Reinforcement Learning: `GridWorld` Extended

Gavin Hull
Memorial University of Newfoundland and Labrador
ghull@mun.ca

August 7, 2024

Introduction

Reinforcement learning (RL) is a machine learning paradigm where agents learn to make decisions by interacting with their environment. This project delves deeper into RL concepts, building upon our previous exploration of the `GridWorld` environment. We investigate more sophisticated algorithms and their applications to increasingly complex scenarios.

This study focuses on four key algorithms: Q-Learning, SARSA, Gradient Monte Carlo, and Semi-Gradient TD(0). Each of these methods offers a unique approach to solving reinforcement learning problems, and in this project we compare the effectiveness, efficiency, and convergence properties of each approach across various `GridWorld` tasks.

By examining these algorithms in different contexts, we aim to provide a comprehensive understanding of their strengths, weaknesses, and appropriate use cases in reinforcement learning problems. All code is made available through GitHub. [1]

Environment Setup

To facilitate our experiments, we inherit the custom `GridWorld` class detailed in our previous work [2]. This class provides the flexibility and efficiency required to execute the various algorithms under study. The environment can be configured with different reward structures, obstacles, and terminal states, allowing us to create diverse scenarios for our investigations.

Part 1: Optimal Policies

In this section, we explore a 5×5 `GridWorld` environment where the agent's goal is to reach either of two terminal states as efficiently as possible while avoiding a wall of special states which reset the agent's position with a reward of -20. We compare the performance of Q-Learning and SARSA to that of policy improvement with value iteration in finding optimal policies for this task.

[1] All code required to replicate results can be found here: https://github.com/CallMeTwitch/Math-4250/tree/main/GridWorldExtended

[2] The code and report for our previous work can be found here: https://github.com/CallMeTwitch/Math-4250/tree/main/GridWorld

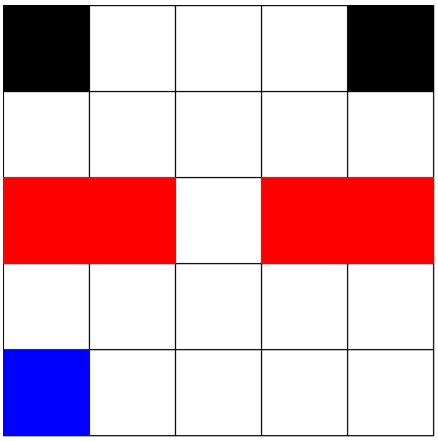

Figure 1: Environment grid A.

1.1 Policy Improvement with Value Iteration

Before diving into Q-Learning and SARSA, we first establish a baseline using policy improvement with value iteration. This method provides a theoretically optimal policy against which to compare our other algorithms. An explicit pseudo-code implementation can be found in Appendix A.

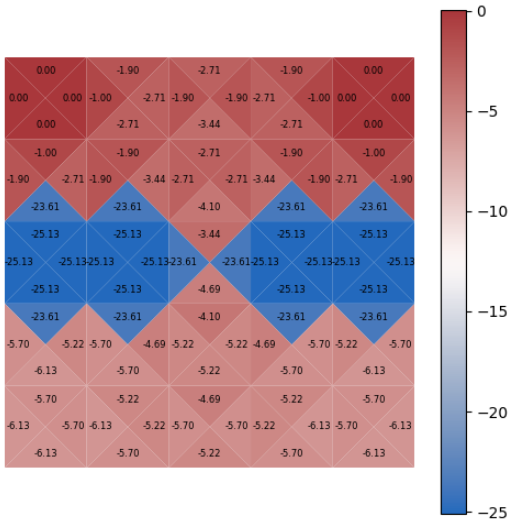

Figure 2: Optimal action-value function for the 5×5 `GridWorld` environment, derived using policy improvement with value iteration.

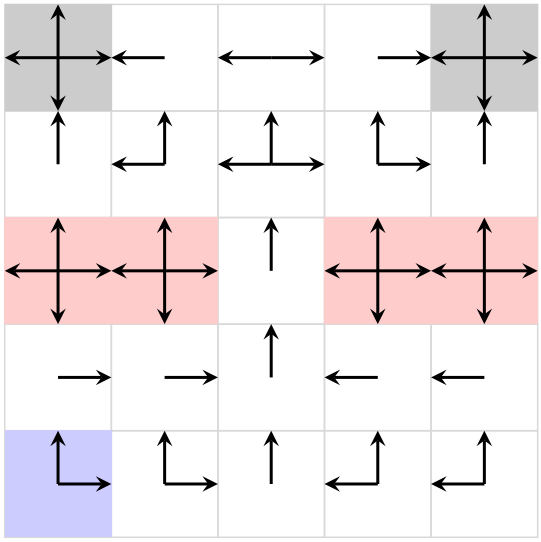

Figure 3: Optimal policy for the 5×5 `GridWorld` environment, derived using policy improvement with value iteration.

As anticipated, the optimal policy directs the agent toward and through the gap in the wall, subsequently guiding it towards either terminal state with equal preference. The value of entering the wall is approximately equivalent to the cumulative values of progressing optimally from the initial cell through the wall, which is consistent with the theoretical expectation. Additionally, the value decreases with distance from the terminal states, reflecting the accumulating negative rewards.

1.2 SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy temporal difference learning algorithm, named for its use of the variables $S_t, A_t, R_t, S_{t+1}, A_{t+1}$. This method can experience increased variance in its updates due to the lack of diversity in its data, unlike off-policy algorithms which collect data from various policies to ensure data diversity and thus update stability. An explicit pseudo-code implementation can be found in Appendix B.

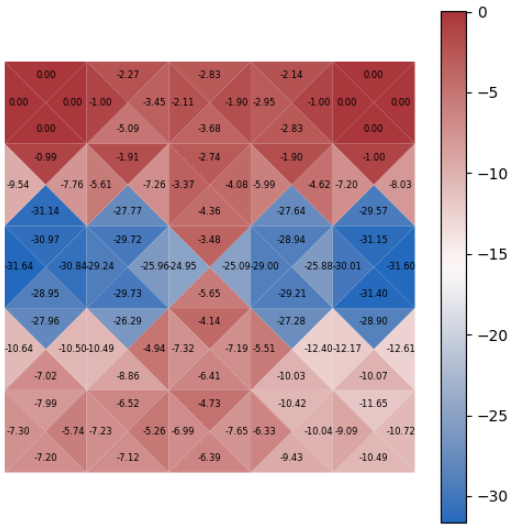

Figure 4: Optimal action-value function for the 5×5 `GridWorld` environment, computed using the SARSA algorithm.

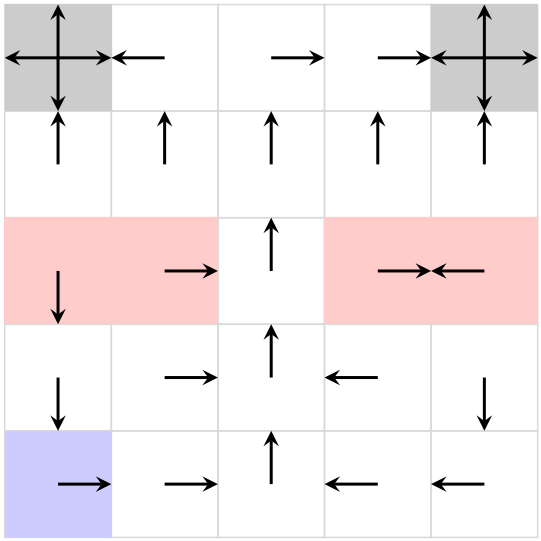

Figure 5: Optimal policy for the 5×5 `GridWorld` environment, computed using SARSA.

The SARSA algorithm, while effective, exhibits incomplete convergence as evidenced by the bias toward a specific direction within the special states; instead, it should favour all directions equally as all actions result in the same next state and reward. This partial convergence could likely be mitigated by increasing the epsilon parameter or the number of episodes. Despite this, the algorithm was still successful in identifying an optimal path, albeit with some inconsistencies.

1.3 Q-Learning

Unlike SARSA, Q-Learning is an off-policy temporal difference learning algorithm. This method often achieves much more stable convergence due to its diverse experiences, stemming from its off-policy nature. An explicit pseudo-code implementation can be found in Appendix C.

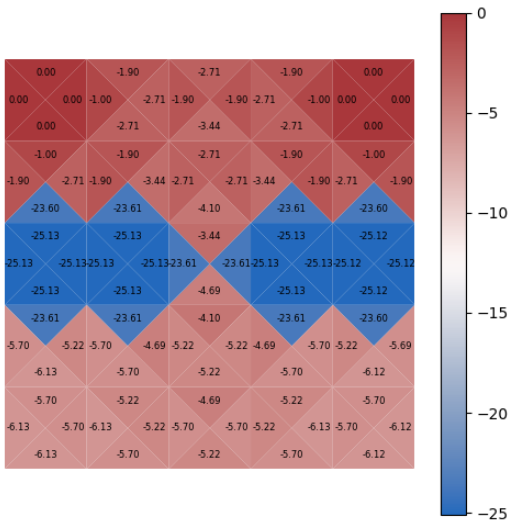

Figure 6: Optimal action-value function for the 5×5 `GridWorld` environment, derived using Q-Learning.

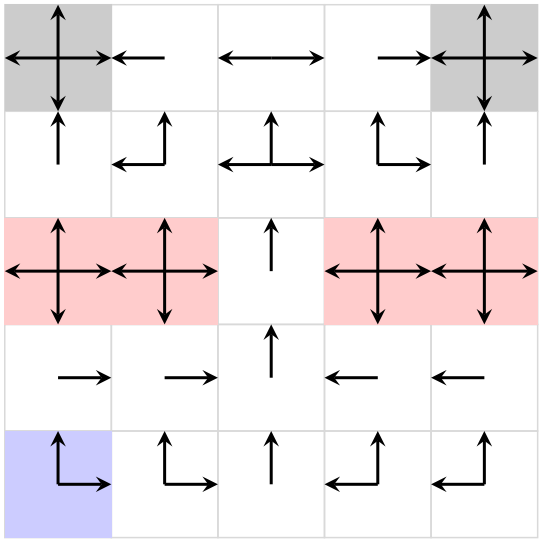

Figure 7: Optimal policy for the 5×5 `GridWorld` environment, derived using Q-Learning.

The Q-Learning algorithm demonstrated remarkable convergence, yielding results that closely align with those obtained with the value iteration method. The consistency between Q-Learning and value iteration underscores the accuracy and reliability of the off-policy algorithms in environments where diverse data is easily accumulated, such as `GridWorld`.

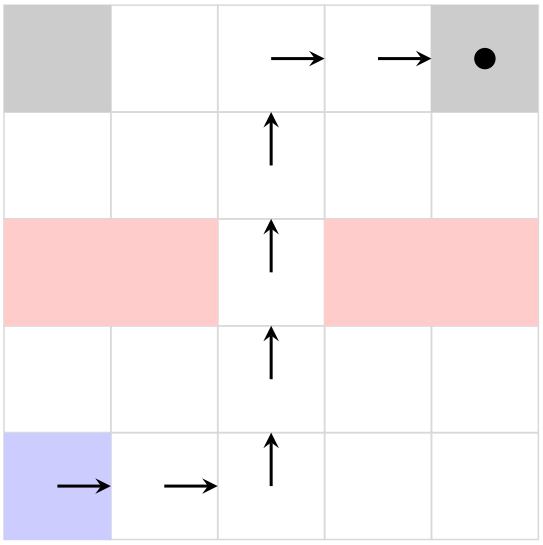

Figure 8: Sample trajectory for the 5×5 `GridWorld` environment, derived using SARSA.

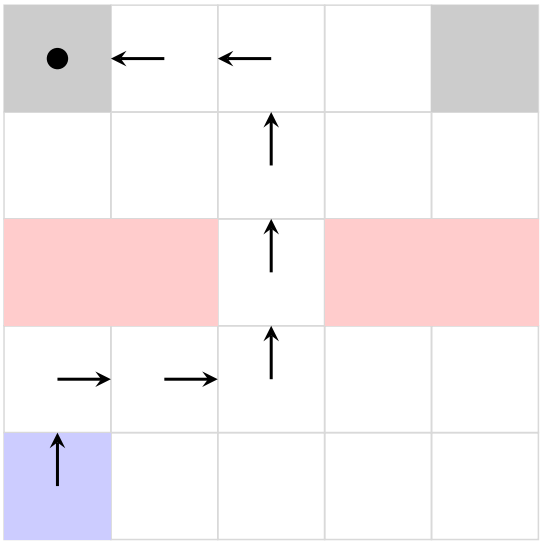

Figure 9: Sample trajectory for the 5×5 `GridWorld` environment, derived using Q-Learning

The paths derived from Q-Learning and SARSA, although distinct, both achieve optimality within the context of the `GridWorld` environment. The variation arises from differences in the

action-value grids, yet neither path is superior to the other. The analysis of cumulative rewards over episodes indicates that Q-Learning not only converges faster than SARSA but also exhibits greater consistency, with less variance in the loss. We posit that this is due to their off-policy and on-policy natures respectively. This observation is supported by the reward plots provided in Appendix D.

Part 2: Random Walk Value Function

In this section, we alter the environment as displayed below and described in the assignment instructions. We employ Gradient Monte Carlo and Semi-Gradient TD(0) to derive the value function of a random walk in this environment. Additionally, we use an explicit solution method to find this value function to ensure the veracity of the value functions derived using these methods.

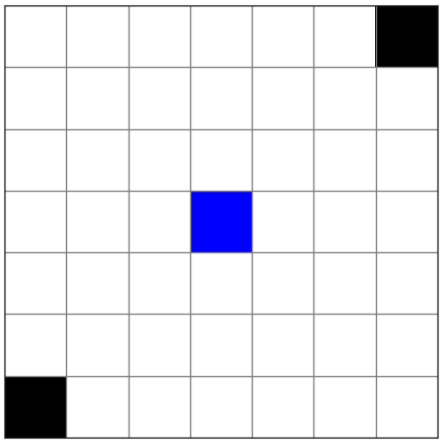

Figure 10: Environment grid B.

2.1 Explicit Solution Method

The explicit solution method directly solves the system of Bellman equations using techniques from linear algebra. This approach provides an efficient route to the exact solution but becomes exponentially more computationally intensive as the state space increases. This method acts as a baseline to compare the approximations obtained from Gradient Monte Carlo and Semi-Gradient TD(0) methods. An explicit pseudo-code implementation can be found in Appendix E.

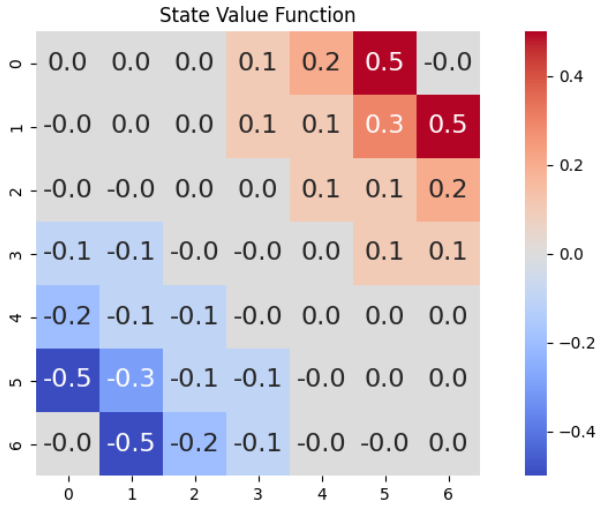

Figure 11: Heatmap of the value function with equiprobable policy in the 7×7 `GridWorld` environment, solved explicitly.

As anticipated, the value function derived via the explicit solution method exhibits higher values near the top-right corner and lower values near the bottom-left corner. The large collection of observed zeros along the central diagonal are likely the result of rounding errors, which could be mitigated by increasing the precision of the figure. It is noteworthy that the value of cells proximal to terminal states approximates the cumulative value of adjacent cells, which is consistent with theoretical expectations.

2.2 Gradient Monte Carlo

The Gradient Monte Carlo method estimates the value function by averaging returns observed from sampled episodes. An explicit pseudo-code implementation can be found in Appendix F.

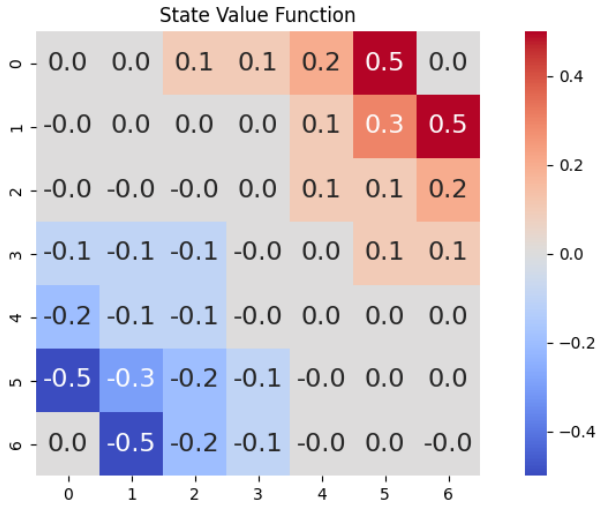

Figure 12: Heatmap of the value function with equiprobable policy in the 7×7 `GridWorld` environment, derived using Gradient Monte Carlo methods.

The Gradient Monte Carlo method, while semi-convergent, did not achieve perfect accuracy. The values of some cells are marginally above or below those obtained through the explicit method despite overall consistency between the two sets of results suggesting correct implementation. This minor discrepancy is characteristic of Monte Carlo algorithms, which are inherently stochastic.

2.3 Semi-Gradient TD(0)

The Semi-Gradient TD(0) method updates the value function using the temporal difference error. Due to the simplicity of this environment, we use affine function approximation in this particular instance of the algorithm. An explicit pseudo-code implementation can be found in Appendix G.

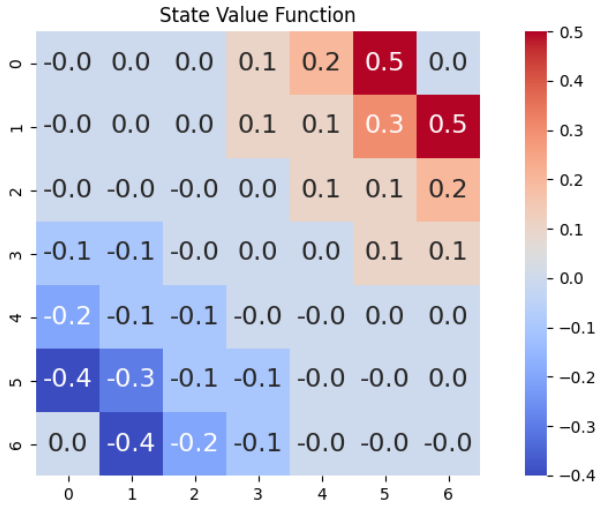

Figure 13: Heatmap of the value function with equiprobable policy in the 7×7 `GridWorld` environment, derived using the Semi-Gradient TD(0) method.

The value function derived using the Semi-Gradient TD(0) method closely matches the explicit solution, except for cells adjacent to the lesser terminal state, which are uniformly 0.1 less. The symmetry across the central line and overall consistency with the explicit solution further corroborate the veracity of the implementation.

The comparison of value functions derived from Gradient Monte Carlo and Semi-Gradient TD(0) methods with the explicit solution reveals that both approximate methods are effective, albeit with minor discrepancies. The consistency and symmetry observed in both methods reinforce their validity, highlighting the robustness of these algorithms in approximating value functions in reinforcement learning tasks.

Part 3: Adversarial `GridWorld` Environments (Bonus)

In this section we consider a 5×5 *adversarial* `GridWorld` environment where a 'predator' agent starting in the top-left corner receives a reward of $+1$ when it catches a 'prey' agent starting in the bottom-right corner, which receives a reward of -1 when caught. Their goals should be to catch/avoid each other for as long as possible without falling off of the grid. We will use Q-Learning and SARSA methods to solve for the optimal policy in this environment, but will not be comparing them to an explicit solution. This is because an explicit solution likely does not exist: it would require a Nash equilibrium (where neither agent can benefit from changing their policy.)

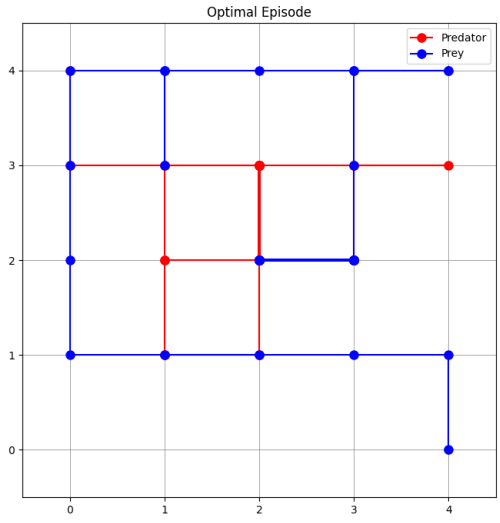

Figure 14: Path generated by 'optimal' SARSA agents.

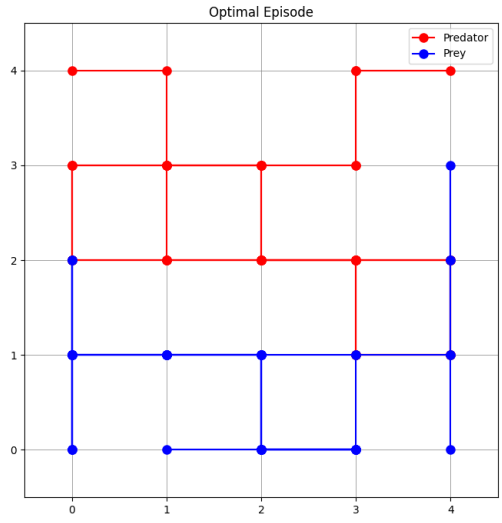

Figure 15: Path generated by 'optimal' Q-Learning agents.

The paths generated by the 'optimal' policies of SARSA and Q-Learning agents illustrate their learned strategies. The predator tends to control the center, while the prey stays near the edges, which makes sense as the the predator would want to maximize the number of cells it has access to, and the prey would want to minimize the number of cells which have access to it. Analysis of the provided gifs [3] shows that both agents learn effective, albeit not optimal, policies. Neither algorithm shows any particular advantage over the other.

Note that the prey has a 'winning' strategy whenever it is one cell from the predator: always move to the cell the predator is coming from. The only way for the predator to remain in place is to walk off the grid, nullifying any reward it would receive from catching the predator. For this reason, the predator maintains a two-grid space when unable to reach the prey, knowing that a one-grid space allows the prey to predict and evade the predator. Conversely, the prey tries to reduce the two-grid space to one grid by moving to the map edges.

The reward plots in Appendix H illustrate the dynamics of learning. Initially, the predator's reward increases as it learns to catch the prey, but it stabilizes once the prey adapts to avoid capture, leading to a balanced reward dynamic where the prey is consistently able to out-maneuver the predator.

4 Conclusion

This project provided an in-depth exploration of various reinforcement learning techniques applied to diverse implementations of the `GridWorld` problem. Through the implementation and analysis of methods such as value iteration, Q-Learning, SARSA, Gradient Monte Carlo, and Semi-Gradient TD(0), we demonstrated the strengths and limitations of each approach in solving Markov Decision Processes. The comparative analysis offered valuable insights into the performance and applicability of these algorithms, contributing to a deeper understanding of core reinforcement learning

[3] Gifs of Q-Learning and SARSA agents can be found on GitHub: https://github.com/CallMeTwitch/Math-4250/tree/main/GridWorldExtended

concepts. The consistency of our results with theoretical expectations underscores the robustness and reliability of the implemented methods. All code and detailed results are made available for reproducibility in alignment with modern standards in machine learning research.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Appendices

All algorithms described in these appendices are from Sutton. [1]

Appendix A

Algorithm 1 Policy Improvement with Value Iteration

Initialize $V(s)$ arbitrarily for all states s
repeat
    $\Delta \leftarrow 0$
    for each state s **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$ (a small threshold)
$\pi^*(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

Appendix B

Algorithm 2 Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
loop
    Initialize S
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    repeat
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$
        $A \leftarrow A'$
    until S is terminal
end loop

Appendix C

Algorithm 3 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$ for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
loop
    Initialize S
    repeat
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until S is terminal
end loop

Appendix D

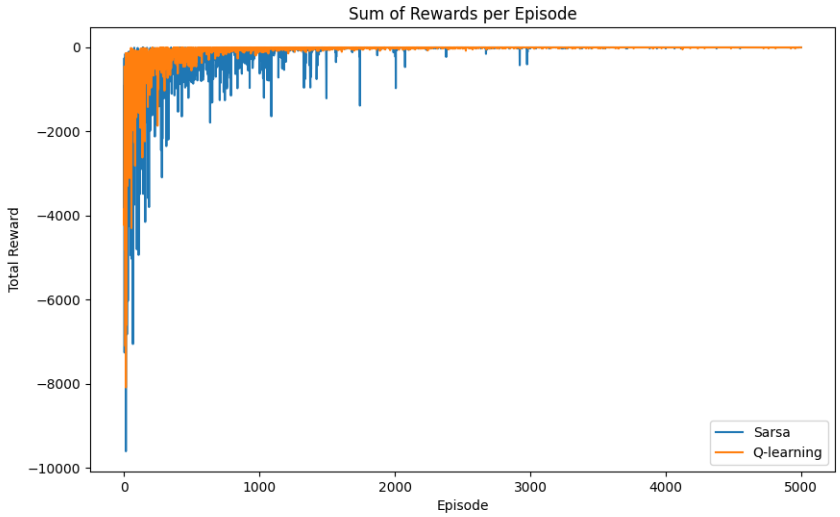

Figure 16: Environment grid A.

Appendix E

Algorithm 4 Explicit Solution for the System of Bellman Equations

Compute state transition probabilities:
$p(s'|s) = \sum_{a,r} \pi(a|s)p(s', r|s, a)$
Compute expected rewards:
$R(s) = \sum_{a,r,s'} \pi(a|s)p(s', r|s, a)R(s, a)$
Solve for value function:
$V = (I - \gamma p)^{-1}R$

Appendix F

Algorithm 5 Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: policy π to be evaluated
Input: differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
loop ▷ for each episode
    Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
    for $t = 0, 1, \ldots, T - 1$ **do**
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}(S_t, \mathbf{w})]\nabla\hat{v}(S_t, \mathbf{w})$
    end for
end loop

Appendix G

Algorithm 6 Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: policy π to be evaluated
Input: differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal}, \cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
loop ▷ for each episode
    Initialize S
    repeat
        Choose $A \sim \pi(\cdot|S)$
        Take action A, observe R, S'
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha[R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})]\nabla\hat{v}(S, \mathbf{w})$
        $S \leftarrow S'$
    until S is terminal
end loop

Appendix H

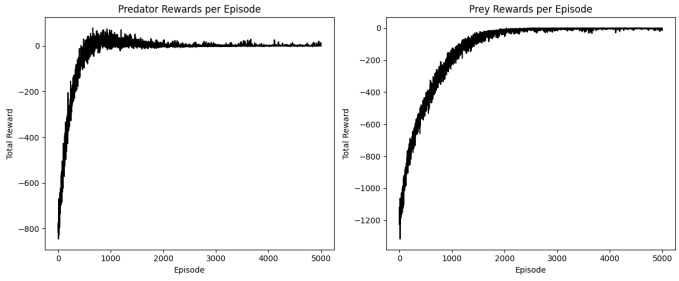

Figure 17: Reward per episode for SARSA.

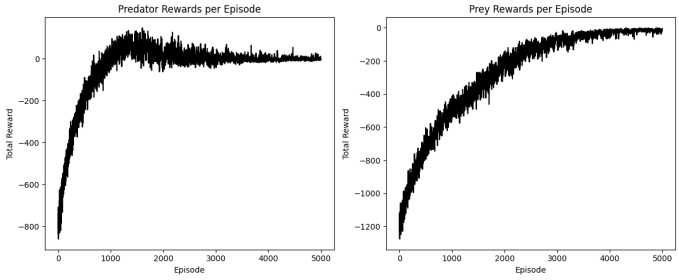

Figure 18: Reward per episode for Q-Learning.