UNIVERSITEIT VAN AMSTERDAM
Faculteit Natuurwetenschappen, Wiskunde en Informatica

Reinforcement Learning Week 2

By

Sergio Alejandro Gutierrez Maury, 11821353
Ilse Feenstra, 13628356

September 21, 2023

Homework: Coding Assignment - Monte Carlo

1. In the chapter 5.6 of the book, we are given incremental update rule for weighted importance sampling (equations 5.7-5.8). However, in the coding assignment, we will use ordinary importance sampling which uses a different update rule. Start with $V_n = \frac{\sum_{k=1}^{n} W_k \cdot G_k}{n}$ and derive the incremental update rule for ordinary importance sampling. Your final answer should be in the form of $V_n = V_{n-1} + a \cdot (b - V_{n-1})$.

We start with:

$$V_n = \frac{\sum_{k=1}^{n} W_k \cdot G_k}{n}$$

and keep in mind that:

$$V_{n-1} = \frac{\sum_{k=1}^{n-1} W_k \cdot G_k}{n-1}$$

$$(n-1) \cdot V_{n-1} = \sum_{k=1}^{n-1} W_k \cdot G_k$$

The derivation starts here:

$$V_n = \frac{1}{n} \cdot (W_n \cdot G_n + \sum_{k=1}^{n-1} W_k \cdot G_k)$$

$$V_n = \frac{1}{n} \cdot (W_n \cdot G_n + (n-1) \cdot V_{n-1})$$

$$V_n = \frac{1}{n} \cdot (W_n \cdot G_n + n \cdot V_{n-1} - V_{n-1})$$

$$V_n = \frac{W_n \cdot G_n + n \cdot V_{n-1} - V_{n-1}}{n}$$

$$V_n = \frac{W_n \cdot G_n}{n} + V_{n-1} - \frac{V_{n-1}}{n}$$

$$V_n - V_{n-1} = \frac{W_n \cdot G_n}{n} - \frac{V_{n-1}}{n}$$

$$V_n - V_{n-1} = \frac{W_n \cdot G_n - V_{n-1}}{n}$$

$$V_n - V_{n-1} = \frac{1}{n} \cdot (W_n \cdot G_n - V_{n-1})$$

$$V_n = V_{n-1} + \frac{1}{n} \cdot (W_n \cdot G_n - V_{n-1})$$

where $a = \frac{1}{n}$ and $b = W_n \cdot G_n$

2. Coding part

3. What are two advantages of Monte Carlo over Dynamic Programming? When would you use the one or the other algorithm?

Monte Carlo (MC) methods have some key advantages over Dynamic Programming (DP). Firstly, MC methods are good at dealing with high-dimensional problems. This is because they're based on sampling, which means they only need to consider states that show up during actual trajectories. This is useful for high-dimensional problems where it's not practical to store every possible state or state-action pair, unlike DP which needs to compute all state transitions.

Secondly, MC methods don't need a model of the environment to start with. They learn from actual experiences or episodes, so there is no need to know transition dynamics or reward functions beforehand. This is different from DP, which needs a full model of the environment to predict the value of each state. So, MC methods are a better fit when the dynamics of the environment are unknown. However, DP can be more efficient for small, deterministic problems, and when the model of the environment is known. So if that is the case, you would choose DP.

Homework: SARSA and Q-learning

Consider the MDP in Figure 1 with states A (starting state), B, C, T (terminal state) and rewards for corresponding actions indicated in the diagram. The reward for action a_1 in state C is parametrized by n. We use an $\varepsilon - greedy$ policy with $\varepsilon = 0.2$ for SARSA and the same policy as a behavior policy for Q-learning. We do not use discounting ($\gamma = 1$).

1. Suppose that $n \in (-\infty, 2)$ and we run SARSA until convergence.

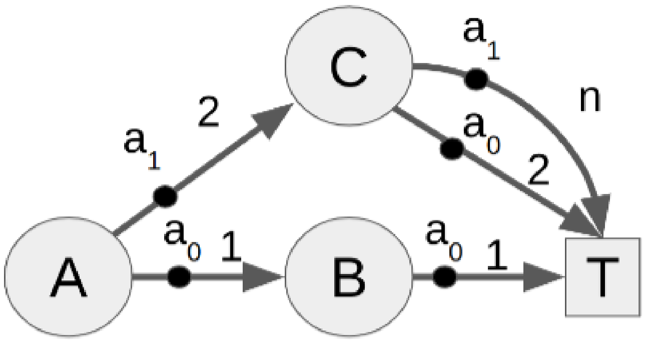

Figure 1: Figure 3 from assignment document

a. What would be the final Q-values for all states and actions? When necessary write your answer using an expression that includes n.

The SARSA formula is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \left[R(s,a) + \gamma \cdot Q(s',a') - Q(s,a)\right] \tag{1}$$

γ is equal to one. Since we run the algorithm until convergence, we can set $\alpha = 1$. This way, there is no learning anymore because the part before the arrow is the same as the part after the arrow. This simplifies the equation to:

$$Q(s,a) \leftarrow R(s,a) + Q(s',a') \tag{2}$$

Substituting the values for the rewards, we get the following Q-values:

* For State A:
  - For action a_0: $Q(A,a_0) = 1 + Q(B,a_0) = 1 + 1 = 2$
  - For action a_1:
    $Q(A,a_1) \leftarrow Q(A,a_1) + \alpha[2 + 0.9 \cdot Q(C,a_0) + 0.1 \cdot Q(C,a_1)Q(A,a_1)]$
    $Q(A,a_1) = 2 + 0.9 \cdot 2 + 0.1 \cdot n$
    $Q(A,a_1) = 3.8 + 0.1 \cdot n$
* For State B:
  - For action a_0: $Q(B,a_0) = 1$
* For State C:
  - For action a_0: $Q(C,a_0) = 2$
  - For action a_1: $Q(C,a_1) = n$

b. For which range of values of n will the final policy prefer to choose action a_1 more often in state A? For which range of values will it prefer to choose a_0? Use final Q-values to explain your answers.

The final policy will prefer to choose action a_1 more often in state A if $Q(A,a_1) > Q(A,a_0)$, which means:

$$3.8 + 0.1 \cdot n > 1 + Q(B,a_0)$$

Since $Q(B,a_0) = 1$, we can write:

$$3.8 + 0.1 \cdot n > 2$$

$$0.1 \cdot n > -1.8$$

$$n > -18$$

For $n \in (-18, 2)$, the final policy will prefer a_1 more often, if not in this range, then the policy will choose a_0 more often.

2. Suppose that $n \in (-\infty, \infty)$ and we run Q-learning until convergence.

a. What would be the final Q-values for all states and actions? When necessary write your answer using an expression that includes n.

Given discounting $\gamma = 1$, the Q-learning update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \max_a Q(s',a') - Q(s,a) \right)$$

We set α to 1 again.

For the provided MDP:

1. $Q(A,a_0)$: Directly leads to state B.

$$Q(A,a_0) = 1 + \max(Q(B,a_0)) = 1 + 1 = 2$$

2. $Q(A,a_1)$: Directly leads to state C.

$$Q(A,a_1) = 2 + \max(Q(C,a_0), Q(C,a_1))$$

For $n \leq 2$:

$$Q(A,a_1) = 2 + 2 = 4$$

For $n > 2$:

$$Q(A,a_1) = 2 + n$$

3. $Q(B,a_0)$: Directly leads to the terminal state.

$$Q(B,a_0) = 1$$

4. $Q(C,a_0)$: Directly leads to the terminal state.

$$Q(C,a_0) = 2$$

5. $Q(C,a_1)$: Directly leads to the terminal state.

$$Q(C,a_1) = n$$

b. For which range of values of n will the final policy prefer to choose action a1 more often in state A? For which range of values will it prefer to choose a0? Use final Q-values to explain your answers.

To determine the preference of the final policy for action a_1 over a_0 in state A, we compared the Q-values of these actions:

$$Q(A,a_0) = 2$$
$$Q(A,a_1) = 2 + \max(2,n)$$

To deduce the preferred action, we evaluate these Q-values:

1. $Q(A,a_0) > Q(A,a_1)$

$$2 > 2 + \max(2,n)$$

Given that the maximum value between 2 and n will always be ≥ 2, this inequality never holds. Thus, the policy will never prefer a_0 over a_1.

2. $Q(A, a_0) < Q(A, a_1)$

$$2 < 2 + \max(2, n)$$

This inequality breaks down into two scenarios:

a) For $n \leq 2$:

$$2 < 4$$

This is always true.

b) For $n > 2$:

$$2 < 2 + n$$

Given $n > 2$, this is also always true.

From the above evaluations, for all values of n, the final policy after Q-learning has converged will prefer a_1 over a_0 in state A.

3 Suppose that n = 1 and we run both SARSA and Q-learning until convergence and record the final performance (average return after convergence). Then we use an adjusted MDP, where there is an extra action a2 in A, that moves to B and gives reward 1. We also run both algorithms until convergence and record the final performance. Would the final performance (average return after convergence) of SARSA or Q-learning be different across these two MDPs? Explain your answer. Note: You don't need to compute the actual values for final performance to answer this question. Make an argument that explains/supports your answer.

The final performance (average return after convergence) of both SARSA and Q-learning will not be significantly different across these two MDPs. The addition of the action a_2 does not provide any advantage or disadvantage concerning the rewards, because it has the same value as a_0. Likewise, a_1 with n=1 is lower than a_0. Both algorithms will recognize the optimal action a_2 to take in state A and a_0 for state C. The learning might slightly vary during the initial exploration, but the converged policies will be similar.