REINFORCEMENT LEARNING: POLICY EVALUATION USING MONTE CARLO METHODS AND TEMPORAL DIFFERENCE LEARNING

Mahlet Nigussie Tesfaye
Addis Ababa Institute of Technology, SiTE
rigbe.rmn@gmail.com
Addis Ababa, Ethiopia.
February 16- 2024

1 REPORT: KEY LEARNING POINTS FROM MONTE CARLO POLICY EVALUATION IN BLACKJACK

1.1 IMPORTANCE OF EPISODIC TASKS IN MC METHODS

Monte Carlo (MC) methods are particularly well-suited for episodic tasks, where the interaction between the agent and the environment can be divided into complete episodes with a clear terminal state. In the context of Blackjack:

- Each episode represents a single game of Blackjack, starting from the initial deal of cards and ending when the player either sticks, goes bust (exceeds 21), or the dealer finishes their turn.
- MC methods require complete episodes because they rely on observing the total return (cumulative reward) from a state until the end of the episode. This is in contrast to temporal difference methods, which can update value estimates incrementally without waiting for the episode to end.

The episodic nature of Blackjack makes it an ideal candidate for MC methods, as the returns can only be computed after the episode terminates. This highlights the importance of episodic tasks in reinforcement learning, particularly when the environment has a clear terminal state.

1.2 HOW MC METHODS RELY ON COMPLETE EPISODES TO COMPUTE RETURNS

MC methods estimate the value function $V(s)$ by averaging the returns observed after visiting state s across multiple episodes. In this exercise:

- The agent follows a fixed policy (e.g., "stick if sum $\geq$ 18, otherwise hit") to generate episodes.
- For each episode, the return G is computed as the cumulative discounted reward from the first visit to state s until the end of the episode:
$$G = \sum_{t=0}^{T} \gamma^t R_{t+1}$$
where γ is the discount factor (set to 1.0 in this case) and R_{t+1} is the reward at time $t+1$.
- The value function $V(s)$ is updated by averaging the returns for each state across all episodes.

MC methods require complete episodes because they need the full sequence of rewards from a state to the terminal state to compute the return. This makes MC methods more data-intensive but also more accurate in estimating the value function for episodic tasks.

1.3 IMPACT OF THE POLICY ON THE VALUE FUNCTION

The policy directly influences the value function $V(s)$, as it determines the actions taken in each state and, consequently, the returns observed. In this exercise:

- A **fixed policy** ("stick if sum $\geq$ 18, otherwise hit") was evaluated using MC methods. This policy is simple but suboptimal, as it does not consider the dealer's visible card.
- A **dynamic policy** was also evaluated, which adjusts its behavior based on the dealer's visible card. For example, it sticks if the player's sum is $\geq$ 18 or if the dealer's card is low (2-6), otherwise it hits.

Comparison of Policies:

- The fixed policy tends to be more conservative, sticking only when the player's sum is high. This can lead to lower expected returns in certain states, especially when the dealer's card is favorable (e.g., 2-6).
- The dynamic policy, on the other hand, takes into account the dealer's card and adjusts its strategy accordingly. This leads to higher expected returns in states where the dealer is likely to bust.

The choice of policy has a significant impact on the value function $V(s)$. A well-designed policy that considers additional information (e.g., the dealer's card) can lead to better outcomes and higher expected returns. This demonstrates the importance of policy design in reinforcement learning and how it directly affects the agent's performance.

1.4 Visualization of Results

The exercise included two key visualizations to demonstrate the learning points:

1.4.1 Heatmap of $V(s)$

The heatmap (Figure 1) shows the estimated value function for each state (player's sum vs. dealer's visible card) under the fixed policy. The heatmap reveals that states with higher player sums (e.g., 20 or 21) and low dealer cards (e.g., 2-6) had higher values, as the player is more likely to win in these scenarios.

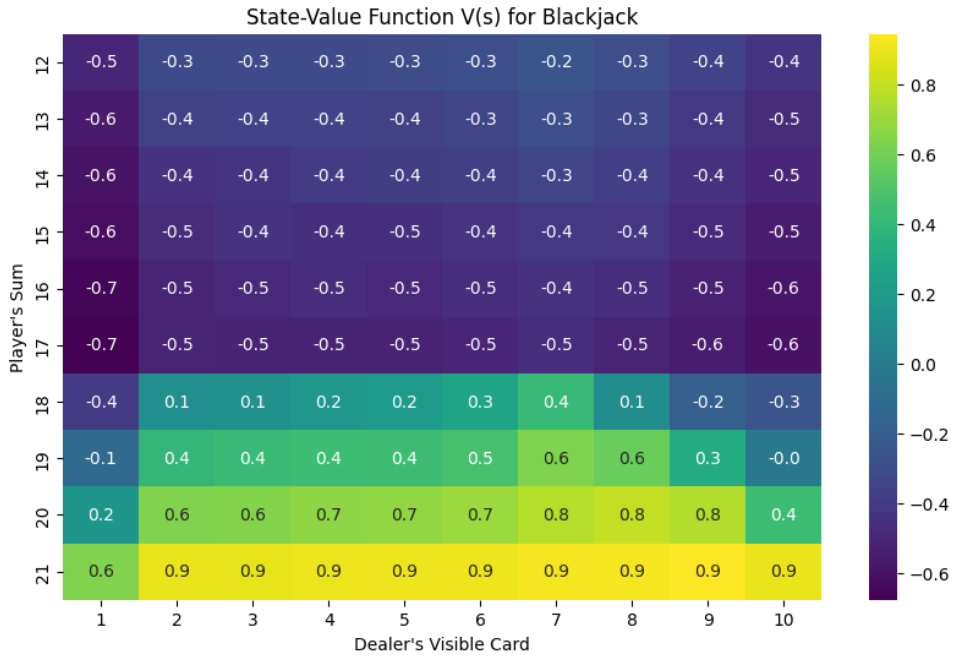

Figure 1: Heatmap of $V(s)$ for the fixed policy.

1.4.2 Convergence Plot

The convergence plot (Figure 2) shows how the value estimate for a specific state (e.g., (20, 10, False)) converged over episodes. The plot demonstrates that MC methods require a large number of episodes to accurately estimate the value function, highlighting the data-intensive nature of these methods.

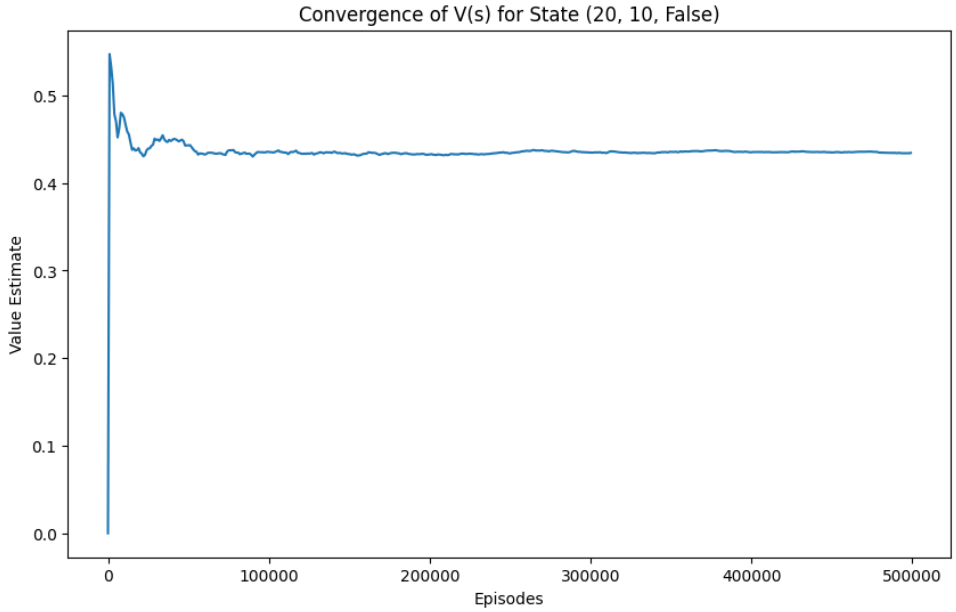

Figure 2: Convergence of $V(s)$ for state (20, 10, False).

1.5 Conclusion

Through this exercise, we learned:

- The importance of episodic tasks in MC methods, as they require complete episodes to compute returns.
- How MC methods rely on averaging observed returns to estimate the value function, making them data-intensive but accurate for episodic tasks.
- The significant impact of the policy on the value function, with better-designed policies leading to higher expected returns.

These insights are fundamental to understanding reinforcement learning and the role of Monte Carlo methods in solving episodic tasks like Blackjack. The exercise also highlighted the importance of policy design and the use of visualizations to analyze and interpret the results of reinforcement learning algorithms.

2 Report: Temporal Difference - Cliff Walking with SARSA

2.1 Introduction

This report describes the implementation of the SARSA algorithm to solve the Cliff Walking environment. SARSA is an on-policy Temporal Difference (TD) control method that learns the optimal policy while following an -greedy exploration strategy.

2.2 Environment Setup

The Cliff Walking environment is a 4x12 grid where:

- The agent starts at the bottom-left corner.
- The goal is to reach the bottom-right corner.
- Stepping into the ”cliff” (middle cells in the bottom row) resets the agent to the start and incurs a high penalty (-100 reward).

2.3 SARSA Algorithm

The SARSA algorithm was implemented with the following components:

- Q-table: Initialized to zeros, representing the action-value function $Q(s, a)$.
- -greedy policy: Balances exploration and exploitation by selecting random actions with probability ϵ and greedy actions otherwise.
- SARSA update rule: Updates the Q-value using:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$$

where α is the learning rate and γ is the discount factor.

2.4 Results

2.4.1 Cumulative Reward per Episode

The cumulative reward per episode (Figure 3) shows the agent's learning progress. The rewards increase over episodes, indicating that the agent is learning to avoid the cliff and reach the goal.

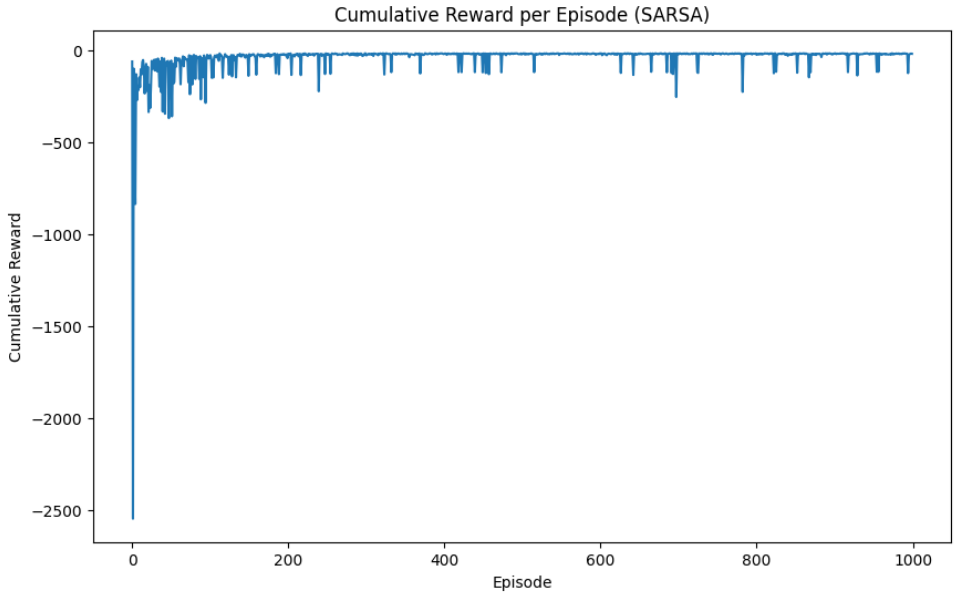

Figure 3: Cumulative Reward per Episode (SARSA).

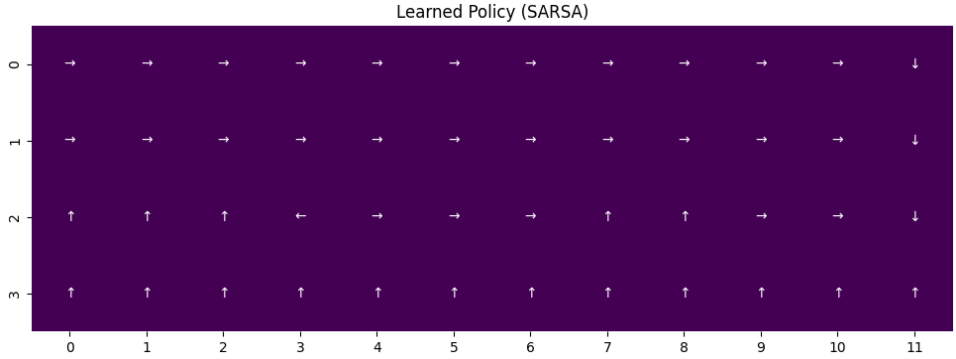

Figure 4: Learned Policy (SARSA).

2.4.2 Learned Policy

The learned policy (Figure 4) is visualized as a heatmap with arrows indicating the optimal action in each state. The agent learns to navigate around the cliff and reach the goal efficiently.

2.5 Conclusion

The SARSA algorithm successfully solves the Cliff Walking environment by learning an optimal policy under the -greedy exploration strategy. The results demonstrate the effectiveness of Temporal Difference methods in reinforcement learning tasks.