COMP 138 RL: Homework 3

Nanxi Liu

March 28, 2025

1 Introduction

Both the Monte Carlo (MC) method and one-step temporal difference (TD) methods have been ways for reinforcement learning (RL) agents to learn with flexibility by episodes. In the MC method, the agent updates the learned action-value functions at the end of each episode. In one-step TD method, the agent updates its action-value functions at every step of each episode. However, these two methods seem to be on two ends of a spectrum, and now the question is: is there a method that is in between? The answer is yes. The n-step TD method is a generalization of both MC and one-step TD. In the n-step TD method, we are given the flexibility of deciding how many steps we can take in each episode before updating the value functions. If n is set to 1, then we will be implementing one-step TD; if n is set to the length of the episode, then we will be implementing MC. The n-step methods can be used in both on-policy (eg. SARSA) and off-policy methods. When used in off-policy methods, a sampling importance weight ρ is used when updating the value functions, much like what is used in the MC off-policy methods. For off-policy n-step methods, an additional mechanism, per decision importance sampling, can be used for more efficient learning. See the formulas below for the difference between per decision importance sampling (1.2) and original updates for G (1.1), the total discounted return overtime.

$$G_{t:h} = R_{t+1} + \gamma G_{t+1:h+1}, \quad t < h < T \tag{1.1}$$

$$G_{t:h} = \rho_t \left(R_{t+1} + \gamma G_{t+1:h+1}\right) + (1 - \rho_t) V_{h-1}(S_t), \quad t < h < T \tag{1.2}$$

2 Goal

This research has two goals. The first is to compare the data efficiencies of regular off-policy n-step TD learning and control-variate off-policy n-step TD learning. The second is to compare the data efficiency of on-policy TD learning with the other two. The author will produce three algorithms according to the description and pseudocode in Sutton and Barto [1], and run the same experiments on all three.

3 Problem Description

This problem is based on Exercise 7.10 from Sutton and Barto [1]. The author has adopted the random-walk problem design for this research. An agent is given a "maze" [1, 2, 3, ... ,n], with n equal to 15. Its starting point is in position 1 and the episode terminates when the agent reaches position n. Based on the policy it is given, the agent can either choose to go left or right. Moving left while at position 1 and moving right while at position n will not produce any movements, but will incur in a reward. Each action of moving to the right will have a reward of 1 and each action of moving to the left will have a reward of -1. The performance of each agent is based on the mean square error derived from the true values of the states and the learned values of the states in each episode.

4 Experiment Design

There are three types of algorithms implemented (regular n-step off-policy, control variate n-step off-policy, and n-step on-policy). All three algorithms are based on the pseudocode (Figure 1) from chapter 7 of Sutton and Barto [1]. With the addition of an important sampling ratio for the off policy methods when updating the value function (see updates to the Q function in Figure 2 for reference). There are two types of probabilities of selecting an action of going right for the target policies (or just the policy in the case of on-policy method) - 0.5 and 0.8. There are also four different n-step values - 2, 4, 6, 8. Each combination of agent, probability, and steps are trained on 200 epochs for 20 times. Each experiment is given a seed for the results to be replicable. The mean squared error output for agents with the same policy and n-step values are plotted in the same graph for comparison purposes. The rest of the parameters needed for training, such as the discount factor (0.8), the learning rate (0.2), and the structure of the maze are the same for all agents. The maze is in the form of a list consisting of integers that increment from 1 to 15 taking steps of 1: [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15], with 15 being the terminal state. The true value of each state is calculated using the Bellman equation and the policy probability.

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t + 1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

Figure 1: Pseudocode from Sutton and Barto [1]

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s, a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n + 1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t + 1$
  | else:
  | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:\tau+n})$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho \left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

Figure 2: Pseudocode from Sutton and Barto [1]

5 Hypothesis

According to Sutton and Barto's analysis [1], the addition of the control variate can avoid situations where a reward becomes zero if an action selected by the behavior policy will never be selected by the target policy, therefore resulting in a lower variance. I'm expecting that the control-variate method will be more data efficient than the regular method because of its lower variance. I'm also expecting that the on-policy method is more data efficient than the off-policy methods because on-policy methods only follow one consistent policy and will not be distracted by the behavior different from the target policy.

6 Results

The following graphs are produced by taking 4 and 8 steps using probability of 0.5 and 0.8. The rest of the results can be found in the appendix. The y-axis of each graph represents the value of the mean squared error and the x-axis the number of episodes. The error of the on-policy method has been averaged using a moving window of 10 episodes for a smoother curve. In each graph, we can observe that the on-policy method produces the least error compared to the off-policy methods and also converges much faster, while the control-variate off-policy method consistently has a lower mean squared error and variance than the regular method.

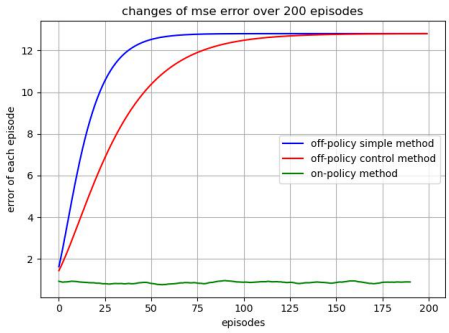

Figure 3: MSE error for probability 0.5 and n = 4

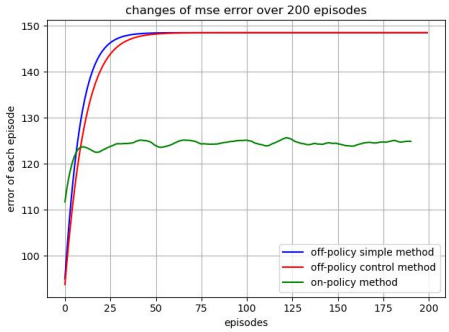

Figure 4: MSE error for probability 0.8 and n = 4

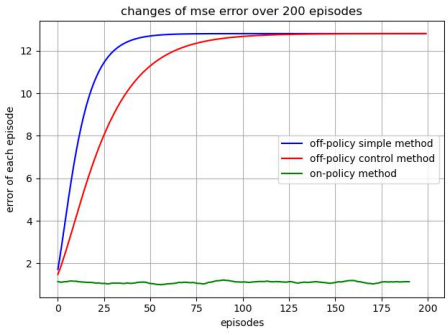

Figure 5: MSE error for probability 0.5 and n = 8

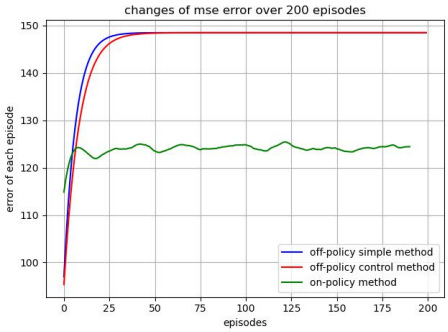

Figure 6: MSE error for probability 0.8 and n = 8

7 Discussion

Based on the Results section, we can see that the output has confirmed the author's hypothesis on the data efficiencies of on-policy and off-policy methods. One interesting observation is that the error curve for the on-policy method differs significantly based on the probability of going right. For going right with probability p of 0.8, the error curve increases significantly before converging while the error curve for probability of 0.5 converges very fast. This could potentially due to the state values being disproportionately updated in $p = 0.8$, as it tends to go right, therefore updating more states near the terminal states than near the starting state; whereas for $p = 0.5$, the states are given an equal chance for update, therefore having less variance in the error curve. In terms of data efficiency, the on-policy method consistently outperformed the off-policy methods. This confirmed the author's analysis in the hypothesis section that on-policy methods having value function updates that correspond to its "behavior" policy (from the perspective of off-policy methods, on-policy methods have the same target and behavior policies) leads to errors with lower variance. The comparison of the two off-policy error curves has also confirmed that the method with control-variate has consistently lower error than the regular method, making the control-variate method more data efficient.

8 Conclusion

Overall, this article has discussed and confirmed two interesting phenomena - the superior data efficiency of the control-variate off-policy n-step TD method versus the regular off-policy n-step TD method, and the overall data efficiency of on-policy methods in the random walk problem. The different ways of updating the value functions has led to these differences, and it gives the broader audience a good chance to reflect on how we can play around with the logic of value update, as a change as small as adding the importance sampling ratio to the expected return can increase data efficiency.

References

[1] Andrew B Richard S. *Reinforcement Learning, An Introduction*. MIT Press, 2018.

9 Appendix

See the Python script and instructions on how to replicate the experiment using the following link: `https://github.com/Nanxi-Flaneuse/Reinforcement-Learning/tree/main/assignments/TD`

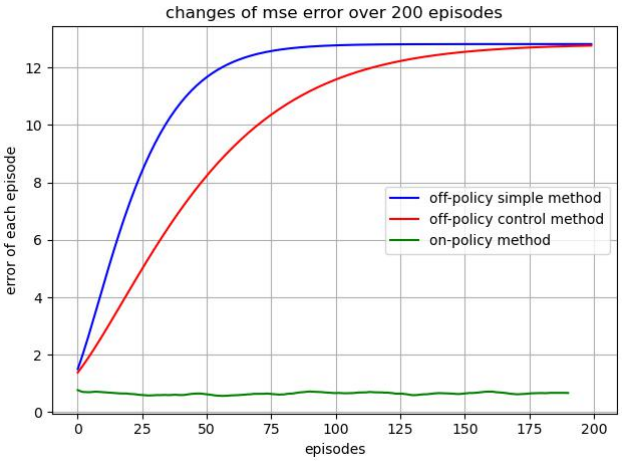

Figure 7: error curves for p = 0.5 and n = 2

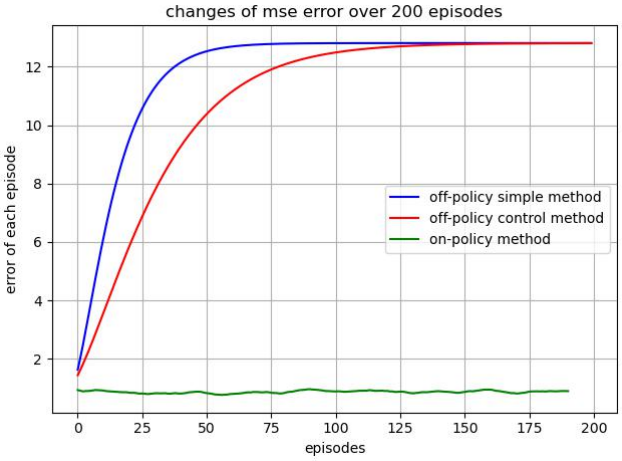

Figure 8: error curves for p = 0.5 and n = 4

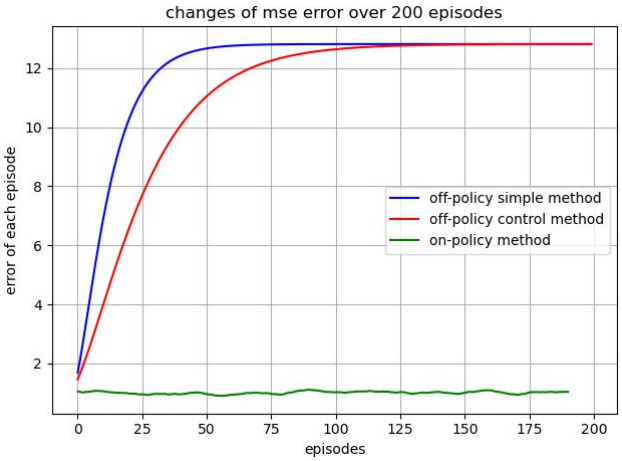

Figure 9: error curves for p = 0.5 and n = 6

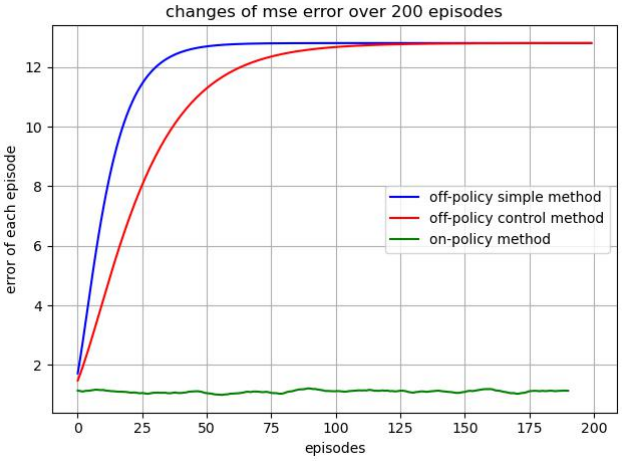

Figure 10: error curves for p = 0.5 and n = 8

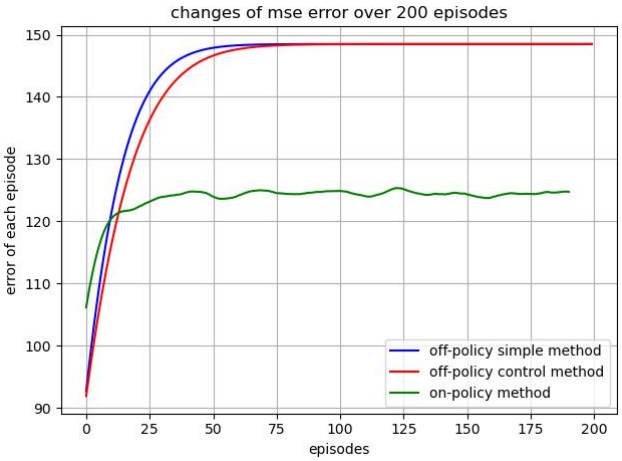

Figure 11: error curves for p = 0.8 and n = 2

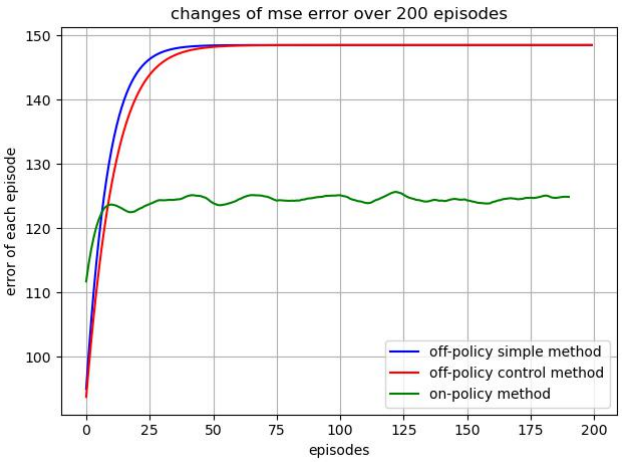

Figure 12: error curves for p = 0.8 and n = 4

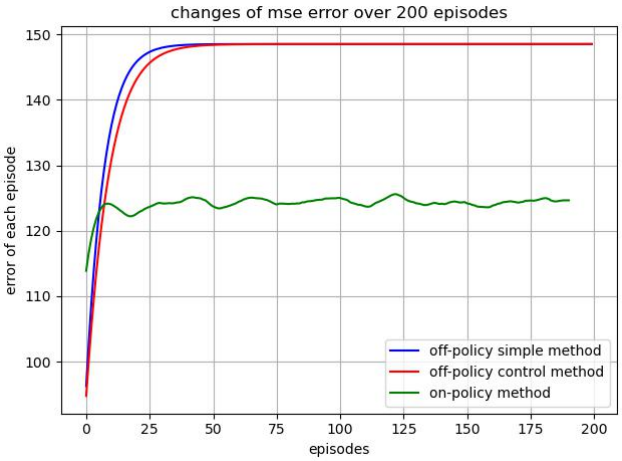

Figure 13: error curves for p = 0.8 and n = 6

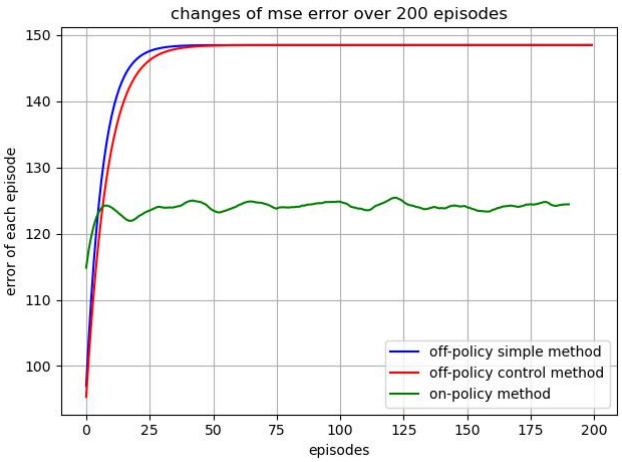

Figure 14: error curves for p = 0.8 and n = 8