AIL 7022: Reinforcement Learning
Assignment 2

Ankur Kumar (2025AIB2557)
Indian Institute of Technology Delhi

1. Temporal Difference

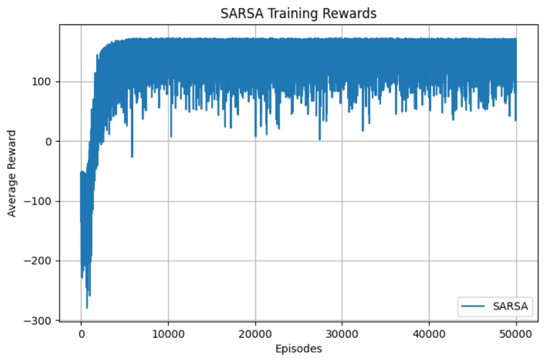
1.1. Cliff Environment

1.1.1. Code Implementation

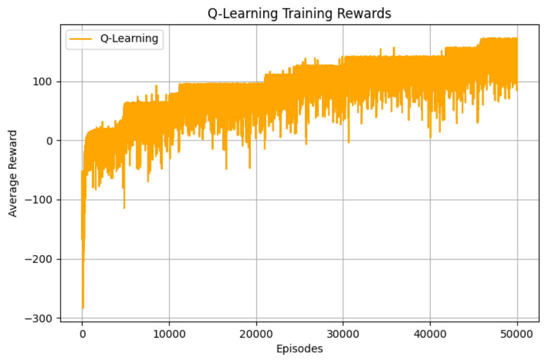
The implementation details are available on GitHub. The repository includes implementations of SARSA, Q-Learning, and Expected SARSA for a custom cliff environment.

1.1.2. Plotting average learning curve

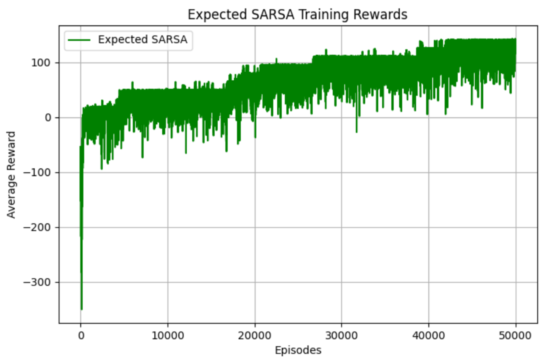
Figure 1 shows the learning curves for SARSA (Fig. 1a), Q-Learning (Fig. 1b), and Expected SARSA (Fig. 1c).



(a) SARSA



(b) Q Learning



(c) Expected SARSA

Figure 1: Average reward for different learning algorithms

In all the plots, the reward converges towards the optimal reward, which is obtained when the agent takes the risky goal. However, we observe significant fluctuations in the rewards even after convergence. This occurs because an ϵ-greedy policy is used to learn the Q-values. As a result, even when the agent has learned near-optimal Q-values, the random exploratory actions can sometimes yield very low rewards. Since the environment is stochastic, this effect is further amplified, leading to noticeable fluctuations in the average reward.

1.1.3. Plot Average Goal visits

Figure 2 shows a grouped bar graph depicting the average number of visits for each of the three algorithms. Table 1 summarizes the average number of visits for all three algorithms.

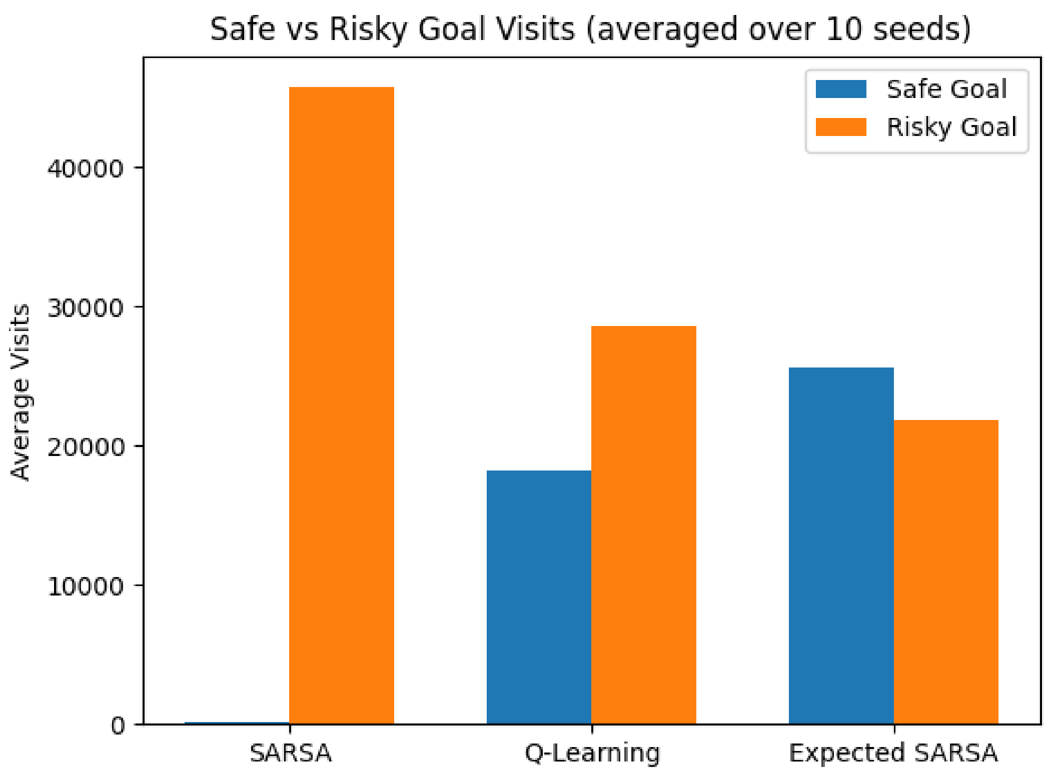


Figure 2: Grouped bar graph showing the average number of visits for each algorithm.

Algorithm	**Safe Goal Visits (avg.)**	**Risky Goal Visits (avg.)**
SARSA	211.0	45710.7
Q-Learning	18254.3	28618.2
Expected SARSA	25662.1	21820.2

Table 1: Average number of visits to safe and risky goals for each algorithm.

Based on the data, we observe that SARSA visits the safe goal the least during training, while Q-Learning and Expected SARSA visit it more frequently. A possible reason is that, in this environment with two goals — one offering a moderate reward and the other a high reward — Q-Learning and Expected SARSA initially get "stuck" in the local optimum corresponding to the safe goal and take some time to explore the risky goal. This behavior is also evident from the learning curves. However, this is not an inherent property of these algorithms; with different hyperparameter choices, SARSA can get stuck in a local optimum, while Q-Learning or Expected SARSA may not, and vice versa. Overall, all three algorithms are highly sensitive to hyperparameters.

1.1.4. Results

Table 2 summarizes the mean and standard deviation of visits for all three algorithms.

Algorithm	**Average Visits (mean $\pm$ std)**
Q-Learning	173.18 ± 1.69
SARSA	169.42 ± 22.27
Expected SARSA	173.39 ± 1.05

Table 2: Average number of visits (mean $\pm$ standard deviation) for each algorithm.

1.1.5. GIF plotting

The GIFs for the best policies can be found on GitHub. Each GIF visualizes the agent's path from start to finish in the environment and is saved in the `gifs` folder as follows:

- SARSA.gif
- Q Learning.gif
- Expected SARSA.gif

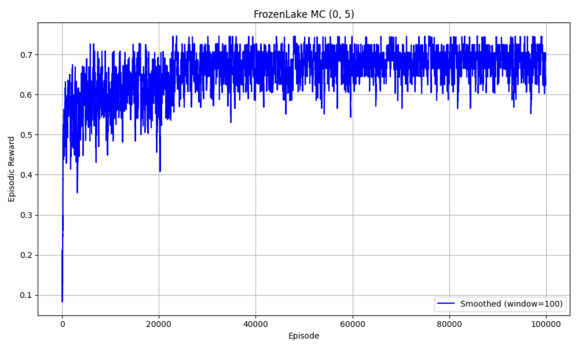
1.2. Frozen Lake Environment

1.2.1. Code Implemenataion

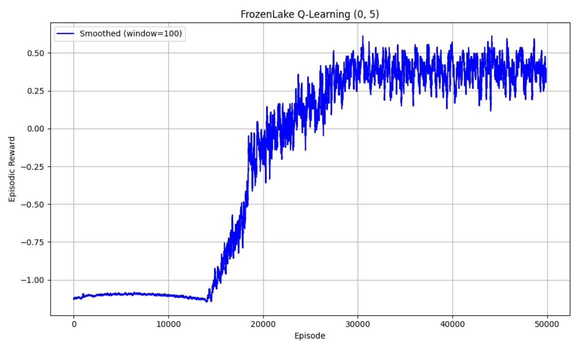
The implementation details are available on GitHub. The repository includes implementations of Q-Learning and Monte Carlo for a custom frozen lake environment.

1.2.2. Training

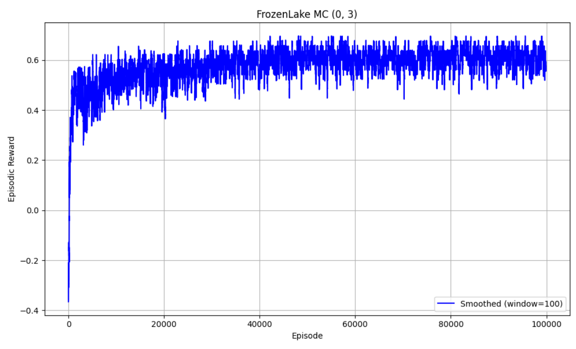
Figure 3a shows the reward curve for Monte Carlo (MC) starting from state (0,5), while Figure 3b shows the reward curve for Q-Learning (QL) from the same start state. Similarly, Figure 3c shows the MC reward curve starting from state (0,3), and Figure 3d shows the QL reward curve from that start state. These images illustrate how the reward evolves over training for each algorithm and how the start state affects convergence.



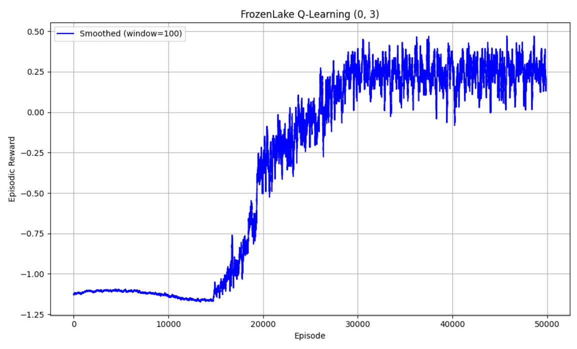
(a) Monte Carlo, start state (0,5)



(b) Q-Learning, start state (0,5)



(c) Monte Carlo, start state (0,3)



(d) Q-Learning, start state (0,3)

Figure 3: Reward curves for Monte Carlo and Q-Learning with two different start states.

1.2.3. Evaluation

Table 3 shows that the returns for both algorithms are identical for each start state, and the variance is effectively zero due to the deterministic nature of the environment.

1.2.4. Generated GIFs

The GIFs for both algorithms from different start states can be found on GitHub. Each GIF visualizes the agent's path from start to finish in the environment:

Algorithm (Start State)	**Mean Return**	**Std. Dev.**
Q-Learning (0,3)	0.7000	2.22×10^{-16}
Monte Carlo On-Policy (0,3)	0.7000	2.22×10^{-16}
Q-Learning (0,5)	0.7500	0.0
Monte Carlo On-Policy (0,5)	0.7500	0.0

Table 3: Mean and standard deviation of returns for Q-Learning and Monte Carlo (On-Policy) from different start states. Since the environment is deterministic, the variance is effectively zero.

- frozenlake_mc_(0,5).gif
- frozenlake_qlearning_(0,5).gif
- frozenlake_mc_(0,3).gif
- frozenlake_qlearning_(0,3).gif

These GIFs illustrate the agent's trajectories for Monte Carlo (MC) and Q-Learning (QL) from the specified start states.

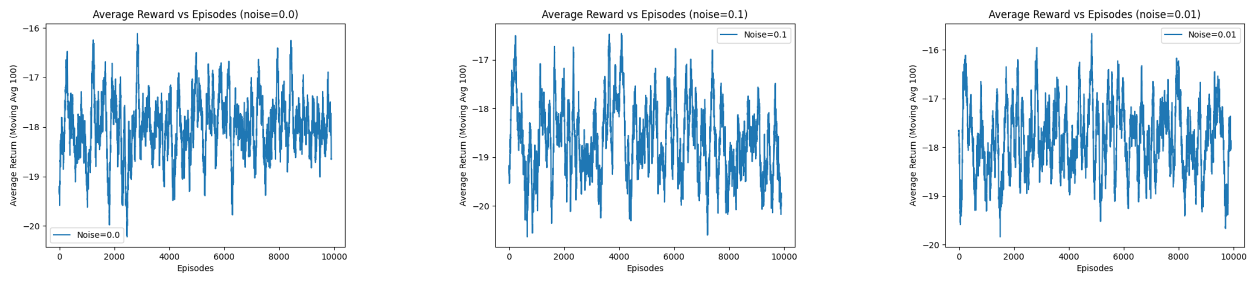
2. Importance Sampling

2.1. Code Implementation

The implementation details are available on GitHub. The repository includes implementations of Temporal Difference with Importance Sampling for gym gridworld environment.

2.2. Training and Learning Analysis

We train the agents from an exploratory behavior policy. Due to this exploration, the rewards observed during training do not necessarily converge, but the policy gradually improves in selecting actions that lead to higher rewards. Figures 4 and 5 illustrate the reward curves for different noise levels for the Monte Carlo and Temporal-Difference Importance Sampling (TD-IS) algorithms, respectively. Each subplot corresponds to a specific noise level in the behavior policy, highlighting how stochasticity affects learning.



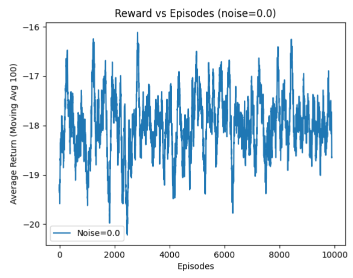
(a) Noise = 0.0 (b) Noise = 0.1 (c) Noise = 0.01

Figure 4: Monte Carlo Importance Sampling: Training reward curves for different noise levels.

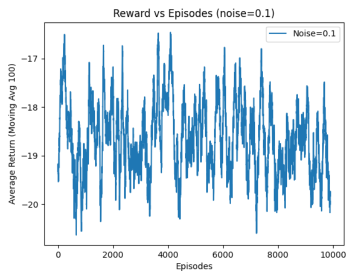
The optimal policy for each algorithm is selected based on the success rate observed across multiple random seeds. Specifically, for each seed, the policy learned during training is evaluated, and the success rate is computed as the fraction of episodes in which the obtained reward exceeds 0.5. The policy corresponding to the highest success rate among all seeds is then chosen as the *best policy* for that algorithm. This approach ensures that the selected policy reliably achieves positive outcomes, even under the stochasticity introduced by the exploratory behavior policy during training.

2.3. Evaluation

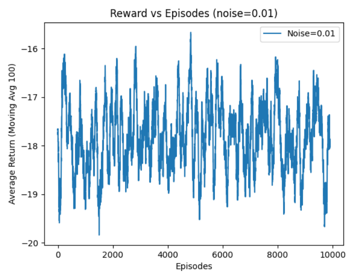
For each algorithm, the *best policy* was selected based on the highest success rate observed across multiple random seeds. After determining the best policy, it was evaluated over 100 independent episodes to assess its performance. The mean reward and standard deviation reported in Table 4 correspond to the average and variability of rewards obtained during these 100 evaluation runs. This procedure ensures that the metrics reflect the policy's reliability in achieving positive outcomes, rather than just performance in a single training episode.



(a) Noise = 0.0



(b) Noise = 0.1



(c) Noise = 0.01

Figure 5: Temporal-Difference Importance Sampling (TD-IS): Training reward curves for different noise levels.

Monte Carlo Importance Sampling		
Noise	Mean Reward	Std Dev
0.0	0.700	1.538
0.1	1.781	0.994
0.01	1.812	0.887

TD Importance Sampling		
Noise	Mean Reward	Std Dev
0.0	0.726	1.593
0.1	1.574	1.647
0.01	1.383	1.532

Table 4: Comparison of mean rewards and standard deviations for Monte Carlo and TD Importance Sampling.

2.4. Policy Demonstration

To visualize the learned policies, we generated GIFs showing the agent's behavior for the best policy of each algorithm across different noise levels. These demonstrations illustrate how the agent navigates the environment under the influence of stochastic exploratory behavior and how the learned policy effectively reaches the goal.

Monte Carlo Importance Sampling

- Noise = 0.0: View GIF
- Noise = 0.1: View GIF
- Noise = 0.01: View GIF

TD Importance Sampling

- Noise = 0.0: View GIF
- Noise = 0.1: View GIF
- Noise = 0.01: View GIF