Model-Free RL on the Frozen Lake Environment

Abhishek Pai
Department of Aerospace Engineering
University of Illinois Urbana-Champaign
avpai4@illinois.edu

***Abstract*—This report investigates the performance of three fundamental model-free reinforcement learning algorithms: On-Policy First-Visit Monte Carlo (MC) Control, SARSA, and Q-learning. These algorithms are applied to the Frozen Lake grid world problem. We analyze their ability to find an optimal policy in two distinct settings: a deterministic (non-slippery) environment and a stochastic (slippery) environment. The performance is evaluated based on learning curves, which plot the average evaluation return against time steps, and visualizations of the final learned policies and state-value functions. The results highlight the differing convergence properties and policy outcomes of on-policy and off-policy, as well as Monte Carlo and Temporal-Difference methods, in environments with varying dynamics.**

I. Introduction

Model-free RL methods are particularly powerful as they do not require a model of the environment's dynamics to learn an optimal policy. This report focuses on implementing and comparing three such algorithms: On-Policy First-Visit MC Control, SARSA, and Q-learning.

The environment for our investigation is the Frozen Lake environment from the Gymnasium library. This environment is formalized as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

- **State Space ($\mathcal{S}$):** The environment is a 4×4 grid, resulting in 16 discrete states. States represent the agent's position, with some being "frozen" (F), "start" (S), "hole" (H), or the "goal" (G).
- **Action Space ($\mathcal{A}$):** The agent can choose from four discrete actions: Left (0), Down (1), Right (2), Up (3).
- **Reward Function (R):** The agent receives a reward of +1 upon reaching the goal state (G) and a reward of 0 for all other transitions.
- **Transition Probabilities (P):** We explore two variants:
  1) **Non-Slippery:** The environment is deterministic. The chosen action is always executed, moving the agent to the intended next state.
  2) **Slippery:** The environment is stochastic. When the agent chooses an action, there is a 1/3 probability of moving in the intended direction and a 1/3 probability for each of the two perpendicular directions.
- **Discount Factor (γ):** A discount factor of $\gamma = 0.95$ is used to value future rewards.

The objective is to find an optimal policy, π_*, which maximizes the expected discounted return for the agent starting from any state.

II. Methods

The following algorithms were implemented to solve the Frozen Lake MDP. They learn an action-value function, $Q(s, a)$, from which a policy can be derived. The policy used for exploration is an ϵ-greedy policy, which selects the greedy action with probability $1 - \epsilon$ and a random action with probability ϵ.

A. On-Policy First-Visit MC Control

Monte Carlo methods learn from complete episodes of experience. In the first-visit variant, the state-action pair is updated only the first time it is visited in an episode. The Q-values are updated by averaging the returns following the first visit to each state-action pair. The pseudocode is presented in Algorithm 1.

Algorithm 1 On-policy first-visit MC control (for ϵ-soft policies)

```
1:  Algorithm parameter: small $\epsilon > 0$
2:  Initialize:
3:      $\pi \leftarrow$ an arbitrary $\epsilon$-soft policy
4:      $Q(s,a) \in R$ (arbitrarily), for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
5:      $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
6:
7:  loop forever (for each episode)
8:      Generate an episode following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
9:      $G \leftarrow 0$
10:     for $t = T-1, T-2, \ldots, 0$ do
11:         $G \leftarrow \gamma G + R_{t+1}$
12:         if the pair $S_t, A_t$ does not appear in $S_0, A_0, S_1, A_1, \ldots, S_{t-1}, A_{t-1}$ then
13:             Append $G$ to $Returns(S_t, A_t)$
14:             $Q(S_t, A_t) \leftarrow$ average($Returns(S_t, A_t)$)
15:             $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
16:             for all $a \in \mathcal{A}(S_t)$ do
17:                 $\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$
```

B. SARSA: On-Policy TD Control

SARSA is a Temporal-Difference (TD) learning algorithm that updates the Q-value for a state-action pair based on the action taken in the next state under the current policy. The update rule uses the tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, giving the algorithm its name. The pseudocode is shown in Algorithm 2.

Algorithm 2 SARSA (On-Policy TD Control)

```
1: Initialize $Q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, and $Q(\text{terminal-state}, \cdot) = 0$
2:
3: loop for each episode
4:     Initialize $S$
5:     Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
6:     loop for each step of episode
7:         Take action $A$, observe $R, S'$
8:         Choose $A'$ from $S'$ using policy derived from $Q$
9:         $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
10:        $S \leftarrow S'; A \leftarrow A'$
11:        if $S$ is terminal then
12:            break
```

C. Q-Learning: Off-Policy TD Control

Q-learning is an off-policy TD control algorithm. It directly learns the optimal action-value function, q_*, independent of the policy being followed. The update rule involves taking the maximum Q-value over all possible actions in the next state, making it a greedy approach in its value updates. The pseudocode is provided in Algorithm 3.

Algorithm 3 Q-Learning (Off-Policy TD Control)

```
1: Initialize $Q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, and $Q(\text{terminal-state}, \cdot) = 0$
2:
3: loop for each episode
4:     Initialize $S$
5:     loop for each step of episode
6:         Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
7:         Take action $A$, observe $R, S'$
8:         $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_{a'} Q(S',a') - Q(S,A)]$
9:         $S \leftarrow S'$
10:        if $S$ is terminal then
11:            break
```

D. Hyperparameters

The policy for all three algorithms was an ϵ-greedy policy where ϵ decayed multiplicatively after each episode from a starting value of 1.0 to a minimum value. The specific hyperparameters were tuned for each environment variant:

- **Non-Slippery:** 20,000 training episodes; ϵ-decay rate = 0.995; min ϵ = 0.01; learning rate α = 0.5 for SARSA and Q-learning.
- **Slippery:** 200,000 training episodes; ϵ-decay rate = 0.999; min ϵ = 0.05; learning rate α = 0.2 for SARSA and Q-learning.

For the On-Policy MC algorithm, the action-value function $Q(s,a)$ was updated using incremental averaging based on first-visit returns. Performance was measured by periodically pausing training to evaluate the greedy policy (derived from the current Q-function) for 300 episodes and averaging the resulting discounted returns. This evaluation occurred every 1000 episodes for the non-slippery case and every 2000 episodes for the slippery case.

III. Results

A. Non-Slippery Environment

The learning curves in Figure 1 show that both On-Policy MC and Q-learning rapidly find an optimal policy and achieve a high average return. Q-learning converges almost instantaneously. In contrast, SARSA completely fails to learn a successful policy, maintaining an average return of zero. This is a example of the risks of on-policy learning in a deterministic environment. An early exploratory action that leads into a hole results in a large negative TD error, which teaches the agent that the preceding state-action pair is poor. Because SARSA's updates are based on the action its current policy actually takes next (which is also exploratory), it cannot learn the value of an optimal path if its exploration policy leads it astray. This discourages any path near a hole, preventing it from discovering the optimal path.

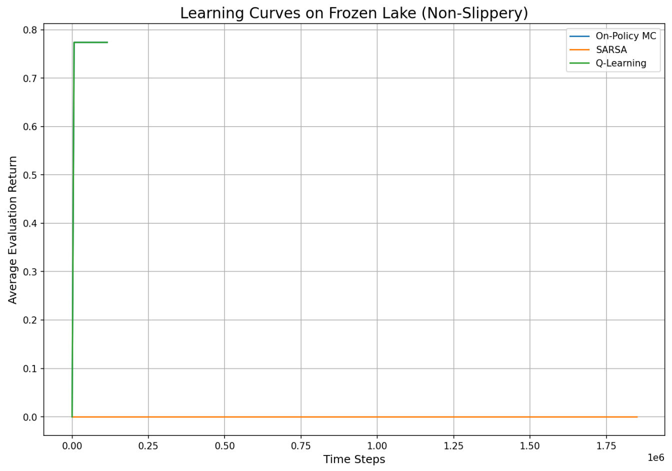

Fig. 1. Learning curves for all three algorithms in the non-slippery Frozen Lake environment.

The learning curves in Figure 1 show that both On-Policy MC and Q-learning rapidly find an optimal policy and achieve a high average return. Q-learning converges almost instantaneously. In contrast, SARSA completely fails to learn a successful policy, maintaining an average return of zero. This is because SARSA's on-policy nature can lead it to learn a "safe" but suboptimal policy. An exploratory move towards a hole results in a large negative update if the next action chosen (also by the exploratory policy) leads away from the goal, discouraging that path.

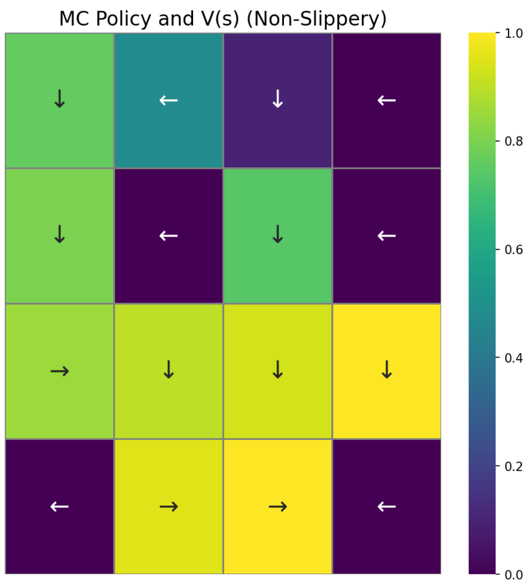

Fig. 2. MC final policy and state values (Non-Slippery).

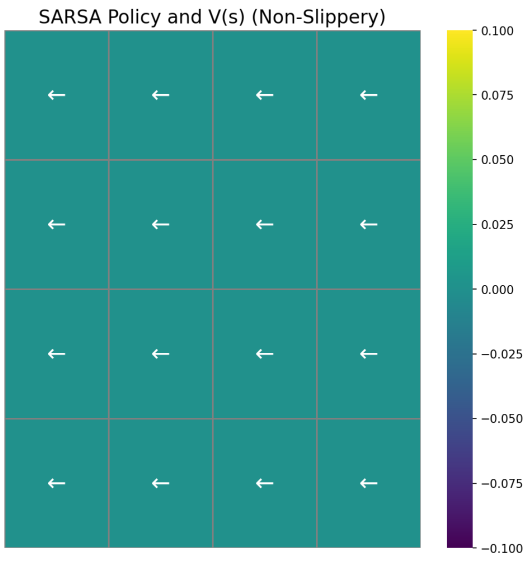

Fig. 3. SARSA final policy and state values (Non-Slippery).

The final policies are visualized in Figures 2-4. Both MC and Q-learning (Figures 2 and 4) discovered an optimal path to the goal. The SARSA policy (Figure 3) is highly suboptimal, directing the agent to move left in most states, which does not lead towards the goal. The state values are correspondingly close to zero, reflecting the policy's failure.

B. Slippery Environment

In the stochastic setting, the environment introduces randomness into the agent's movements.

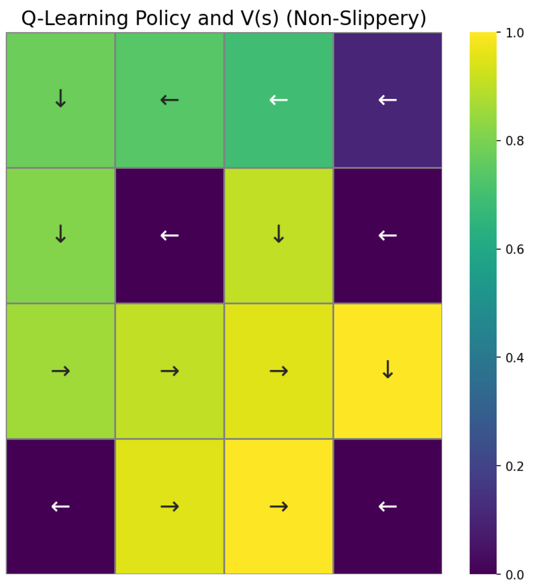

Fig. 4. Q-Learning final policy and state values (Non-Slippery).

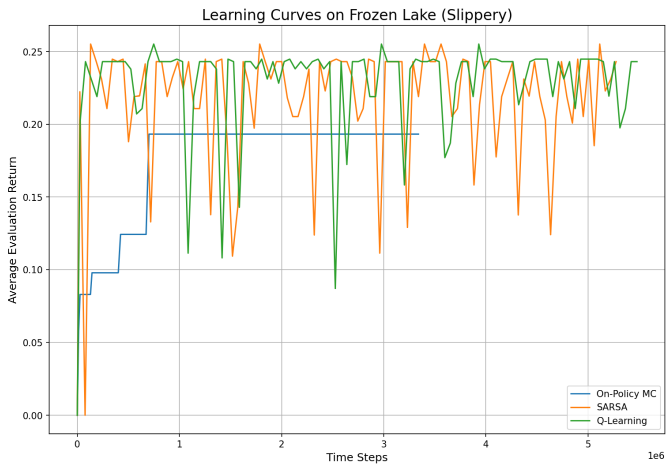

Fig. 5. Learning curves for all three algorithms in the slippery Frozen Lake environment.

The learning curves in Figure 5 demonstrate that the stochastic nature of the environment makes learning significantly more challenging. All three algorithms exhibit much greater variance in their performance. Q-learning and SARSA appear to learn faster initially but show high instability throughout training. On-Policy MC learns more slowly and plateaus for long periods, but its performance is more stable once a reasonably good policy is found. All three algorithms manage to find policies that achieve a positive average return, a marked improvement for SARSA compared to the non-slippery case.

Figures 6-8 show the final learned policies in the slippery environment. The optimal policies are now more conservative, trying to avoid paths that are adjacent to holes due to the risk of slipping in. The policies learned by SARSA and Q-learning are very similar, which is expected as SARSA's

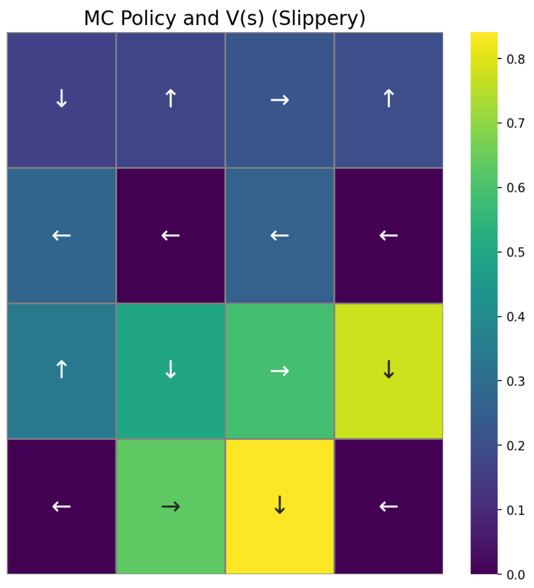

Fig. 6. MC final policy and state values (Slippery).

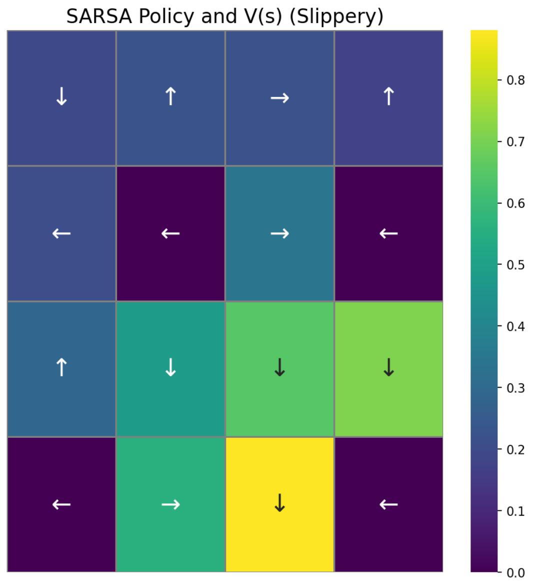

Fig. 7. SARSA final policy and state values (Slippery).

behavior approaches Q-learning's when the policy becomes more deterministic (i.e., as ϵ decays). The MC policy is also qualitatively similar, finding a safe path to the goal. The state-value functions correctly assign higher values to states closer to the goal along these safe paths.

IV. Conclusion

This report successfully implemented and compared On-Policy MC Control, SARSA, and Q-learning on the Frozen Lake problem. In the deterministic (non-slippery) environment, the off-policy nature of Q-learning allowed it to find the optimal policy with remarkable efficiency, while the on-policy SARSA algorithm failed completely due to its sensitivity to

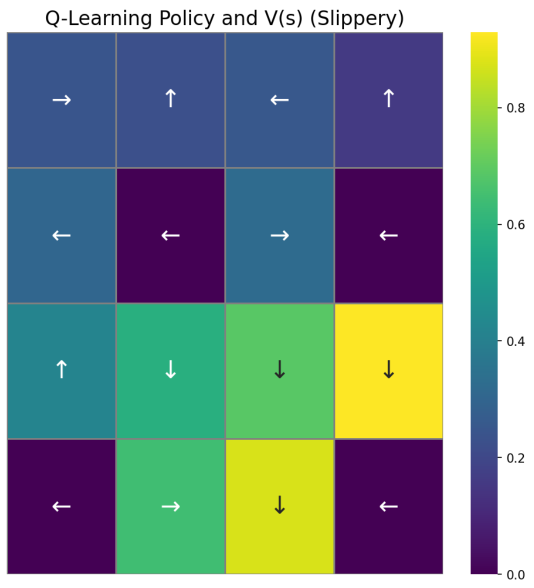

Fig. 8. Q-Learning final policy and state values (Slippery).

exploratory actions. In the stochastic (slippery) environment, all three algorithms found successful, albeit more conservative, policies. The TD methods (SARSA and Q-learning) learned faster but were more unstable, whereas the MC method learned more slowly but demonstrated greater stability. This comparison highlights the fundamental trade-offs between on-policy and off-policy learning, and between Monte Carlo and Temporal-Difference methods, when tackling reinforcement learning problems.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.