Reinforcement Learning: Assignment 3

Sameer Ahamed (202381922)

August 9, 2024

1 Problem 1

1.1 Problem Summary

The task involves solving a grid world problem using two different reinforcement learning algorithms: SARSA and Q-Learning. The environment is a 5x5 grid with a start state, terminal states, and red states that impose a penalty. The goal is to learn an optimal policy to navigate from the start state to one of the terminal states while minimizing penalties.

1.2 Gridworld Layout

- **Start State:** (4, 0)
- **Terminal States:** (0, 0), (0, 4)
- **Red States:** (2, 0), (2, 1), (2, 3), (2, 4)

The grid is a 5x5 matrix where:

- Blue square: Start state
- Black squares: Terminal states
- Red squares: States that impose a penalty and reset the agent to the start state

1.3 Objectives

- Implement SARSA and Q-Learning algorithms.
- Train both algorithms to learn the optimal policy.
- Compare the policies learned by SARSA and Q-Learning.
- Analyze the sum of rewards over episodes for both algorithms.

1.4 Solving using SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy algorithm that updates the Q-value based on the action actually taken by the agent.

1.4.1 Output Analysis

Policy:

```
T ← → → T
↑ ← ↑ ↑ ↑
R R ↑ R R
→ → ↑ ← ↓
S → ↑ ↑ ←
```

Explanation: The SARSA-derived policy shows that the agent tends to favor certain actions over others to avoid penalties and reach terminal states. The policy includes a mix of directional actions with some variations indicating exploration.

1.5 Solving with Q-Learning

Q-Learning is an off-policy algorithm that updates the Q-value based on the maximum reward action possible from the next state, regardless of the agent's current policy.

1.5.1 Output Analysis

Policy:

```
T ← → → T
↑ ↑ ↑ ↑ ↑
R R ↑ R R
→ → ↑ ← ←
S ↑ ↑ ← ←
```

Explanation: The Q-Learning-derived policy shows a more consistent pattern of actions. The agent tends to choose actions that lead to terminal states directly, showing a more deterministic and optimized approach.

1.6 Similarities and Differences between SARSA and Q-Learning Output

- **Similarities:**
  - Both policies aim to avoid red states and reach terminal states.
  - Both algorithms use an ϵ-greedy action selection strategy for exploration.
  - Both policies aim to avoid the red states as much as possible due to the high penalty (-20).
  - Both policies have learned to navigate towards terminal states (T) located at (0, 0) and (0, 4).
  - The overall structure of the policies shows a tendency to move upwards ($\uparrow$) or sideways ($\rightarrow$ or $\leftarrow$) to avoid penalties.
- **Differences:**
  - SARSA's policy includes more diverse actions, reflecting its on-policy nature and tendency to consider the current policy. The SARSA policy involves a mix of up ($\uparrow$), right ($\rightarrow$), and left ($\leftarrow$) moves, with some paths leading directly to terminal states (T) and others occasionally encountering red states (R) and having to restart.
  - Q-Learning's policy is more deterministic and consistent, reflecting its off-policy nature and focus on optimal actions. The Q-learning policy shows a more consistent pattern of upward ($\uparrow$) movements, indicating a more direct path towards terminal states while minimizing the risk of encountering red states.

Reasoning:

- SARSA considers the action actually taken and updates Q-values based on that, leading to policies that might reflect exploration tendencies.
- Q-Learning updates Q-values based on the maximum future reward possible, leading to more optimized and consistent policies.

1.7 Sum of Rewards over Episodes

The sum of rewards over episodes for both SARSA and Q-Learning is shown in the provided graph.

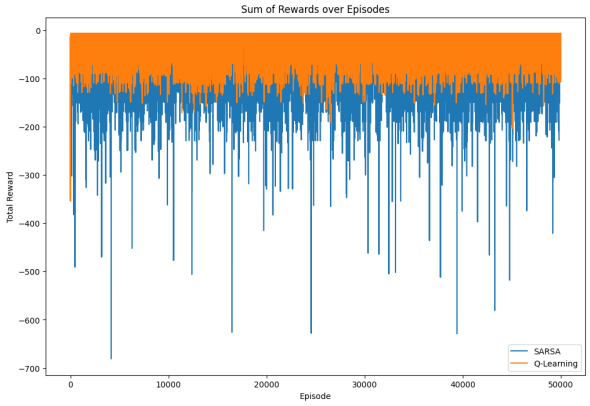

Figure 1: Sum of Rewards over Episodes for SARSA and Q-Learning. The graph shows the total reward per episode for both SARSA (blue) and Q-Learning (orange) algorithms over 50,000 episodes. While both methods experience significant fluctuations in rewards, Q-Learning demonstrates a quicker stabilization in performance compared to SARSA.

1.8 Behavior Analysis

- **SARSA:**
  - The total reward fluctuates significantly across episodes, indicating continuous exploration and learning.
  - There are sharp declines due to penalties from hitting red states and boundaries.
- **Q-Learning:**
  - The total reward also fluctuates but tends to stabilize more quickly compared to SARSA.
  - The policy converges faster to optimal actions, leading to a more consistent sum of rewards over time.

Explanation: The fluctuations in rewards for both algorithms are due to the ϵ-greedy action selection, which introduces exploration. SARSA's on-policy nature results in more variability, while Q-Learning's off-policy nature results in faster convergence to optimal actions.

By comparing the sum of rewards and the policies derived, it can be concluded that Q-Learning tends to provide a more stable and optimized policy faster, while SARSA shows more exploration before converging to an optimal policy.

2 Problem 2

2.1 Problem Summary

In this task, we are given a 7x7 gridworld where an agent performs a random walk, starting at the center of the grid. The agent can move up, down, left, or right with equal probability. Two terminal states exist: the lower left corner with a reward of -1 and the upper right corner with a reward of 1. The goal is to compute the value function for this random walk policy using two different methods: Gradient Monte Carlo and Semi-gradient TD(0) with an affine function approximation. We will then compare these computed value functions against the exact value function obtained via a dynamic programming approach.

2.2 Gridworld Layout

- **Grid Size:** 7x7 grid.
- **Start State:** The center of the grid (3, 3).

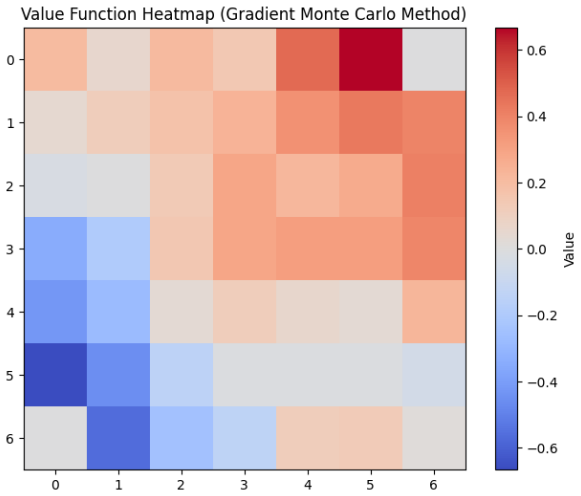

Figure 2: Heatmap of the Value Function computed using the Gradient Monte Carlo Method.

- **Terminal States:**
  - Lower left corner (6, 0) with a reward of -1.
  - Upper right corner (0, 6) with a reward of 1.
- **Rewards:**
  - The agent receives a reward of 0 for moving to any non-terminal state.
  - An attempt to move outside the grid results in the agent staying in the same spot, also with a reward of 0.

2.3 Objectives

- Implement the Gradient Monte Carlo method to estimate the value function for the gridworld environment.
- Implement the Semi-gradient TD(0) method with an affine function approximation to estimate the value function.
- Compare the value functions obtained from both methods against the exact value function calculated using dynamic programming.

2.4 Solving using Gradient Monte Carlo

2.4.1 Method Explanation

Gradient Monte Carlo is an on-policy method for estimating the value function by updating the weights based on the returns observed in each episode. In this approach, the value function is represented as a linear combination of features, and the weights are updated after each episode to reduce the error between the predicted and observed returns.

2.4.2 Output Analysis

The value function computed using the Gradient Monte Carlo method is displayed in both a pandas DataFrame and a heatmap. The heatmap shows how values are spread across the grid, indicating that the values are generally higher near the upper right corner and lower near the lower left corner, aligning with the rewards of the terminal states. However, the Monte Carlo method introduces some noise and inconsistency in the middle of the grid due to the variability in episode outcomes and the random nature of the method.

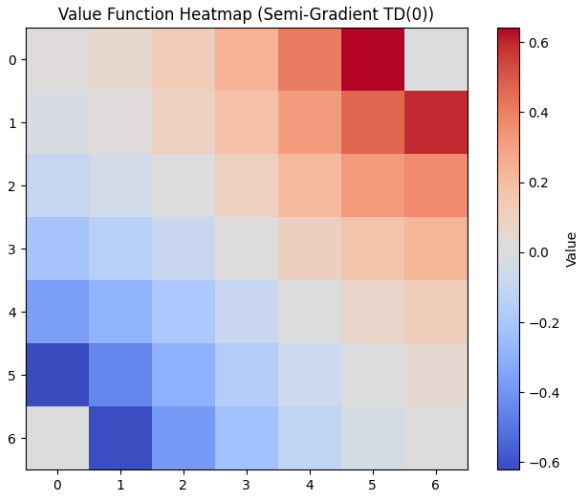

Figure 3: Heatmap of the Value Function computed using the Semi-Gradient TD(0) Method.

2.4.3 Pandas DataFrame Output

	0	1	2	3	4	5	6
0	0.473869	0.423383	0.614166	0.457046	0.376787	0.506075	0.000000
1	0.501330	0.390167	0.453172	0.481317	0.534417	0.539509	0.674654
2	0.421475	0.302586	0.425677	0.465863	0.532151	0.559630	0.602029
3	0.235818	0.121414	0.308247	0.381382	0.483563	0.533808	0.378284
4	-0.090196	-0.012643	0.055756	0.259148	0.318169	0.333981	0.234439
5	-0.440782	-0.173070	-0.053688	0.171935	0.199711	0.286275	0.235614
6	0.000000	-0.201045	-0.090632	0.029243	0.034282	0.211983	0.337047

2.5 Solving using Semi-gradient TD(0) method with an affine function approximation

2.5.1 Method Explanation

Semi-gradient TD(0) is an on-policy method that estimates the value function using temporal differences. This method updates the weights incrementally as the agent interacts with the environment, reducing the temporal difference error after each step rather than waiting for the end of an episode.

2.5.2 Output Analysis

The value function computed using the Semi-gradient TD(0) method is more consistent and smoother than that of the Gradient Monte Carlo method. The values transition more gradually from the lower left to the upper right corners, showing that this method captures the overall trend more effectively. The differences are less pronounced, but some deviations from the exact value function are still evident.

2.5.3 Pandas DataFrame Output

	0	1	2	3	4	5	6
0	0.019319	0.058530	0.126487	0.234172	0.404632	0.640463	0.000000
1	-0.014682	0.017168	0.095053	0.186024	0.311554	0.464921	0.592670
2	-0.090252	-0.043025	0.008642	0.100421	0.207670	0.320983	0.367894
3	-0.211278	-0.147712	-0.082251	0.012727	0.105166	0.169792	0.225372
4	-0.368265	-0.284140	-0.191946	-0.081416	0.009086	0.074661	0.122272
5	-0.621301	-0.447600	-0.293203	-0.163077	-0.065275	0.000153	0.045192
6	0.000000	-0.616051	-0.380929	-0.222634	-0.104685	-0.029051	0.011298

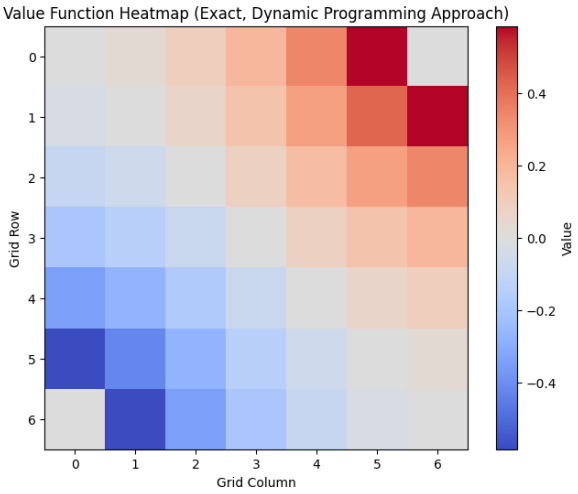

Figure 4: Heatmap of the Exact Value Function computed using Dynamic Programming Approach.

2.6 Comparing Both Methods Against Exact Value Function

2.6.1 Exact Value Function

The exact value function was computed using a dynamic programming approach, which iteratively updates the value function until convergence. This method produces the most accurate value function, representing the true expected returns from each state.

2.6.2 Pandas DataFrame Output

	0	1	2	3	4	5	6
0	1.749387e-18	3.068081e-02	0.093282	0.192963	0.344374	5.845415e-01	0.000000e+00
1	-3.068081e-02	2.602085e-18	0.059972	0.149028	0.269533	4.227669e-01	5.845415e-01
2	-9.328206e-02	-5.997172e-02	0.000000	0.079666	0.172853	2.695331e-01	3.443744e-01
3	-1.929626e-01	-1.490279e-01	-0.079666	0.000000	0.079666	1.490279e-01	1.929626e-01
4	-3.443744e-01	-2.695331e-01	-0.172853	-0.079666	0.000000	5.997172e-02	9.328206e-02
5	-5.845415e-01	-4.227669e-01	-0.269533	-0.149028	-0.059972	6.938894e-18	3.068081e-02
6	0.000000e+00	-5.845415e-01	-0.344374	-0.192963	-0.093282	-3.068081e-02	4.336809e-18

2.6.3 Comparison

- **Gradient Monte Carlo vs. Exact:** The Gradient Monte Carlo method provides a relatively good approximation of the exact value function but introduces noise and irregularities, especially in areas far from the terminal states. This noise arises from the variability in the returns and the randomness inherent in Monte Carlo methods.
- **Semi-gradient TD(0) vs. Exact:** The Semi-gradient TD(0) method provides a closer approximation to the exact value function than the Gradient Monte Carlo method. It captures the general trend from negative to positive values more smoothly and with less variance, demonstrating its advantage in reducing temporal difference errors more effectively.