COMP 138 RL: Programming Assignment 3

Jane Slagle

November 25, 2024

1 Introduction

1.1 Problem Background

1.1.1 Off-Policy Learning Methods

This programming assignment focuses on two distinct off-policy learning techniques: a multi-step off-policy method and a per-decision method leveraging control variates.

Off-policy learning involves estimating the value function for a target policy, represented as π, while following a different policy known as the behavior policy, represented as b. Typically, π is a greedy policy based on the current action-value function estimate, while b is an exploratory policy. To account for the differences between π and b, their relative probabilities of selecting the observed actions are used. We will explore how this is handled differently in the following two approaches:

Multi-Step Off-Policy Methods n-step off-policy learning leverages experience from exploratory policies while ensuring the updates align with the target policy. It achieves this by using importance sampling ratios to weight returns over n steps, which makes it a powerful and flexible method for reinforcement learning.

Per-decision Methods with Control Variates improves upon the traditional multi-step off-policy methods described above by incorporating a more sophisticated approach to importance sampling. Where a multi-step off-policy method directly weights the n-step return by the importance sampling ratio, this method introduces a *control variate*, which is an additional term added to the return to reduce variance in the updates. This control variate term helps stabilize learning, meaning that this method is exploring in a stable manner, making it more efficient than conventional off-policy methods.

1.1.2 The Frozen Lake Game

Frozen Lake is a game commonly played in reinforcement learning scenarios. It involves crossing a frozen lake from a start to a marked goal position by walking over the frozen lake without falling into any holes that may have formed on the lake's surface.

The lake is represented as a grid with the following specifications:

Action Space the player in the game can take one of four possible actions at each step while traversing across the lake: moving left, down, right, or up.

Observation Space the observation is an integer value representing the player's current position on the lake.

Starting State each episode starts with the player in state [0], corresponding with location $([0, 0])$ on the grid, which is always the very upper leftmost corner of the lake grid.

Rewards The reward for reaching the goal state is $+1$ while the reward for falling into a hole or ending an episode in a frozen state is 0.

Episode End An episode terminates when a player moves into a hole or when a player reaches the goal state.

The frozen lake game can be visualized as follows:

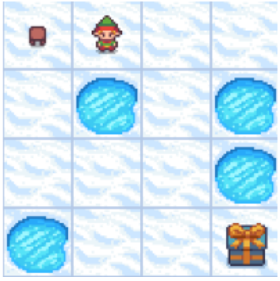


Figure 1: A visualization of the Frozen Lake game on a 4×4 grid from OpenAI Gym [1].

In the above representation, the frozen lake environment is depicted on a 4×4 grid, where the locations of the holes were placed randomly and can be clearly seen. The goal state is marked with a present in the very rightmost lower corner of the grid, and we can see that the player has made one step to the right from the starting position.

1.2 Relation of Problem to Reinforcement Learning

1.2.1 Off-Policy Learning Methods

Off-policy learning methods, including multi-step off-policy approaches and per-decision methods with control variates, are advanced techniques in reinforcement learning. They each address a fundamental reinforcement learning concept, which is how an agent can learn a target policy while following a behavior policy to generate experiences.

Multi-step methods extend off-policy learning by incorporating n-step returns. Instead of solely relying on immediate rewards, such as in 1-step Temporal Difference methods, or waiting until an episode ends to update rewards, such as in Monte Carlo methods, this approach finds a balance between the two by considering a future number of n steps.

Per-decision methods also extend the idea of off-policy learning methods, but in a different way than multi-step methods. Per-decision methods introduce control variates to reduce the variance in reward updates. Control variates act as additional terms in the learning process, helping to stabilize those updates made.

1.2.2 The Frozen Lake Game

The frozen lake game provides us with a simple, yet challenging environment that perfectly encapsulates the core concepts of reinforcement learning. While the rules of the game are simple, the environment can still provide complex challenges, especially on larger grids or when more obstacles (holes) are placed on the grid. The agent must learn to navigate this environment with obstacles, which mimics real-world decision-making processes, making it well-suited to various reinforcement learning tasks.

2 Problem

2.1 Problem Statement

Exercise 7.10 from Chapter 7 of *Sutton and Barto*:
Devise a small off-policy prediction problem and use it to show that the off-policy learning algorithm using (7.13) and (7.2) is more data efficient than the simpler algorithm using (7.1) and (7.9).

We note that equations (7.13), (7.2), (7.1), and (7.9) are given in Chapter 7 of *Sutton and Barto* [2] by:

$$G_{t:h} = \rho_t(R_{t+1} + \gamma G_{t+1:h}) + (1 - \rho_t)V_{h-1}(S_t) \tag{7.13}$$

$$\text{where } t < h < T \text{ and } G_{h:h} = V_{h-1}(S_h)$$

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \gamma[G_{t:t+n} - V_{t+n-1}(S_t)] \qquad (7.2)$$
$$\text{where } 0 \leq t < T$$

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \qquad (7.1)$$

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)] \qquad (7.9)$$
$$\text{where } 0 \leq t < T$$

and $\rho_{t:t+n-1}$ is the *importance sampling ratio* given by:

$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)} \qquad (7.10)$$

2.2 Approach

2.2.1 Originally Stated Question

In our approach, we model a small off-policy prediction problem by using the Frozen Lake environment from OpenAI Gym [1]. Specifically, we simulate a player navigating across the lake using two distinct algorithmic approaches:

Algorithm One implements a **per-decision method with control variates** by combining Equations 7.13 and 7.2, as outlined in **Section 2.1**. Equation 7.13 represents an off-policy n-step return with control variates, which calculates the total return, G, by considering the next n steps. This return is then used as a target for updating the value estimates, V, as described in Equation 7.2. Equation 7.2 updates the value estimates for the current state by incorporating the observed reward and the value of the next state, taking advantage of bootstrapping to refine those estimates.

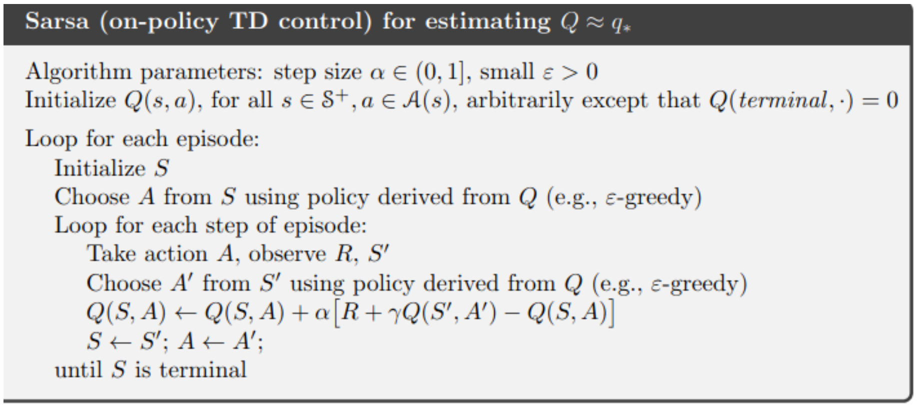
Algorithm Two implements a **multi-step off-policy method** by combining equations 7.9 and 7.1, as described in **Section 2.1**. Equation 7.9 updates the state value estimates, V, by incorporating rewards and future states values over the next n time steps. This update is used in tandem with equation 7.1, which computes the total return G by recursively summing the immediate rewards and the discounted future returns.

We implement and evaluate both algorithms on a 4×4 grid representation of the frozen lake environment. The exact grid is one that we randomly generate with the help of Gym [1]. We simulate each algorithm over $5,000$ episodes, with each episode having a maximum of 100 time steps. The frozen lake environment includes a parameter, *is_slippery*, that controls the stochasticity of the player's movements across the lake. In our implementation, we set *is_slippery* to be false.

2.2.2 Additional Question Asked

To deepen our analysis, we also investigate how the agent performs in the same frozen lake environment when following a **SARSA On-Policy TD Control Method**. To accomplish this, we follow the exact **SARSA On-Policy TD Control** algorithm given in **Section 6.4**, **page 130** of the book [2].

The pseudocode for this specific algorithm is given as follows:



```
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., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-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
```

Figure 2: Psuedocode for the SARSA On-Policy TD Control Algorithm that we follow in our implementation as given by the book [2].

2.3 Motivation Behind Approach

2.3.1 Why implement algorithms One and Two?

Algorithms One and Two offer contrasting approaches to off-policy learning, particularly in how they make learning updates and incorporate control variance.

Algorithm One includes control variates, which reduce the variance in off-policy learning updates. This can lead to more stabilized updates, which can contribute to more reliable convergence. This can be particularly advantageous in certain scenarios such as when the behavior policy, b, significantly differs from the target policy, π.

Algorithm Two on the other hand, does not use control variates in its implementation. It instead makes updates directly based on the n step return. This means that this approach is simpler and computationally less complex since it avoids the additional complexity that comes from adding a control variate term. The advantages from implementing algorithm two come from its simplicity and straightforward implementation process.

2.3.2 Why use SARSA as an additional question?

The originally stated question (provided in **Section 2.1** focuses on *off-policy* methods, so we were especially curious in the additional exploration of SARSA as it allowed us to compare the agent's performance in the frozen lake environment in an *on-policy* setting instead.

SARSA works by directly updating the value of the current policy based on the actions that the agent actually takes. Algorithms One and Two in contrast, make updates based on the target policy, π, that again, may differ from the behavior policy, b. By implementing SARSA as an additional question, we are able to investigate how those differences in policy decisions impact the agent's ability to learn and perform in the frozen lake environment. We were particularly excited to see if a difference in policy strategy would have a significant impact on the results we found or not.

2.3.3 Why model the problem in a frozen lake environment?

We implement algorithms One, Two and SARSA on a 4×4 grid representation of the Frozen Lake environment provided by Gym [1].

The recorded class lectures on Canvas dedicated a significant amount of time to exploring the Frozen Lake environment, seeming like they used it as the primary example to introduce new reinforcement learning algorithms for a couple weeks. After finishing Programming Assignment 2, which was centered on the racetrack environment, we were eager for a different environment, wanting a fresh, new environment to work with after spending so much time with the racetrack one. Since we had become well-acquainted with the Frozen Lake game through the recorded class lectures, we wanted to see how the algorithms in this problem would perform with the frozen lake environment!

The Grid The Frozen Lake environment from Gym has a 4×4 grid by default. We decided to keep this default 4×4 grid for the sake of simplicity. Following the specifications of the environment given by Gym, we also created the 4×4 grid by randomly generating it. Doing so helps simulate certain aspects in the environment, such as real-world variability.

is_slippery We decided to set the environment's *is_slippery* parameter to be false to eliminate any added complexity of stochastic transitions that it could introduce into the problem. Since the problem statement in **Section 2.1** specifies keeping the problem simple, we were motivated to ensure that the agent's movement across the grid lake was deterministic, leading to decreased unpredictability in the agent's movements.

Number of Episodes We experimented with episode counts ranging from 500 to $100,000$ to determine the optimal number of episodes to run when training. The number of episodes influences how much information the agent is able to collect and thus, learn from. During this exploration, we noted that values below $5,000$ would result in lower success rates, while values above $5,000$ did not show any significant improvement from $5,000$ itself. This suggested that $5,000$ was some type of plateau, where additional episodes provided minimal benefit. This observed behavior can be attributed to the simplicity and small environment that we were working in, so too few of episodes led to underfitting while too many episodes led to diminished returns, as the agent had already learned the optimal policy within the smaller environment.

Number of Maximum Steps Per Episode We chose to have a maximum of 100 possible time steps for each of the $5,000$ episodes. This limit was well suited to the smaller 4×4 grid environment that we had, still providing the agent with opportunities to explore while fitting with the small size and complexity of the environment itself.

3 Code

We implemented algorithms One, Two and SARSA in a frozen lake environment as described above in a jupyter notebook, titled *FrozenLake.ipynb*.

3.1 The Frozen Lake Environment Set-Up

We created the 4×4 frozen lake grid environment with OpenAI's Gym [1].

The randomly generated grid that we used to run the algorithms with and generate our results with was given by:

```
SFFH
FFFF
FFFF
FFFG
```

where S represents the start state, F represents a part of the lake that is frozen over, H represents a part of the lake where a hole has formed, and G represents the goal state. In all randomly generated grids, the position of S and G are fixed, while the locations of H and F are random and vary. For our experiment, we ran all 3 algorithms we are interested in (One, Two, and SARSA) on the above grid environment, which we note only has one hole in it.

3.2 Algorithms One and Two

After setting up the frozen lake environment via OpenAI's Gym [1] in our jupyter notebook, we defined helper functions to implement algorithms One and Two. Both algorithms use equation 7.10 to calculate the importance sampling ratio, following the exact formula provided in **Section 2.1** above. Equation 7.10 requires both a behavior policy and a target policy. In line with standard practice for defining these policies, we used a random policy for the behavior policy, b, and a greedy policy for the target policy, π. We then ran both algorithms through all episodes, using the environment setup described in **Section 3.1**. For algorithm One, we applied equations 7.13 and 7.2, while for algorithm Two, we used equations 7.9 and 7.1.

3.3 SARSA On-Policy TD Control Algorithm

To implement SARSA in our problem, we exactly followed the pseudocode provided in **Figure 2** of **Section 2.2.2** above. We ran through all $5,000$ episodes on the exact same environment given in **Section 3.1**.

4 Expectation of Results

4.1 Originally Stated Question

We anticipated that algorithm One would outperform algorithm Two in our environment. Since it is explicitly stated in the problem statement given in **Section 2.1** that algorithm One is a more advanced algorithm than algorithm Two, we expected it to be more data-efficient. Algorithm One uses control variates, meaning that it should be able to reduce variance without compromising the expected update value. This can help enhance stability, which leads to more robust learning.

4.2 Additional Question Asked

We expected SARSA to perform worse than algorithms One and Two with our environment here. Since SARSA is an on-policy algorithm, it updates its value function based on the current policy, which could result in suboptimal updates if the policy has not efficiently explored. In contrast, the off-policy methods used in algorithms One and Two allow the agent to learn from a broader range of experiences, improving data efficiency as the agent has access to more information. Because SARSA requires the agent to follow the policy it is optimizing, it may be more restricted in its ability to update the policy.

5 Results

In assessing the data efficiency of each algorithm, we focused on the number of steps taken to reach the goal, the percentage of episodes in which the goal state was reached, and the percentage of episodes where the agent did not reach the goal, either because it fell into a hole or ended the episode in a frozen state. Each of these three metrics provide insight into how efficiently each algorithm uses the data. A lower number of steps to reach the goal indicates better learning efficiency of the data as the agent is able to achieve the goal with fewer interactions with the environment. A higher percentage of successful goal completions suggests that the algorithm is learning a more robust and efficient policy.

5.1 Results from Originally Stated Question

Algorithm One:

average steps taken to reach the goal state over all episodes	30.66 steps
% time goal state was reached across all episodes	41.28 %
% time goal state was not reached across all episodes	58.72 %

Table 1: Results from running algorithm One (equations 7.13 + 7.2) on the frozen lake grid environment specified in **Section 3.1** over 5,000 episodes.

Algorithm Two:

average steps taken to reach the goal state over all episodes	30.72 steps
% time goal state was reached across all episodes	43.28 %
% time goal state was not reached across all episodes	56.72 %

Table 2: Results from running algorithm Two (equations 7.9 + 7.1) on the frozen lake grid environment specified in **Section 3.1** over 5,000 episodes.

5.2 Results from Additional Question Asked

average steps taken to reach the goal state over all episodes	7.03 steps
% time goal state was reached across all episodes	97.52 %
% time goal state was not reached across all episodes	2.48 %

Table 3: Results from running SARSA On-Policy TD Control Algorithm on the frozen lake grid environment specified in **Section 3.1** over 5,000 episodes.

6 Analysis of Results

6.1 Originally Stated Question

The actual results we obtained from running algorithms One and Two in our experiment did not meet our expectations. We firmly anticipated algorithm One, which uses control variates and more complex update methods, to outperform algorithm Two, which was simpler and made updates directly based on the n-step return. However, from our results, we observe that the two algorithms basically performed the exact same as one another.

While it is possible that these unexpected results could be due to the small grid and simpler environment or limited exploration leading to diminished returns with not enough data, I believe these factors are unlikely. I ran the experiments with up to $100,000$ episodes, and the results remained consistent with those stated in **Section 5** of this report, with the same number of steps and the same percentage rates being displayed across all the data. This suggests that diminished returns are not contributing to the obtained results. To me, this instead suggests that the environment and data size alone are not the main contributors to the unusual results.

Therefore, I suspect that there may be an issue with my code implementation, though I was unsuccessful in pinpointing exactly what the issue was. It could be that I somehow misapplied equations 7.13 + 7.2 for algorithm One or equations 7.9 + 7.1 for algorithm Two. Any small or subtle mistake in my code could have led to these unintended consequences, causing incorrect convergence. However, to the best of my knowledge, I followed all the formulas and procedures as outlined.

6.2 Additional Question Asked

The actual results we obtained from running SARSA in our experiment also did not meet our expectations. We again, firmly anticipated algorithms One and Two, which employ an off-policy approach, to outperform the on-policy approach of SARSA in our experiment. However, SARSA dramatically outperformed both algorithms One and Two when run on the same environment as them.

While we originally thought that the on-policy nature of SARSA would lead to it performing worse than algorithms one and two, it might be the very reason why SARSA actually performed better. Since SARSA is on-policy, it updates its value function based on the current policy that the agent is following. This could have led to making more stable updates than those made with algorithms One and Two. In addition, SARSA is abl to update the policy continually based on the information gathered from each episode, which might have made it more able to adapt to the environment here.

Algorithms One and Two follow off-policy methods, which intend to learn from a broader set of information across all episodes. However, since our environment was so small here, only being a 4×4 grid, the exploration of algorithms One and Two might have been more limited here than we originally thought, which could have contributed to them performing worse than SARSA here.

7 Conclusion

From this programming assignment, I have gained a deeper understanding of the critical roles that the choice of algorithms, policies, and environments play in shaping the dynamics of a reinforcement learning problem. By comparing algorithms One and Two with SARSA, I have learned how the specific choice of algorithm *and* policy can dramatically impact the learning process of the agent and the results obtained. For example, the on-policy nature of SARSA resulted in more defined learning, while the off-policy approaches resulted in less refined results in this context.

The outcomes I achieved from this experiment also emphasized the importance of experimenting with different parameters to fine-tune the environment to best achieve optimal results. At the end of this experiment, I am also left wondering if the unexpected results were due to any possible limitations of the frozen lake environment, it might be the case that it is not ideal for the algorithms employed in this assignment. To investigate this, I would be interested in testing the algorithms across a range of different environments to better understand their behavior and performance, as well as to gain a better understanding on where I have gone wrong in my code.

8 Extra Credit

For the extra credit portion of this assignment, I completed exercises 7.1 and 7.4 from Chapter 7 of *Sutton and Barto*. Both are non-programming tasks that were not covered in the live Zoom meetings nor the pre-recorded lectures.

8.1 Exercise 7.1

Exercises 7.1 states:

> In Chapter 6 we noted that the Monte Carlo error can be written as the sum of TD errors (6.6) if the value estimates don't change from step to step. Show that the n-step error used in (7.2) can also be written as a sum of TD errors (again if the value estimates don't change) generalizing the earlier result.

Solution:
Equation (6.6) is given in Section 6.1 from the book [2] as:

$$\begin{aligned} G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) \\ &= \delta_t + \gamma(G_{t+1} - V(S_{t+1})) \\ &\ldots \\ &= \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k \end{aligned}$$

Recall from **Section 2.1** of this paper that (7.2) is given as:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \gamma[G_{t:t+n} - V_{t+n-1}(S_t)]$$

where we note that the n-step error is $G_{t:t+n} - V_{t+n-1}(S_t)$

From equation (6.6) we can rewrite this n-step error as follows:

$$G_{t:t+n} - V_{t+n-1}(S_t) = R_{t+n+1} + \gamma G_{t+n-1} - V_{t+n}(S_t)$$

$$= R_{t+n+1} + \gamma V_{t+n}(S_{t+1}) - V_{t+n}(S_t) + \gamma G_{t+n_1} - \gamma V_{t+n}(S_{t+1})$$

note that $\delta_{t:t+n} = R_{t+n+1} + \gamma V_{t+n}(S_{t+1}) - V_{t+n}(S_t)$ so this can be rewritten as

$$= \delta_{t:t+n} + \gamma(G_{t+n+1} + V_{t+n}(S_{t+1}))$$

$$= \delta_{t:t+n} + \gamma[R_{t+n+2} + \gamma G_{t+n+2} - V_{t+n+1}(S_{t+2}]$$

$$= \delta_{t:t+n} + \gamma\delta_{t+1:t+n+1} + \gamma^2(G_{t+n+2} + V_{t+n+1}(S_{t+1}))$$

$$= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_{t+k:n+k}$$

Thus, we have shown that the n-step error used in (7.2) can also be written as a sum of TD errors.

8.2 Exercise 7.4

Exercises 7.4 states:

> Prove that the n-step return of Sarsa (7.4) can be written exactly in terms of a novel TD error, as:

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{min(t+n,T)-1} \gamma^{k-t}[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)] \tag{7.6}$$

> where (7.4) is given as:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1}R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{7.4}$$

[2]

Solution:
We will show that we can derive (7.4) from (7.6).

So, starting with (7.6) and expanding its sum we get the following:

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{min(t+n,T)-1} \gamma^{k-t}[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k)]$$

$$= Q_{t-1}(S_t, A_t) + \gamma^0[R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_{t-1}(S_t, A_t)]$$

$$+\gamma^1[R_{t+2} + \gamma Q_t(S_{t+2}, A_{t+2}) - Q_{t-2}(S_t, A_t)]$$

$$...$$

$$+\gamma^{n-1}[R_{t+n} + \gamma Q_t(S_{t+n}, A_{t+n}) - Q_{t-n}(S_t, A_t)]$$

distributing γ across all terms we get

$$= Q_{t-1}(S_t, A_t) + R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_{t-1}(S_t, A_t)$$

$$+\gamma R_{t+2} + \gamma^2 Q_t(S_{t+2}, A_{t+2}) - \gamma Q_{t-2}(S_t, A_t)$$

$$...$$

$$+\gamma^{n-1} R_{t+n} + \gamma^n Q_t(S_{t+n}, A_{t+n}) - \gamma^{n-1} Q_{t-n}(S_t, A_t)$$

canceling out terms leaves with us

$$= Q_{t-1}(S_t, A_t) + R_{t+1} - Q_{t+1}(S_t, A_t) + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

which can be rewritten as

$$= R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

which we note is exactly (7.4).

Thus, we have shown that the n-step return of Sarsa (7.4) can be written exactly in terms of a novel TD error (such as 7.6).

References

[1] OpenAI Gym. Frozen lake environment — openai gym, 2024.

[2] Richard S. Sutton and Barto Andrew G. *Reinforcement learning : an introduction.* The MIT Press, 2018.