AE598 Homework 2: Monte Carlo and TD(0)

Raymond Chen

rc18@illinois.edu

I. Introduction

Previously, our reinforcement learning problem was modeled as a Markov Decision Process (MDP) with known transition dynamics. However, in many real-world settings, the dynamics of the environment are not available to the agent. In this project, I use the Frozen Lake environment, represented as a 4×4 grid with terminal goal and hole states, to investigate model-free tabular methods.

Using one Monte Carlo method (on-policy first-visit MC control) and two temporal-difference (TD) methods (SARSA and Q-Learning), I evaluate their performance in learning optimal policies. The objective is to analyze how these algorithms perform in both deterministic (non-slippery) and stochastic (slippery) variants of the environment, and to understand how the underlying environment dynamics influence the resulting value functions and learned strategies.

II. Methods

A. Initialization

The Frozen Lake environment is initialized analogously to an MDP. The state space consists of integers 0–15, representing the cells of a 4×4 grid indexed left to right, top to bottom. The action space consists of integers 0–3, corresponding to movements *left*, *down*, *right*, and *up*, respectively.

States 5, 7, 11, and 12 are terminal hole states, and state 15 is the terminal goal state. The reward function assigns $+1.0$ for reaching the goal and 0 otherwise. Because the transition function is unknown, both the policy and the state-action value function are updated through episodic interactions with the environment based solely on observed rewards.

The initial policy assigns equal probability to all actions in every state, and the initial action-value function $Q(s, a)$ is initialized to zero for all state–action pairs. The hyperparameters are set as follows: step size $\alpha = 0.2$ and discount factor $\gamma = 0.95$. The exploration rate ϵ is varied between environments: $\epsilon = 0.2$ for the deterministic (non-slippery) environment to encourage exploration, and $\epsilon = 0.1$ for the stochastic (slippery) case.

B. On-Policy Monte Carlo First-Visit

The on-policy first-visit Monte Carlo (MC) control algorithm is described in *Reinforcement Learning: An Introduction* by Sutton and Barto, and illustrated in Figure 1. The policy follows an ϵ-soft strategy, where each non-greedy action has a small probability ϵ of being selected to ensure exploration.

In each iteration, a full episode from the start state to a terminal state is generated. The return G_t is computed for each time step t and appended to the returns list if the state–action pair (S_t, A_t) appears for the first time in that episode. The action-value estimate $Q(S_t, A_t)$ is updated as the average of the observed returns, and the policy is updated toward ϵ-soft optimality.

To improve computational efficiency, the returns list is implemented as cumulative sums and counts rather than storing all past returns explicitly.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Fig. 1: On-policy first-visit Monte Carlo control algorithm.

C. SARSA: On-Policy Temporal-Difference Control

SARSA is a temporal-difference (TD) control algorithm that combines aspects of Monte Carlo and dynamic programming methods. Unlike MC methods, SARSA updates its value estimates after each step using bootstrapping rather than waiting until the end of an episode.

For each episode, the initial state S is set to the starting position, and the action-value function Q is initialized to zero. The policy is derived from Q using an ϵ-greedy strategy. After taking an action A, the agent observes the reward R and next state S', selects the next action A' according to the current policy, and updates:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A) \right].$$

D. Q-Learning: Off-Policy Temporal-Difference Control

Q-learning is closely related to SARSA, but it differs in its update target. Rather than following the current ϵ-greedy policy, Q-learning learns the optimal policy directly by updating toward the maximum estimated value:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right].$$

Here, Q approximates the optimal action-value function q^* independently of the behavior policy. This off-policy nature

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

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Fig. 2: SARSA algorithm.

typically leads to faster convergence in deterministic environments, though it may introduce instability in stochastic settings.

III. Results

A. Slippery Environment

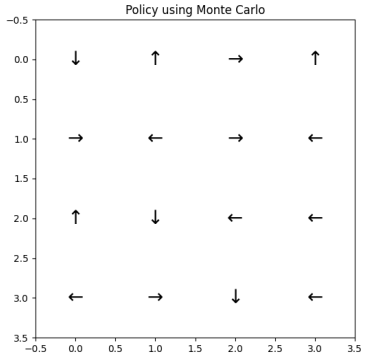
In the stochastic environment, the policies produced from Monte-Carlo, SARSA, and Q-learning were fairly different, although the best action for each state was to traverse in the direction of the goal state while minimizing hole states in both intended action and stochastic "slip" actions (Figure 3). For example, although state 6 for the three algorithms differed in best actions, all three best action was to transition towards a hole state, because the slip behavior (66% chance) led to the previous state or a closer state. Following this reasoning, we can conclude that these algorithms produced policies that reasonably led to the goal state.

Evaluation return was also visualized during the training of each algorithm. Each point in the evaluation return in Figure 4 represents the average return of 100 independent evaluation episodes on the greedy policy after every 1000 iterations in the training loop. The returns remain at zero due to need for an episode to reach the goal state before the policy/state-value function can be updated. Once the initial update occurs, the policy allows for increasing easier goal state termination until the policy is near optimized, at which the return fluctuates slightly due to stochastic behavior.

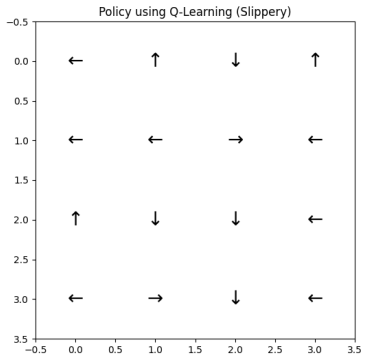
B. Non-Slippery environment

In a deterministic environment, policy for each algorithm remained mostly the same. The only differing action is in state 9, where either transition to the right or down led to the same outcome of a subsequent transition to state 14. Figure 5 shows the policy plot for all three algorithms:

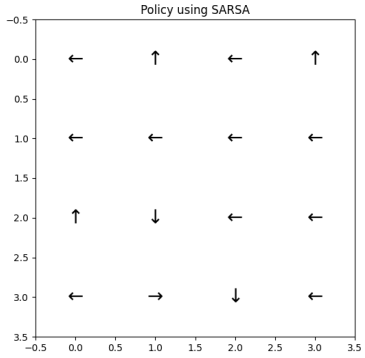
Evaluation returns exhibited steeper returns gain after the initial goal state episode, and fluctuations after the optimal policy were smaller than stochastic environment which resulted from epsilon randomness (Figure 6).



(a) Monte-Carlo Policy



(b) Q-Learning Policy



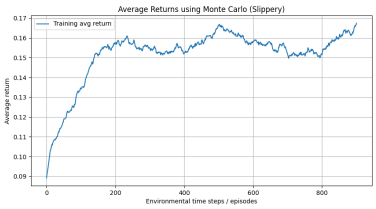
(c) SARSA Policy

Fig. 3: Policy for three algorithms

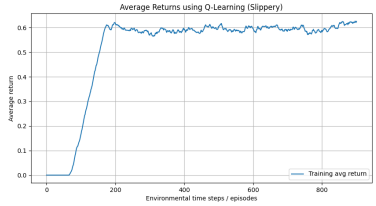
IV. Conclusion

This study compared three model-free reinforcement learning algorithms—on-policy first-visit Monte Carlo, SARSA, and Q-learning—on the Frozen Lake environment under both deterministic and stochastic dynamics. In the deterministic (non-slippery) case, all three methods converged to nearly identical optimal policies, demonstrating that both Monte Carlo and TD methods can effectively learn from interaction when transitions are predictable.

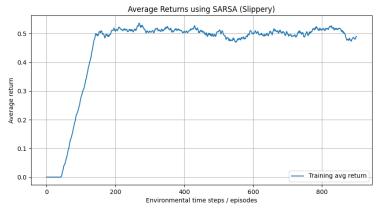
In the stochastic (slippery) variant, however, the algorithms diverged slightly in their learned strategies. Q-learning, being off-policy, tended to converge faster but exhibited higher variance, while SARSA's on-policy nature produced more conservative yet stable behavior. Monte Carlo control showed slower initial learning due to its episodic updates but achieved



(a) Monte-Carlo

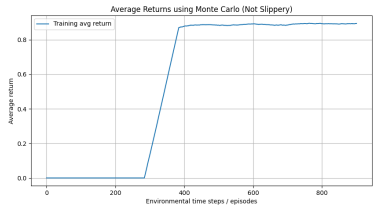


(b) Q-Learning

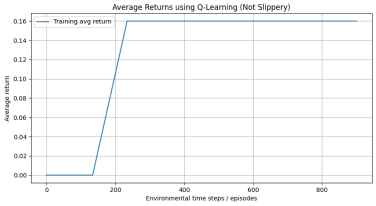


(c) SARSA

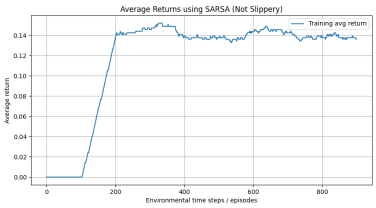
Fig. 4: Return Evaluations for three algorithms



(a) Monte-Carlo



(b) Q-Learning



(c) SARSA

Fig. 6: Return Evaluations for three algorithms

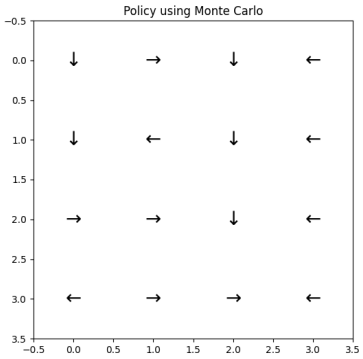


Fig. 5: Policy For Deterministic Environment

comparable final performance.

There were many notable implementation approaches that were considered throughout the projects. Firstly, evaluation return denoted by *Multi-Agent Reinforcement Learning* by Albrecht, Christianos and Schafer recommended averaging each average return over 100 training runs. Because of the smaller scale of this project and computation resources, two successful training run was done on each algorithm (one for slippery and non-slippery). Second, each training run was initialized to have 50,000 total iterations to ensure that the agent can randomly transition to the goal state and then subsequently update policy or value function enough to optimize. Because of the random nature of the first few runs, some training runs never reached the goal state, and thus policy/value function remained at the initialized values. These training runs were repeated until the policy can update. For deterministic runs, episodes had to be capped at a max time step in the case of either being stuck or endless looping, and epsilon increased to encourage exploration at the beginning.

Overall, these results highlight the trade-offs between sample efficiency and stability in model-free methods. Deterministic environments favor faster convergence, while stochastic dynamics emphasize the importance of careful balance between exploration and exploitation in achieving robust policy learning.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[2] Farama Foundation, "Frozen Lake," *Gymnasium Documentation*, [Online]. Available: https://gymnasium.farama.org/environments/toy_text/frozen_lake/. [Accessed: Sept. 16, 2025].

[3] S. V. Albrecht and F. Christianos and L. Schafer, *Reinforcement Learning: An Introduction*, [Online]. Available: https://www.marl-book.com/download/marl-book.pdf