Homework 2

Elias Dubbeldam, Ankur Satya

September 23, 2022

3.5 Coding Assignment - Monte Carlo

1. We derive the incremental update rule for ordinary importance sampling as follows:

$$\begin{aligned}
V_n &= \frac{1}{n}\sum_{k=1}^{n} W_k G_k \\
&= \frac{n-1}{n-1}\frac{1}{n}\left(W_1 G_1 + W_2 G_2 + \dots W_{n-1} G_{n-1}\right) + \frac{1}{n} W_n G_n \\
&= \frac{n-1}{n}\frac{1}{n-1}\sum_{n=1}^{n-1} W_k G_k + \frac{1}{n} W_n G_n - V_{n-1} + V_{n-1} \\
&= \frac{n-1}{n} V_{n-1} + \frac{1}{n} W_n G_n - V_{n-1} + V_{n-1} \\
&= \left(\frac{n-1}{n} - \frac{n}{n}\right) V_{n-1} + V_{n-1} + \frac{1}{n} W_n G_n \\
&= V_{n-1} + \frac{1}{n}\left(W_n G_n - V_{n-1}\right)
\end{aligned}$$

   So the final answer is indeed of the form $V_n = V_{n-1} + a * (b - V_{n-1})$ with:

$$\Rightarrow a = V_{n-1}, \quad b = W_n G_n$$

   If we shift $n \Rightarrow n+1$, we obtain an update rule:

$$V_{n+1} = V_n + \frac{1}{n+1}\left(W_{n+1} G_{n+1} - V_n\right)$$

   Rewriting it to pseudo-code gives:

$$V_n \leftarrow V_n + \frac{1}{n+1}\left(W_{n+1} G_{n+1} - V_n\right)$$

2. The predictions with off-policy Monte Carlo of the Blackjack game, following part 2 of the notebook, are visualized in Figure 2. However, the results do not look correct. We are not sure whether this is due to an mistake in the derived update rule above, the implementation of the ordinary importance sampling algorithm in the code, or something else.

   The results seem quite off from what we would expect. We would expect similar results as in the previous setup, but without the 'ridge', as there is in the case of 'normal' MC in Figure 1, for the player's sum value around 21. For the ordinary importance sampling, we would expect there would be a more gentle slope between the high and low player's sum values. We expect this because the MC would also better learn with a random policy for the cases where the player's sum is just below 20.

3. *What is the difference between Dynamic Programming and Monte Carlo? When would you use the one or the other algorithm?*

   They key difference between Dynamic Programming (DP) and Monte Carlo (MC) methods is twofold. First, MC methods operate on sample experience. Therefore, MC methods can be used for direct learning without a model. DP methods, on the other hand, require complete knowledge of all possible transitions and the environment. This is because of how they learn. MC goes until the end of the episode until it updates, while DP include one-step transitions directly. Second, MC methods do not bootstrap. That means that value estimates are not updated on the basis of other value estimates.

   When one has access to the model, the transition probabilities, one would use DP, because it learns directly without waiting until the end of the episode. However, having access to the model is often not possible in real world examples. If that is the case, one would use MC.

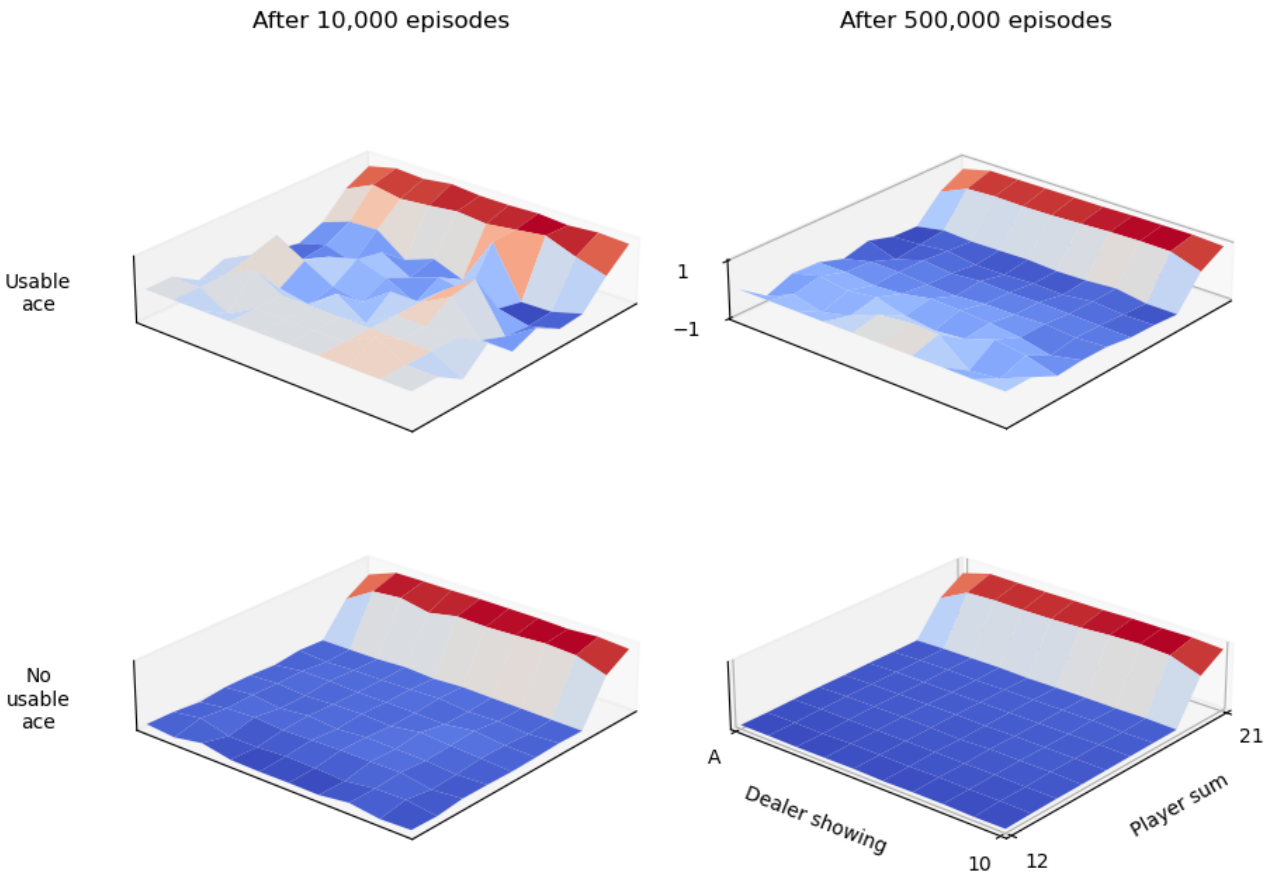


Figure 1: Approximate state-value functions of the Blackjack game with Monte Carl policy evaluation. The policy sticks only on 20 and 21.

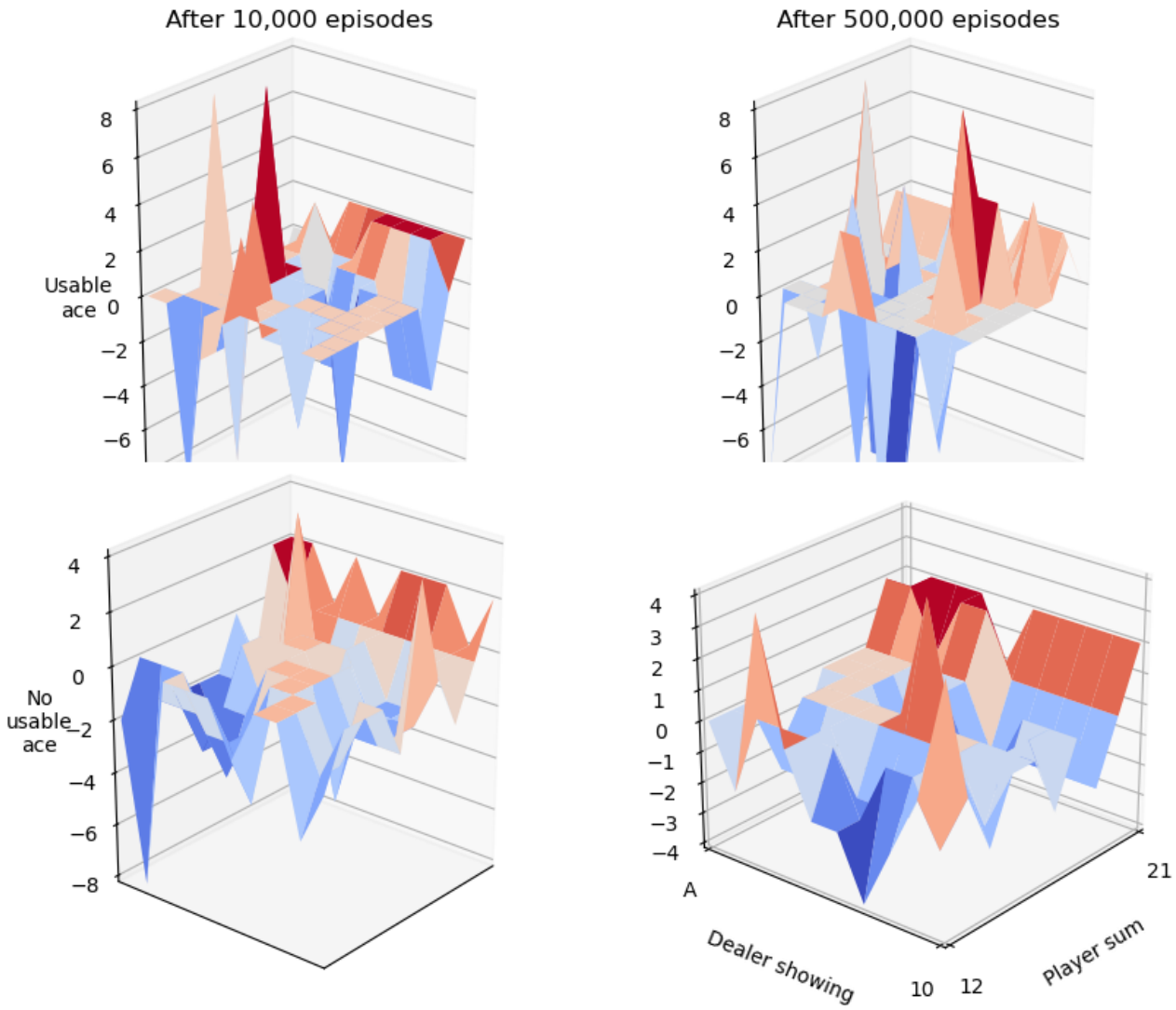


Figure 2: Approximate state-value functions of the Blackjack game with ordinary importance sampling Monte Carl policy evaluation. The target policy sticks only on 20 and 21, the behavior policy randomly sticks or hit in each state.

4.4 SARSA and Q-learning

See the written notes below.

4.4.

(1)(a) $Q(B, a_0) = 1 \cdot 1 = 1$

- C will choose the greedy action with the probability :- $p = 1 - \varepsilon + \frac{\varepsilon}{N}$

where $\varepsilon = 0.9$ and $N = 2$

$\Rightarrow p = 0.95$; for the state C the greedy action will be a_0 since $n \in (-\infty, \infty)$

$\Rightarrow Q(C, a_0) = 0.95 \times 2 = 1.90$

$Q(C, a_1) = 0.05 \times n$

- For A, the greedy action a_1 will be chosen with the probability $p = 0.95$

$\Rightarrow Q(A, a_0) = 0.05 \times \sum_a Q(B, a)$

$= 0.05 \times 1 = 0.05$ — (1)

$Q(A, a_1) = 0.95 \times \sum_a Q(C, a)$

$= 0.95 \times [1.9 + 0.05n]$

$= 1.805 + 0.0475n$ — (2)

(b) Using (1) & (2) from 4.4.1.(a) we can say that the final policy will choose action a_1 more than a_0 in the state A when :-

$Q(A, a_1) \geq Q(A, a_0)$

$\Rightarrow 1.805 + 0.0475n > 0.05$

$\Rightarrow n > -36.94$

for $n < -36.94$, it will choose action a_0.

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

- $Q(C, a_0) = \begin{cases} 2 & n < 2 \\ 0 & \text{otherwise} \end{cases}$

$Q(C, a_1) = \begin{cases} 0 & \text{if } n < 2 \\ 4 & \text{otherwise} \end{cases}$

- ~~$Q(A, a_0) =$~~

Let's assume that we are not using the greedy policy for now. In that case

$Q(A, a_0) = 2$

$Q(A, a_1) = \begin{cases} 4 & n < 2 \\ n+2 & \text{otherwise} \end{cases}$

We can clearly see that $Q(A, a_1)$ is always greater than $Q(A, a_0)$. So in the presence of the greedy policy like in the case of Q-learning:

$Q(A, a_1) = \begin{cases} 4 & n < 2 \\ n+2 & \text{otherwise} \end{cases}$

$Q(A, a_0) = 0$.

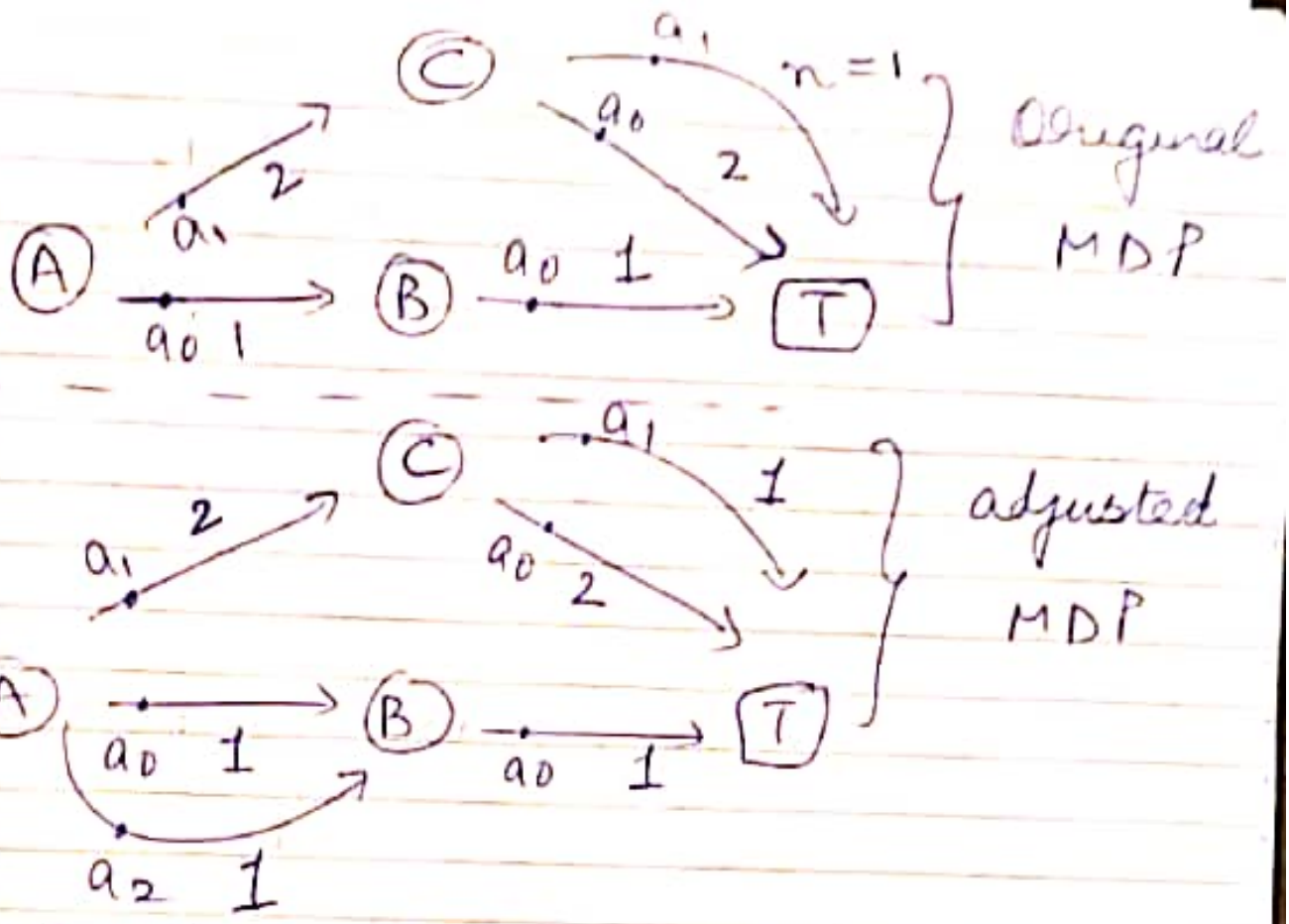
It wasn't clear from the question whether we should assume absolute greedy policy for Q learning or ε-greedy

Above we have showed the final Q-value if the ~~target~~ behaviour policy was absolute greedy.

If ε-greedy policy was to be used as the behaviour policy, then the final Q-values would be the same as in the case of SARSA.

(b) Since $Q(A, a_1)$ is always greater than $Q(A, a_0)$, the final policy will always choose a_1 when in the state A.

(3)



The only difference between the two MDPs is an additional action available for the state A in the adjusted MDP.

This will change the probability of choosing the greedy action when in state A for the adjusted MDP. It will be:

$$P = 1 - \varepsilon + \frac{\varepsilon}{N} \quad \text{where } \varepsilon = 0.1 \text{ & } N = 3$$

$$= 0.933$$

This value is less than the probability of choosing the greedy action when in State A for the original MDP as shown in 4.4.1.

Q-values for all the other states will remain the same. So, the average return after convergence will be less in the case of adjusted MDP.

In the case of Q-learning, the average return after convergence will be the same in both the cases since the greedy action will remain the same at every state.

TO conclude:

	MDP	adjusted MDP	
SARSA	v_1	v_2	$v_1 > v_2$
Q-learning	v_3	v_4	$v_3 = v_4$

where v_1, v_2, v_3 & v_4 are placeholders for average return after convergence.