Assignment-2

Kunal Kumar Sahoo (2025AIZ8459)
AIL7022

October 6, 2025

1 Temporal Difference

1.1 Cliff Environment

1. The algorithms `SARSA`, `Q Learning`, and `Expected SARSA` have been trained on the Cliff environment across 10 different seed values. The update rules of these algorithms can be expressed as:

   - `SARSA`:

   $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right]$$

   - `Q-Learning`:

   $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)\right]$$

   - `Expected SARSA`:

   $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_{a'} \pi(S_{t+1}, a') Q(S_{t+1}, a') - Q(S_t, A_t)\right]$$

   Policy: `Softmax` or `Boltzmann` policy defined as follows:

   $$\pi(a \mid s) = \frac{\exp\left(Q(s,a)/\tau\right)}{\sum_{a'} \exp\left(Q(s,a')/\tau\right)},\ \tau > 0$$

   § This has been adopted for the problem because of the following reasons:

   - Softmax provides a more nuanced exploration strategy by assigning higher probabilities to actions with higher Q-values, even during exploratory choices. This means exploration is "graded" based on the relative quality of actions.

- The temperature τ allows for continuous tuning of exploration: high τ makes the policy nearly uniform (maximal entropy), while low τ makes it nearly deterministic (maximal exploitation). This enables annealing schedules that guarantee convergence to the optimal policy.
- Softmax adapts exploration probabilities dynamically based on Q-value differences, which is particularly beneficial in noisy or non-stationary environments. It avoids over-exploring actions that are confidently inferior.

Key training settings:

Table 1: Training Configuration for the Cliff Environment

Hyperparameter	**Value**
# seeds	10
# episodes / seed	10000
Max. # steps	1000
Discount factor	0.99
Start temperature	5.0
Temperature decay rate	0.999
Step size	0.9

Key observations:

- These algorithms are very sensitive to hyperparameter tuning and hence hyperparameter optimization techniques like `GridSearchCV` or `Bayesian Tuning` can be easily employed.
- Hyperparameters influence the number of safe and risky visits amongst the algorithms.
- Unlike conventional settings, the agents learn better when step size and temperature are high.
- Within 10000 episodes, all the three algorithms almost converge to the optimal policy.
- The observed decrease in risky goal visits stems directely from the algorithms' update strategies. Q-Learning being off-policy, uses an optimistic max operator that learns the value of the ideal path, ignoring exploration risks and thus aggressively pursuing the high-reward goal. In contrast, on-policy SARSA learns the value of its actual, exploratory policy by using the specific next action taken, making it more conservative as it accounts for the real possibility of penalties. Expected SARSA is the most risk-averse because it enhances the on-policy approach by using the expected value over all future actions. This averaging reduces update variance and produces the most stable, cautious policy.

2. Refer to Figure 1 for the average episodic rewards across 10 seeds for the algorithms: `SARSA`, `Q-Learning`, and `Expected SARSA`:

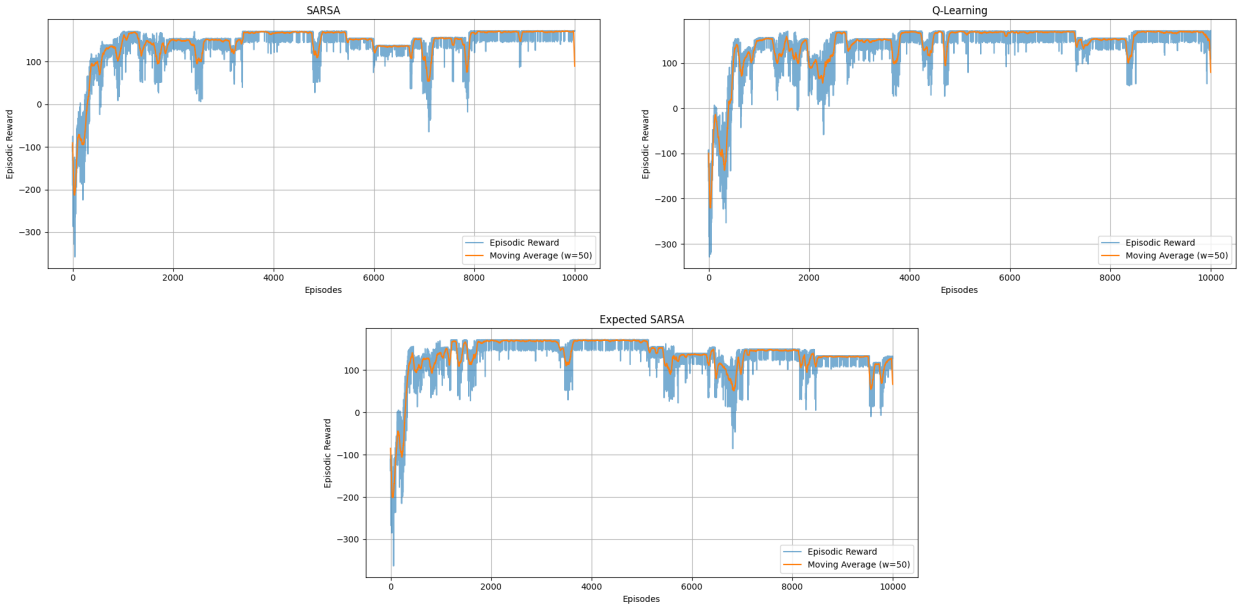

Figure 1: Average episodic rewards for `SARSA`, `Q-Learning`, and `Expected SARSA`

3. Refer to the Figure 2 for all the algorithms in single plot that shows the average number of times the agents reached the "Safe Goal" versus the "Risky Goal" during training, averaged across 10 seeds:

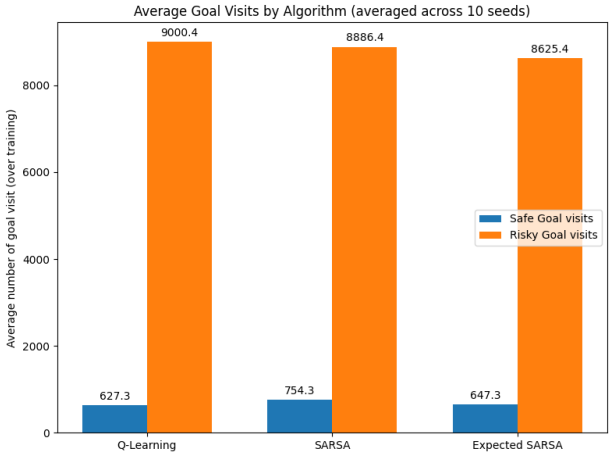

Figure 2: Average goal visits for the algorithms `SARSA`, `Q-Learning`, and `Expected SARSA`.

4. § Definition of the **Best Policy**: A best policy is defined as the Q-table that corresponds to the policy yielding the highest average cumulative reward over the last k episodes (where k is specified by `last_k` in the environment settings) across multiple random seeds. For each seed, the mean cumulative reward over the last k episodes is computed, then the best Q-table is then selected as the Q-table from the seed that achieves the highest value in `seed_last_k_means`.

$$Q^* = Q_{s^*}, \text{ where } s^* = \arg \max_{s \in \{1,2,\ldots \text{num\_seeds}\}} \left\{ \frac{1}{k} \sum_{e=\text{num\_episodes}-k+1}^{\text{num\_episodes}} r_{s,e} \right\}$$

5. The final performance of the agents trained using `SARSA`, `Q-Learning`, and `Expected SARSA` algorithms are reported and saved in `cliff_evaluation_results.json` whose content is presented in a tabular manner in Table 2:

Table 2: Performance of `SARSA`, `Q-Learning`, and `Expected SARSA` algorithms

Algorithm	**Mean Reward**	**Standard Deviation**
`SARSA`	170.96	22.56
`Q-Learning`	173.50	1.12
`Expected SARSA`	166.51	35.09

6. Refer to the following URLs to see the GIFs of performance of different algorithms on the environment:
   (a) `SARSA`: `https://drive.google.com/file/d/1fy0TJNdEgewMAFdCNAvDSf89ms-mr1l5/view?usp=sharing`
   (b) `Q-Learning`: `https://drive.google.com/file/d/1twQl-LQKtwRs8QHuvk11dtEhh9Fk9QnL/view?usp=sharing`
   (c) `Expected SARSA`: `https://drive.google.com/file/d/1rasX6KKtNkltRN-Dp-lIHtgqaFSoRF61/view?usp=sharing`

1.2 Frozen Lake Variant

1. `Monte Carlo On Policy` and `Q-Learning` control algorithms have been implemented. The update rules of these algorithms are as follows:
   (a) `Monte Carlo On-Policy Control`:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} - Q(S_t, A_t) \right]$$

   (b) `Q-Learning Off-Policy Control`:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

   Key training settings:

Table 3: Training Configuration for Frozen Lake Variant Environment

Hyperparameter	**Value**
Seed Value	12345
# episodes	1000000
Max. # steps	100
Discount factor	1.0
Start temperature	1.0
Temperature decay rate	0.999995
Step size	0.5

For this environment as well, softmax policies have been adopted for the reasons mentioned in § 1.

Key observations:

- These algorithms are very sensitive to hyperparameter tuning and hence hyperparameter optimization techniques like `GridSearchCV` or `Bayesian Tuning` can be easily employed.
- `Monte-Carlo` algorithm converges faster than `Q-Learning` algorithm.
- The faster convergence of the Monte-Carlo algorithm in the Frozen Lake Variant is due to the environment's deterministic transitions and sparse rewards. Monte-Carlo's update mechanism, which uses the complete return after an episode, allows for highly efficient credit assignment by propagating the final reward to an entire successful trajectory in a single update. Conversely, Q-Learning's incremental, one-step updates are substantially slower at backpropagating this distant reward signal. Furthermore, the deterministic nature of the environment negates Monte-Carlo's primary weakness of high variance while diminishing Q-Learning's main advantage of lower variance, making Monte-Carlo's unbiased, full-return updates more effective for this specific task.

2. Refer to the Figure 3 and Figure 4 to visualize the performance of the `Monte Carlo Control` and `Q-Learning Control` algorithms respectively for different starting position of the agent:

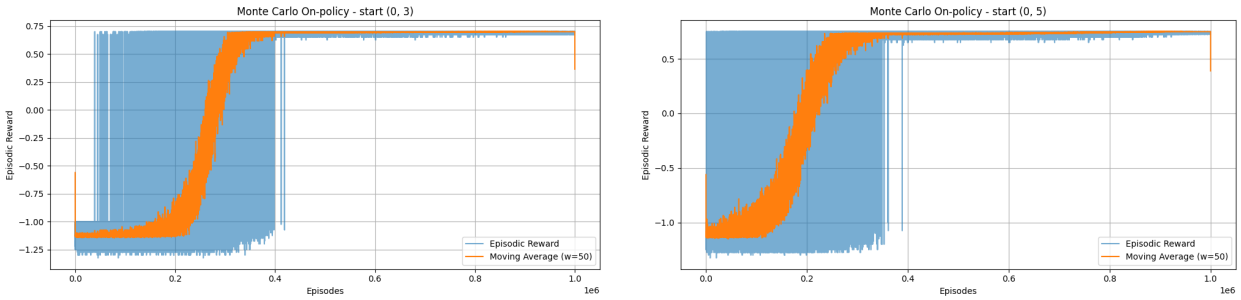

Figure 3: Performance `Monte-Carlo On-Policy Control` Algorithm for Starting Positions `(0, 3)` and `(0, 5)`.

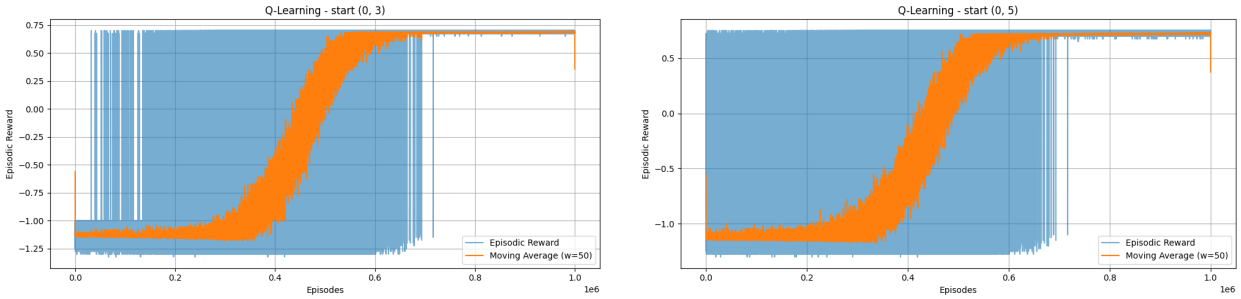

Figure 4: Performance `Q-Learning Control` Algorithm for Starting Positions `(0, 3)` and `(0, 5)`.

3. Refer to Table 4 to compare the mean reward and their standard deviation for `Monte Carlo` and `Q-Learning` algorithms

Table 4: Comparison of Mean and Standard Deviation Rewards for `Monte-Carlo` and `Q-Learning`

State	`Monte Carlo (MC)`		`Q-Learning`	
	Mean	Std Dev	Mean	Std Dev
(0, 3)	0.7000	2.22×10^{-16}	0.7000	2.22×10^{-16}
(0, 5)	0.7500	0.00	0.7500	0.00

4. Refer to the following URLs to see the GIFs of performance of different algorithms on the environment:
   (a) `Monte Carlo`:
      i. $(0, 3)$: `https://drive.google.com/file/d/1Hn9wM1Tc3jAB6j81e8Ng9lj2204DvCzo/view?usp=sharing`
      ii. $(0, 5)$: `https://drive.google.com/file/d/1iO9qgj0QOWkHnkWRv1pkO0Bh2HUNIOLH/view?usp=sharing`
   (b) `Q-Learning`:
      i. $(0, 3)$: `https://drive.google.com/file/d/1BJsG2ehePP24cnBa96I40L2zMmW_zj8w/view?usp=sharing`
      ii. $(0, 5)$: `https://drive.google.com/file/d/1qNN-9k6kbF9fmd31ZHSw3WvIGH6Xiix3/view?usp=sharing`

2 Importance Sampling

1. The algorithms `Off-Policy TD[0] Control with Importance Sampling` and `Off-Policy Monte-Carlo Control with Weighted Importance Sampling` are implemented. The update rules for both the algorithms are as follows:

- `Off-Policy TD[0] Control with Importance Sampling`

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left\{ \frac{\pi(A_t \mid S_t)}{\mu(A_t \mid S_t)} \left[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1}, a) \right] - Q(S_t, A_t) \right\}$$

- `Off-Policy Monte Carlo Control with Weighted Importance Sampling`

$$W(S_t, A_t) \leftarrow W(S_t, A_t) + \frac{\pi_{\text{greedy}}(A_t \mid S_t)}{\pi(A_t \mid S_t)}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{W(S_t, A_t)} \left[\frac{\pi_{\text{greedy}}(A_t \mid S_t)}{\pi(A_t \mid S_t)} \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} - Q(S_t, A_t) \right]$$

Key observations:

- This environment setting is not very sensitive to hyperparameter tuning.
- The learning behavior from a behavior policy is not as good as native exploration-exploitation. It might be due to sampling inefficiency from the behavior policy.
- `TD[0] Control` seems to learn more optimal policies than `Monte Carlo Control`. This is because Monte Carlo methods have higher variance than TD methods.
- The noted inefficiency of learning from a behavior policy is fundamental to the off-policy approach, as experience is often gathered from states irrelevant to the optimal target policy, leading to discounted or discarded updates. Finally, the TD algorithm's superior performance stems from its low-variance, single-step updates, which are more stable in this stochastic setting compared to the high-variance, full-return updates of the Monte Carlo method that struggle to converge reliably.

2. Refer to the § 4 for the definition of the best Q-table from the training metrics. Key training settings:

Table 5: Training Configuration for the Grid Environment

Hyperparameter	`TD[0] with Importance Sampling`	`MC with Weighted Importance Sampling`
# seeds	10	10
# episodes / seed	50000	50000
Discount factor	0.99	0.99
Start temperature	5.0	5.0
Temperature decay rate	0.999	0.999
Step size	0.8	N/A

3. Refer to Table 6 to check the evaluation results of the best Q-tables of both the algorithms across 100 different episodes.

Table 6: Comparison of Mean and Standard Deviation for `Monte Carlo` and `Temporal Difference 0` Methods

Noise	`Monte Carlo (MC)`		`Temporal Difference (TD)`	
	Mean	Std Dev	Mean	Std Dev
0.0	1.009	1.2945	1.831	0.8704
0.01	0.916	1.4576	1.743	1.0209
0.1	0.528	1.4890	1.850	0.8529

4. Refer to Figure 5 and Figure 6 to visualize the performance of `Monte-Carlo Control with Weighted Importance Sampling` and `TD[0] Control with Importance Sampling` for noise level $\{0.0, 0.01, 0.1\}$

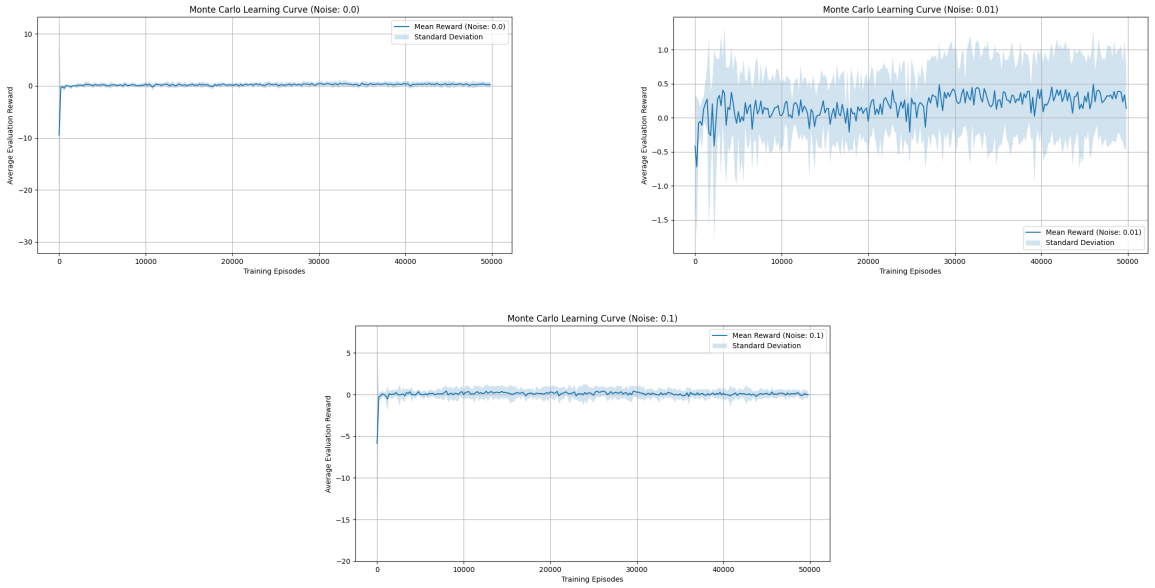

Figure 5: Average episodic rewards for `Off-Policy Monte-Carlo Control with Weighted Importance Sampling` for noise levels 0.0, 0.01, and 0.1 respectively.

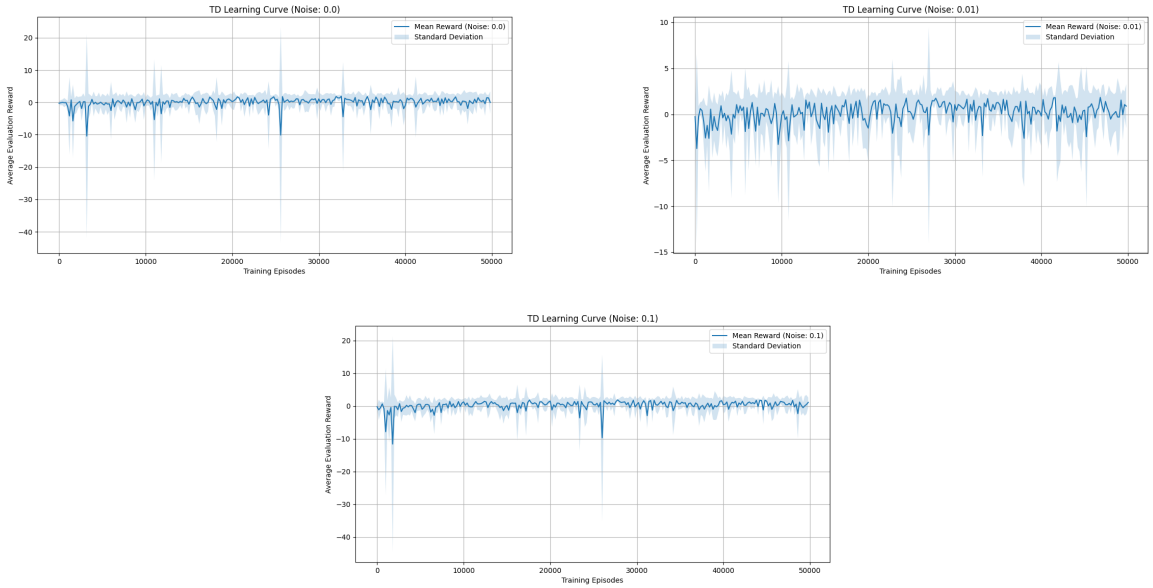

Figure 6: Average episodic rewards for `Off-Policy TD[0] Control with Importance Sampling` for noise levels 0.0, 0.01, and 0.1 respectively.

5. Refer to the following URLs to see the GIFs of performance of different algorithms on the environment:

   (a) `Off-Policy Monte Carlo Control with Weighted Importance Sampling`:

      i. Noise = 0.0: `https://drive.google.com/file/d/1Wi2Ye0LUVriLSUOmqiX-3mjDEw29d7A5/view?usp=sharing`

      ii. Noise = 0.01: `https://drive.google.com/file/d/1FYAKvjNkxDYccCTEFq48k1KDn7kkeVCT/view?usp=sharing`

      iii. Noise = 0.1: `https://drive.google.com/file/d/1DZj-BAVCFph6wLah52UdXBfENbkBM0lN/view?usp=sharing`

   (b) `Off-Policy TD[0] Control with Importance Sampling`:

      i. Noise = 0.0: `https://drive.google.com/file/d/1Yk3NKleL556L6zj7KCj4VttnF0ElmPy4/view?usp=sharing`

      ii. Noise = 0.01: `https://drive.google.com/file/d/1XwspjcsiXMWZwMUI6uteIXGJCiv9Mw5n/view?usp=sharing`

      iii. Noise = 0.1: `https://drive.google.com/file/d/1WjgWnDT4Yuf3Jbb7ctEY2vgmpOghHbb4/view?usp=sharing`