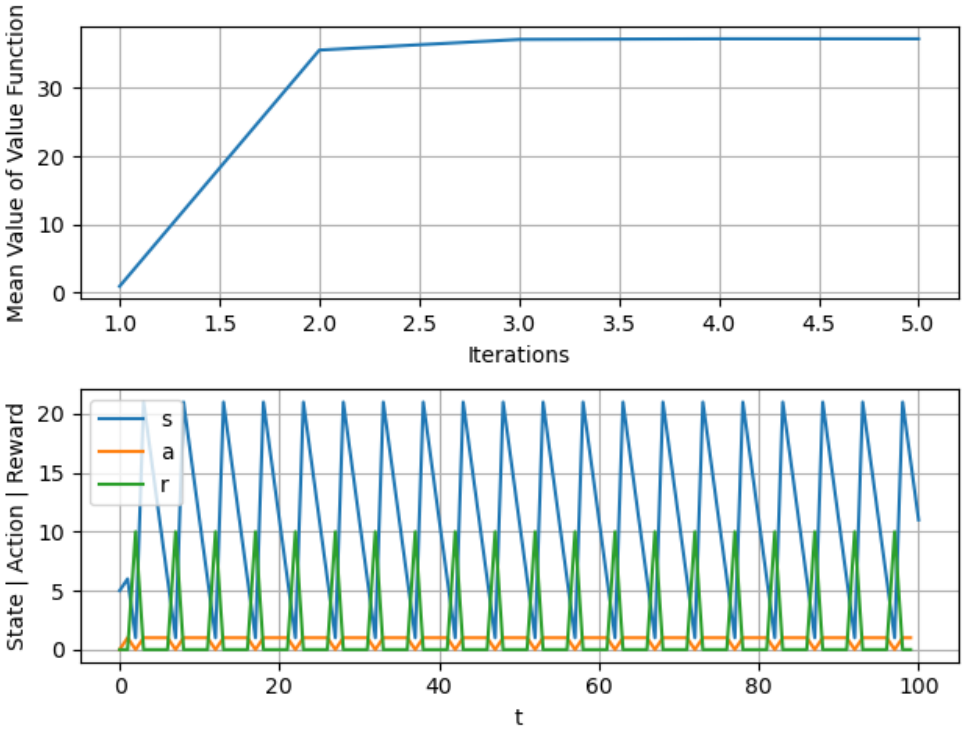

Fig. 1 PI: Learning Curve & Example Trajectory of Trained Agent

Figure 1 (top) shows the learning curve for the agent for policy iteration. As expected policy iteration converges in very few iterations and converges fairly quickly. For policy iteration, "the number of value iterations" is the number of iterations spent on policy evaluation. Figure 1 (bottom) shows the trajectory of the trained agent, the state, action, and reward over time. As expected the trajectory is fairly smooth and periodic as expected when having access to the model transition probabilities.

Figure 2 (left) shows the optimal policy for the gridworld model and figure 2 (right) shows the optimal state value function. Looking visually at the plots it can be seen that the policy directly corresponds to movements that result in optimal state value function transitions.

2. Value Iteration

Figure 3 (top) shows the learning curve for the agent for policy iteration. Value iteration converges in more iterations than policy iteration. For value iteration, "the number of value iterations" is synonymous with the number of iterations of the algorithm. Figure 3 (bottom) shows the trajectory of the trained agent, which again is fairly smooth and periodic as expected when having access to the model transition probabilities.

Figure 4 (left) shows the optimal policy for the gridworld model and figure 4 (right) shows the optimal state value function. These plots match those from the policy iteration, showing that both methods did indeed converge to the optimal policy.

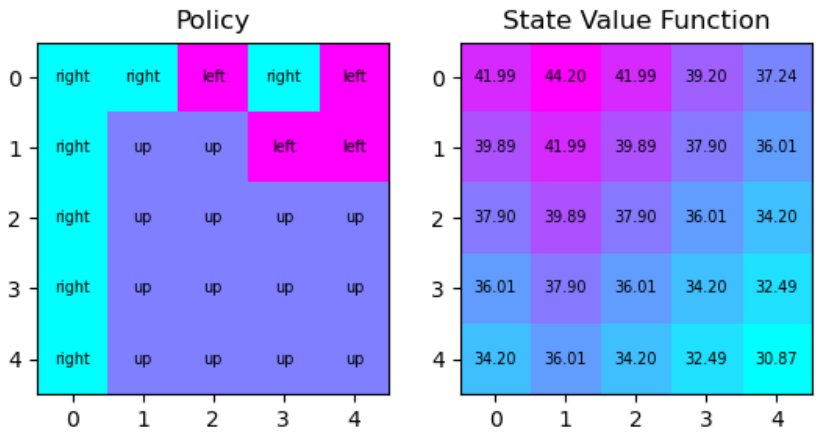

Fig. 2 PI: Policy and State-Value Function Corresponding to Agent

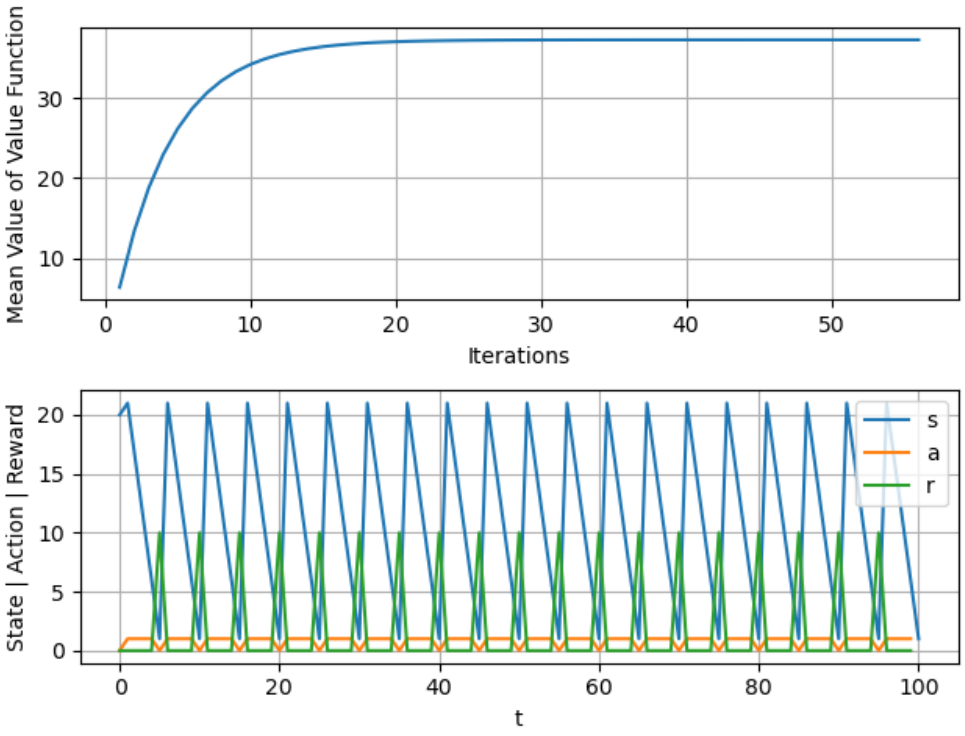

Fig. 3 VI: Learning Curve & Example Trajectory of Trained Agent

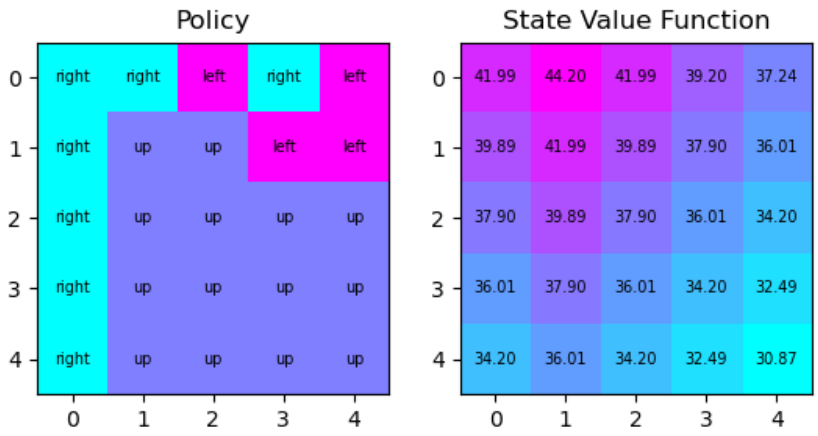

Fig. 4 VI: Policy and State-Value Function Corresponding to Agent

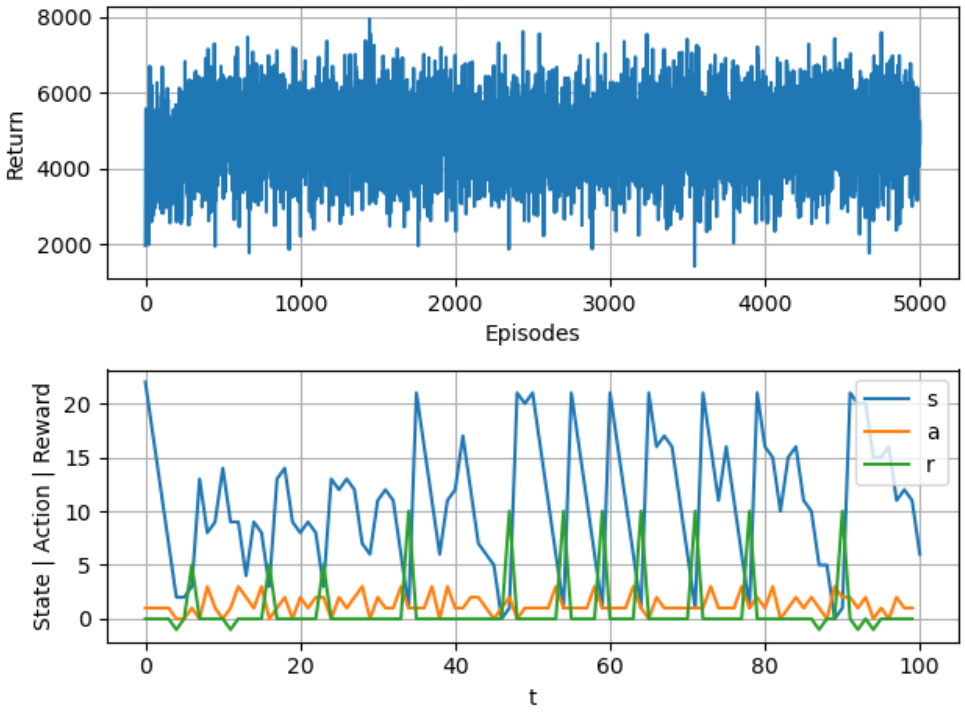

Fig. 5 SARSA: Learning Curve & Example Trajectory of Trained Agent

3. SARSA with ϵ-Greedy Policy

Figure 5 (top) shows the learning curve for the agent for SARSA. This shows the return for each episode as the agent learns how to navigate the gridworld. As the episodes progress the average return slightly increases then remains relatively constant for the remainder of the episodes. Figure 5 (bottom) shows the trajectory of the trained agent, which again is no longer fairly smooth. The periodicity remains as expected. The jagged curves are likely due to no longer having access to the model transition probabilities.

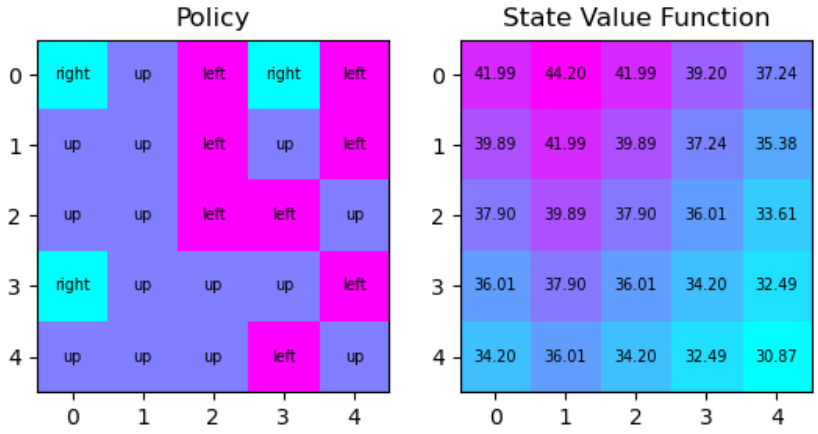

Fig. 6 SARSA: Policy and State-Value Function Corresponding to Agent

Figure 6 (left) shows the optimal policy for the gridworld model and figure 6 (right) shows the optimal state value function. These plots do not exactly match the ones from the policy and value iteration. Some of the state-values are slightly different yet the relationship between the state transitions remains the same. The difference in the policy can be attributed SARSA learning a near optimal policy through exploring.

These results were for $\epsilon = 0.5$ and $\alpha = 0.4$. For this situation the balance between exploration and exploitation is evenly split, and the learning rate was chosen to favor old information slightly more than new information.

Figure 7 (top) shows the return for various values of ϵ for an α value of 0.4. As can be see, as the ϵ value increases from 0.25 the return decreases until $\epsilon = 1$ yields almost zero return for a purely exploitation scheme with no exploration. Figure 7 (bottom) shows the return for various values of α for an ϵ value of 0.5. As the α value is increased starting from 0.25 the return decreases as the learning step size grows larger. When $\alpha = 0$ the return is negative.

4. Q-LEARNING with ϵ-Greedy Policy

Figure 8 (top) shows the learning curve for the agent for SARSA. Again, as the episodes progress the average return slightly increases then remains relatively constant for the remainder of the episodes. Figure 8 (bottom) shows the trajectory of the trained agent, which again is no longer fairly smooth. The periodicity remains as expected. The jagged curves are likely due to no longer having access to the model transition probabilities.

Figure 9 (left) shows the optimal policy for the gridworld model and figure 9 (right) shows the optimal state value function. These plots exactly match the ones from the policy and value iteration. This converges to the same solution because Q-learning is directly learning the optimal policy. These results were for $\epsilon = 0.5$ and $\alpha = 0.4$ again for similar reasoning as before.

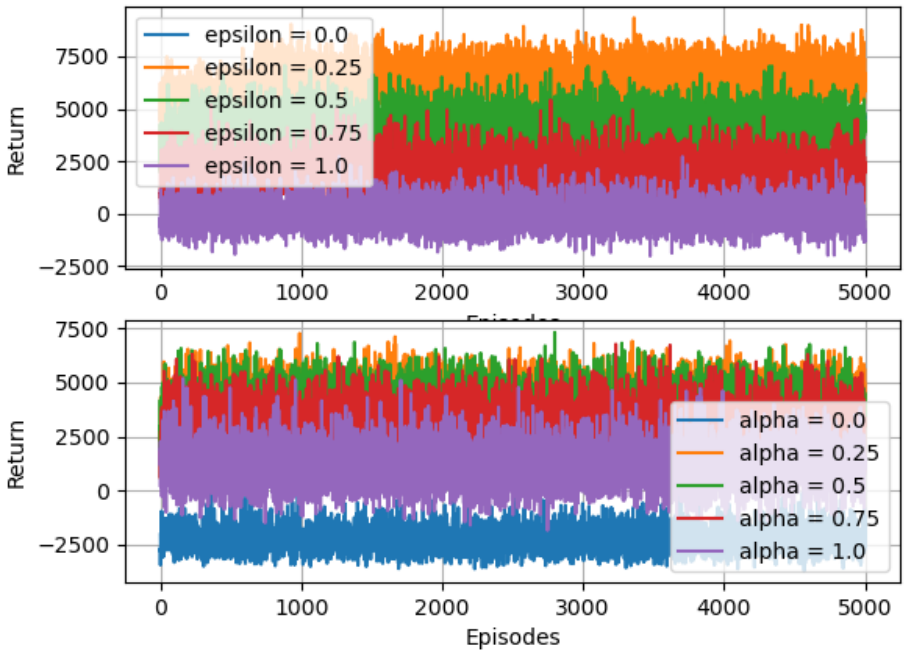

Fig. 7 SARSA: Learning Curves for Several Different Values of ϵ and α

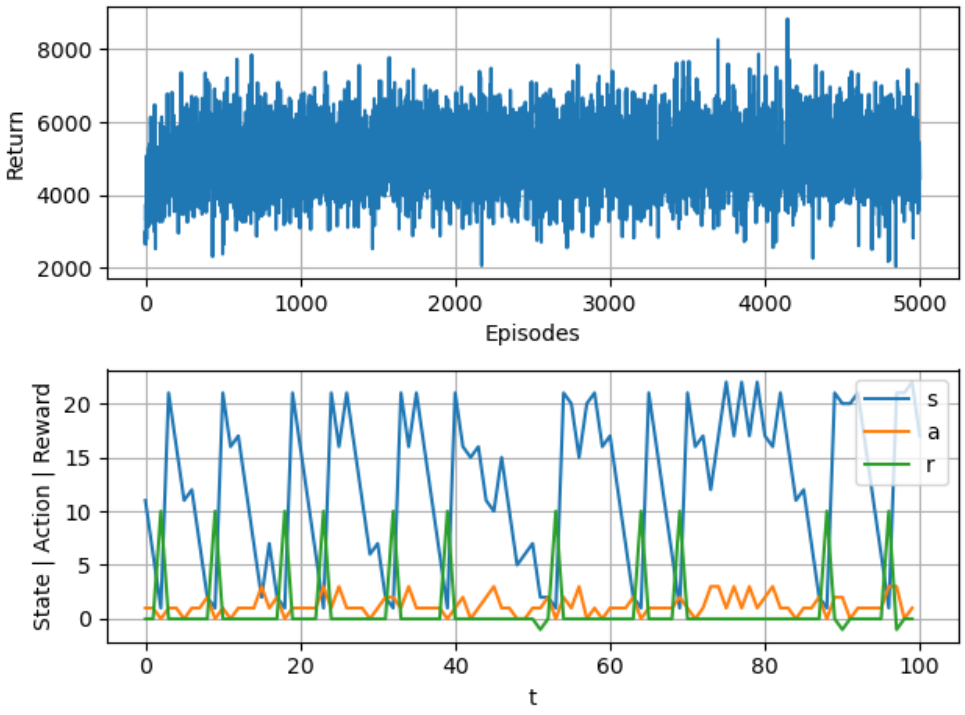

Fig. 8 Q-learning: Learning Curve & Example Trajectory of Trained Agent

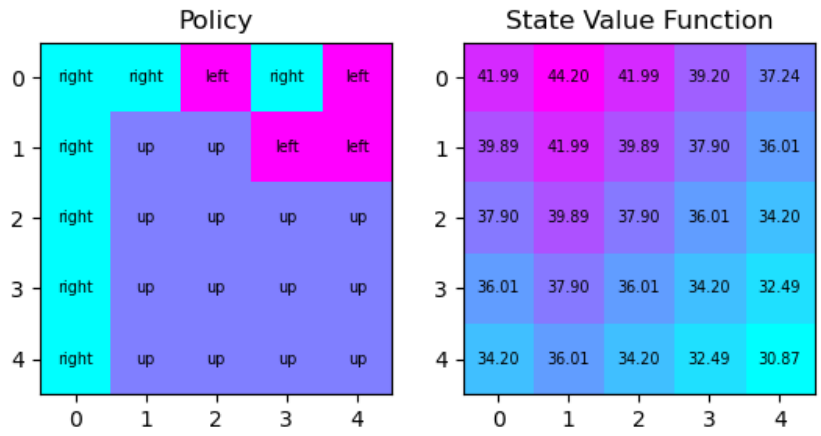

Fig. 9 Q-learning: Policy and State-Value Function Corresponding to Agent

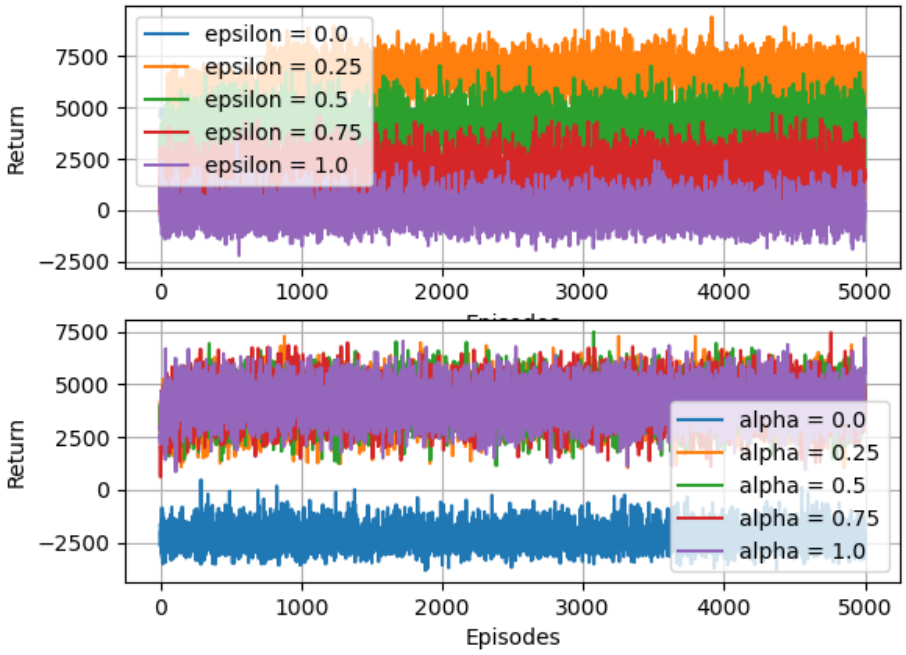

Fig. 10 Q-learning: Learning Curves for Several Different Values of ϵ and α

Figure 10 (top) shows the return for various values of ϵ for an α value of 0.4. As can be see, as the ϵ value increases from 0.25 the return decreases until $\epsilon = 1$ yields almost zero return for a purely exploitation scheme with no exploration. Figure 10 (bottom) shows the return for various values of α for an ϵ value of 0.5. As the α value is increased starting from 0.25 the return remains about the same as the learning step size grows larger. When $\alpha = 0$ the return is negative.

B. Pendulum

1. SARSA with ϵ-Greedy Policy

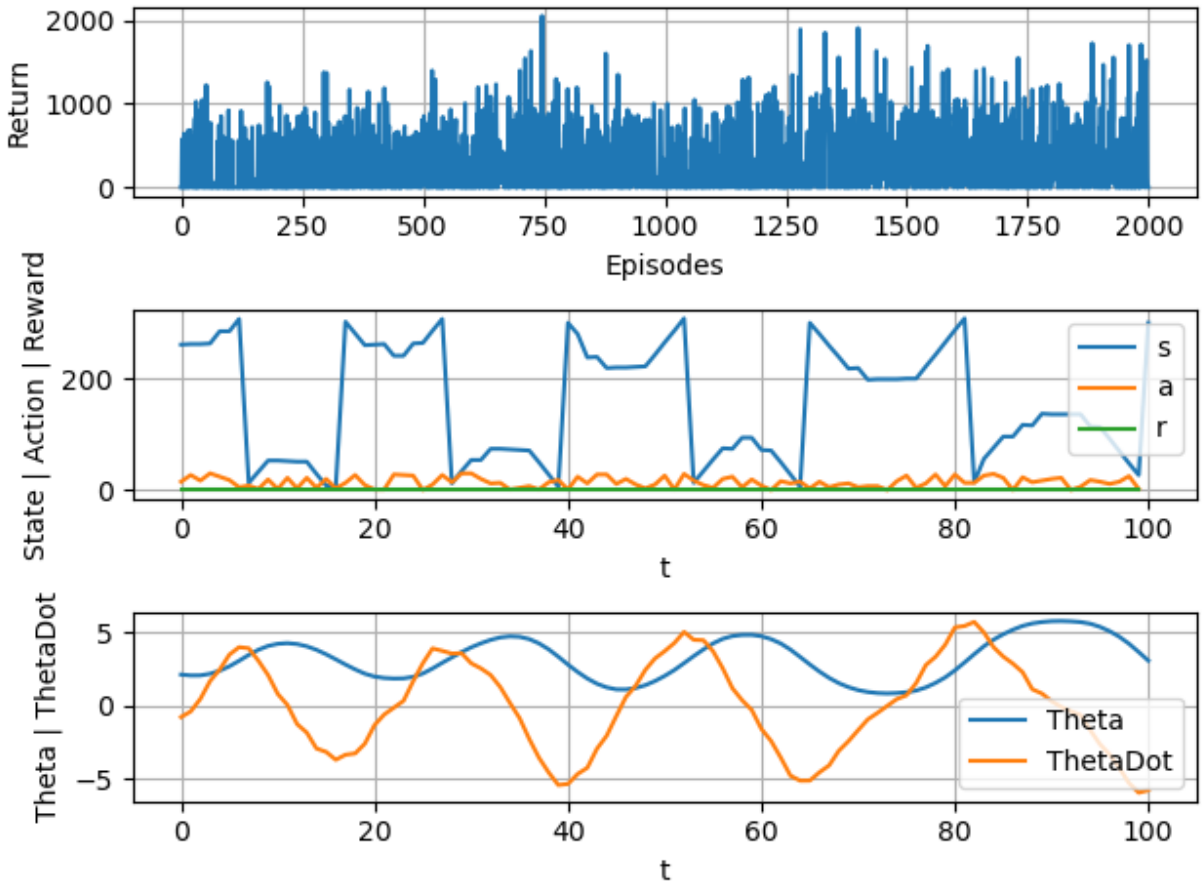

Fig. 11 SARSA: Learning Curve & Example Trajectory of Trained Agent

Figure 11 (top) shows the learning curve for the agent for SARSA. As the episodes progress the return slightly increases then remains relatively constant for the remainder of the episodes. Figure 11 (bottom) shows the trajectory of the trained agent, which shows a periodic trajectory. I believe the code has not converged to the optimal value function as the pendulum oscillates quickly between the states as the maximum theta value is below the threshold of 15°. More episodes or more experimentation with the hyper-parameters is likely required to determine the proper global optimal policy.

These results were accomplished with $\epsilon = 0.75$ and $\alpha = 0.2$. These values were chosen so that a smaller learning step was utilized along with more exploitation.

Figure 12 (left) shows the optimal policy for the pendulum model and figure 12 (right) shows the optimal state value function. Again the oscillatory behavior indicative of a pendulum can be seen in the color gradient of value function and policy.

Figure 13 (top) shows the return for various values of ϵ for an α value of 0.2. Although it is difficult to see, as the ϵ value increases from 0.25 the return decreases slightly to $\epsilon = 1$ for a purely exploitation scheme with no exploration.

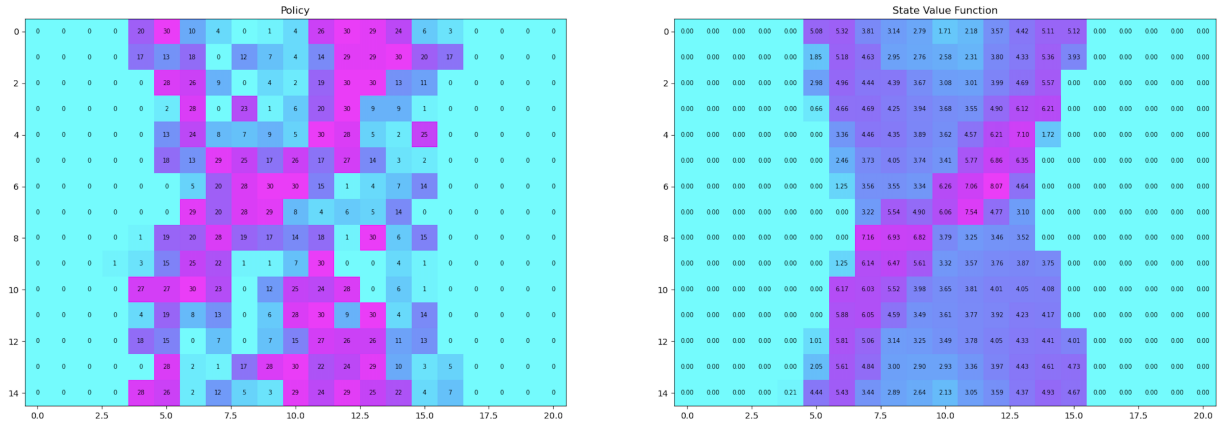

Fig. 12 SARSA: Policy and State-Value Function Corresponding to Agent

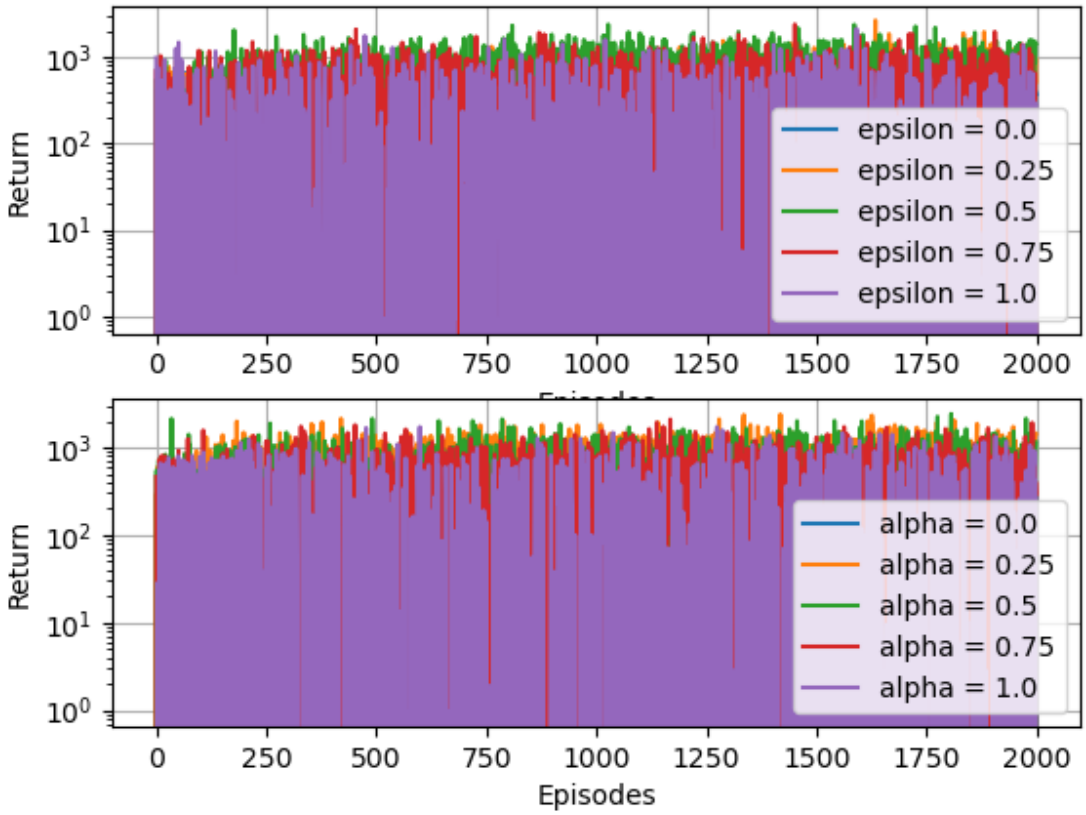

Fig. 13 SARSA: Learning Curves for Several Different Values of ϵ and α

This is a similar trend from the gridworld. Figure 13 (bottom) shows the return for various values of α for an ϵ value of 0.5. As the α value is increased starting from 0.25 the return remains about the same as the learning step size grows larger.

2. Q-LEARNING with ϵ-Greedy Policy

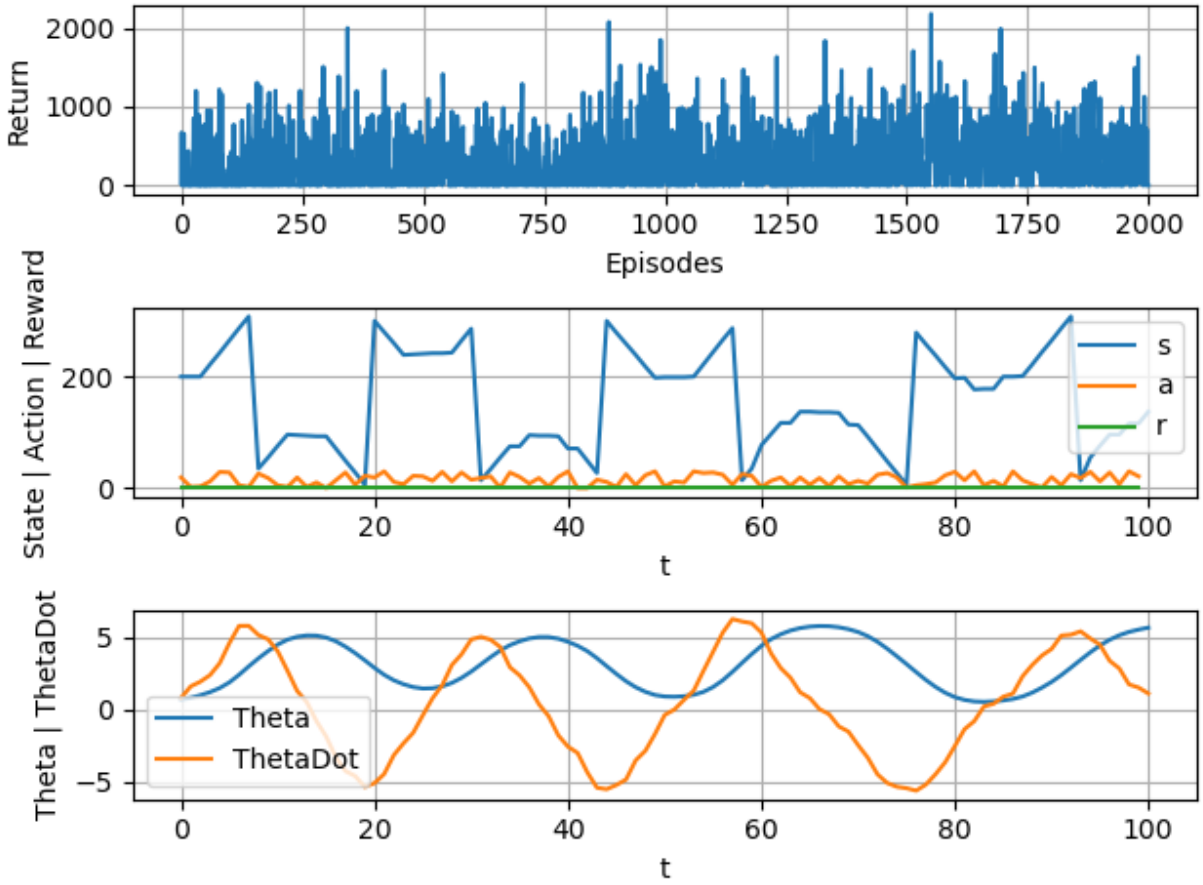

Fig. 14 Q-learning: Learning Curve & Example Trajectory of Trained Agent

Figure 14 (top) shows the learning curve for the agent for Q-learning. As the episodes progress the return slightly increases then remains relatively constant for the remainder of the episodes, similarly for SARSA. Figure 14 (bottom) shows the trajectory of the trained agent, which shows a periodic trajectory. Again, I believe the code has not converged to the optimal value function as the pendulum continues to oscillates quickly between states well below the maximum theta threshold of 15°. More episodes or more experimentation with the hyper-parameters is likely required to determine the proper global optimal policy.

These results were accomplished with $\epsilon = 0.75$ and $\alpha = 0.2$ for the same reasoning as above.

Figure 15 (left) shows the optimal policy for the pendulum model and figure 15 (right) shows the optimal state value function. Again, the oscillatory behavior indicative of a pendulum can be seen in the color gradient of value function and policy. These figures are similar to those determined from SARSA due to the inherent differences in the algorithms as discussed before.

Figure 16 (top) shows the return for various values of ϵ for an α value of 0.2. Although it is difficult to see, as the ϵ value increases from 0.25 the return decreases slightly to $\epsilon = 1$ for a purely exploitation scheme with no exploration. This is a similar to what is seen for SARSA. Figure 16 (bottom) shows the return for various values of α for an ϵ value of 0.5. As the α value is increased starting from 0.25 the return remains about the same as the learning step size grows larger. Again this is similar to what was seen in SARSA.

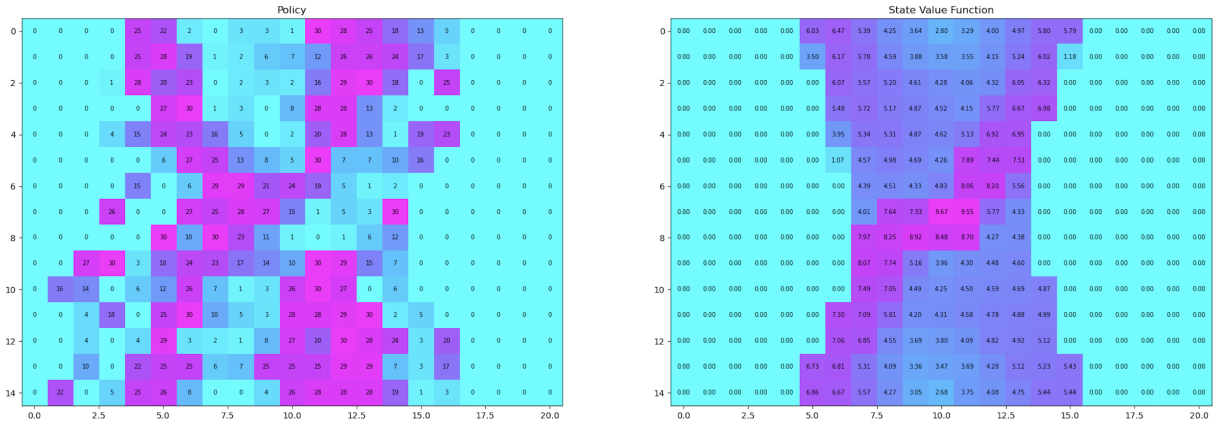

Fig. 15 Q-learning: Policy and State-Value Function Corresponding to Agent

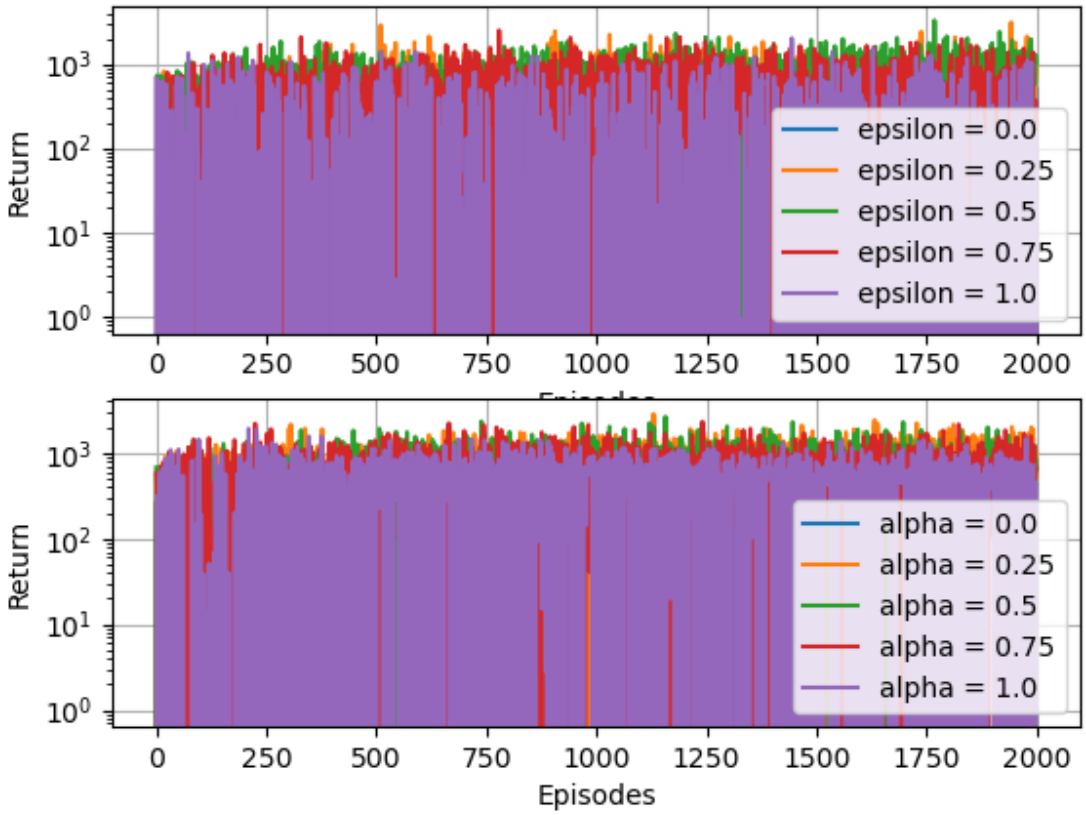

Fig. 16 Q-learning: Learning Curves for Several Different Values of ϵ and α