UNIVERSITEIT VAN AMSTERDAM
Faculteit Natuurwetenschappen, Wiskunde en Informatica

Reinforcement Learning Week 3

By

Sergio Alejandro Gutierrez Maury, 11821353
Ilse Feenstra, 13628356

October 1, 2023

Homework: Coding Assignment - Temporal Difference Learning

1. Coding part

> Done in Jupyter Notebook

2. In the lab notebook you plotted the average returns during training for Q-learning and SARSA algorithms. Which algorithm achieves higher average return? Do you observe the same phenomenon as in the Example 6.6 in the book? Explain why or why not.

> In the notebook, Q-learning achieved a higher average return compared to SARSA. In example 6.6 from the book, SARSA achieved a higher average return compared to Q-learning. So, we did not observe the phenomenon in our notebook compared to the book. This can be explained because we are comparing two different problems. The problem of the notebook, the windy grid, has lower risky pathways compared to example 6.6, the cliff world. Therefore, learning the optimal pathway for the windy grid is less risky compared to the optimal pathway of the cliff world. In problems where risky pathways lead to significant penalties, SARSA might perform better because of its more conservative approach, while Q-learning might outperform SARSA when there are no high penalties because it tends to be more aggressive in pursuing the optimal strategy. This is in line with what we observed.

3. In the cliff world, which algorithm achieves a smaller variance return? Explain why.

> **SARSA will have a smaller variance in return compared to Q-learning.**
>
> Being an on-policy algorithm, SARSA tends to learn a more cautious policy in the cliff world. When using an exploration strategy like ε-greedy, SARSA often learns to take a safer route that avoids getting too close to the cliff, even if it's slightly longer. This behaviour is because SARSA accounts for the possibility of exploration (and hence the risk of falling off the cliff due to a non-optimal action) in its updates. In contrast, Q-learning learns the value of the optimal policy, because it is an off-policy method. It will find that the quickest path is to move adjacent to

the cliff. However, when combined with an ε-greedy exploration strategy, every once in a while, the agent will take a random action. If this happens when the agent is next to the cliff, it's likely to fall in. This leads to a large negative reward occasionally.

While Q-learning might achieve a higher average return when it doesn't fall into the cliff, it also occasionally receives a significant penalty when it does. This leads to a higher variance in returns. SARSA, on the other hand, by taking the safer route consistently, tends to avoid the cliff and hence has more consistent (and typically slightly lower) returns. This consistency results in a smaller variance in returns for SARSA.

In conclusion, in the cliff world environment, SARSA usually has a smaller variance in return because it learns to consistently avoid the risky cliff area, whereas Q-learning occasionally receives a substantial negative reward due to its attempt to take the most optimal path next to the cliff combined with the stochasticity of the exploration strategy.

4. Under which condition SARSA and Q-learning could be the same algorithm?

Q-learning is an off-policy algorithm, while SARSA is an on-policy algorithm. If the behaviour policy is the same as the target policy, Q-learning will be the same algorithm as SARSA.

5. Which of the two algorithms is considered an off-policy control method? Why?

Q-learning is considered an off-policy control method because it learns the value function of the optimal policy regardless of the policy it's following to explore, while SARSA is on-policy because it learns the value function of the policy it's currently following.

Homework: Maximization Bias

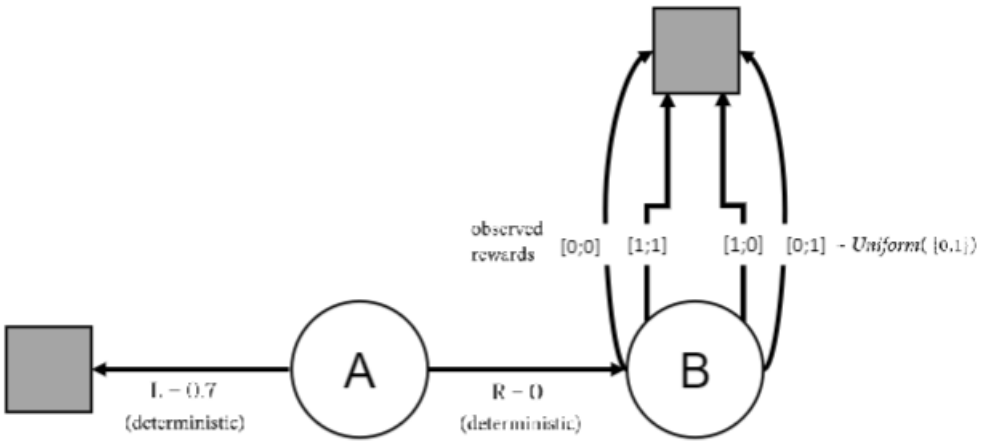

Figure 1: Figure 3 from Assignment 3

Consider the undiscounted MDP in Figure 1 where we have a starting state A with two actions (L,R), one ending in the terminal state T (with $V(T) = 0$) and always yielding a reward of 0.7 and another action that transitions to state B with zero reward. From state B we can take four different actions, each transitioning to the terminal state and yielding a reward of either 0 or 1.

with equal probability. Suppose that from a behavior policy we sample two episodes where we try action L and eight episodes where we try action R followed by an action from state B (each action in B is used twice). The behavior policy chooses actions in B uniformly. The observed rewards for each of the four actions from state B are indicated in the Figure, e.g. the rightmost action received a reward of 0 and a reward of 1. Note: *The scenario in this exercise is a bit artificial, as it is designed to highlight the effect of maximization bias.*

1. What are the Q-values of the four outgoing actions from state B after the updates corresponding to the described observations? Give your answers for both Q-learning and expected SARSA. Use an initial Q-value of 0.7 for all state-action pairs and a learning rate $\alpha = 0.2$. For the purpose of this exercise, the behavior policy is not updated.

The formulas for Q-learning:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \cdot [R_{t+1} + \gamma \cdot \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$$

and Expected SARSA:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \cdot [R_{t+1} + \gamma \cdot \sum_a \pi(a|S_{t+1}) \cdot Q(S_{t+1},a) - Q(S_t,A_t)]$$

where $\pi(a|S_{t+1})$ is the probability of taking action a in state S_{t+1} under the target policy.

Introducing some notation, we can name the actions as follows:

$$\{a_{LT}, a_{RB}\}$$

Where a_{LT} is the action of choosing left and ending in a terminal state, and similarly for a_{RB}, is the action of choosing right and entering state B.

From state B, we observe these actions with their rewards:

$$\{a^1_{RB}, a^2_{RB}, a^3_{RB}, a^4_{RB}\} = \{[0;0],[1;1],[1;0],[0;1]\} \sim Uniform([0,1])$$

In each episode where action a_{RB} is taken from state A, one of the four actions from state B is also taken, yielding a reward of either 0 or 1 and ending in state T. The Q-values of these actions are updated as follows:

$$Q(B,a^i_{RB}) \leftarrow 0.7 + 0.2[R_i + \gamma \cdot \max_a Q(T,a) - 0.7]$$

and for SARSA:

$$Q(B,a^i_{RB}) \leftarrow 0.7 + 0.2[R_i + \gamma \cdot \sum_a \pi(a|T) \cdot Q(T,a) - 0.7]$$

where R_i is the observed reward for action a^i_{RB}.

Given that all actions in state B transition to the terminal state T with $V(T) = 0$, we have: $\max_a Q(T,a) = 0$ for Q-learning, and $\sum_a \pi(a|T) \cdot Q(T,a) = 0$ for Expected SARSA.

Using an initial Q-value of 0.7 for all state-action pairs and a learning rate $\alpha = 0.2$, the calculations for both Q-learning and Expected SARSA can be done as follows:

$$Q(B,a^1_{RB}) \leftarrow 0.7 + 0.2[0 - 0.7] = 0.56$$

$$Q(B,a^2_{RB}) \leftarrow 0.7 + 0.2[1 - 0.7] = 0.76$$

$$Q(B,a^3_{RB}) \leftarrow 0.7 + 0.2[1 - 0.7] = 0.76$$

$$Q(B, a^4_{RB}) \leftarrow 0.7 + 0.2[0 - 0.7] = 0.56$$

After two updates for each action from state B, the Q-values are:

$$Q(B, a^1_{RB}) \leftarrow 0.56 + 0.2[0 - 0.56] = 0.448$$

$$Q(B, a^2_{RB}) \leftarrow 0.76 + 0.2[1 - 0.76] = 0.808$$

$$Q(B, a^3_{RB}) \leftarrow 0.76 + 0.2[0 - 0.76] = 0.608$$

$$Q(B, a^4_{RB}) \leftarrow 0.56 + 0.2[1 - 0.56] = 0.648$$

2. For the sake of this example, ignore any updates during the first ten transitions described above to the Q-values $Q(A,L)$ and $Q(A,R)$. So assume these are still at their initial value of 0.7. What happens to $Q(A,L)$ if now we execute action "L" in state A and perform an update? Similarly, what happens to the value of $Q(A,R)$ if we execute "R" in state A and perform an update? Give your answer again for both Q-learning and expected SARSA.

For Q-learning:

- If we execute action "L" in state A, the Q-value update would be:

$$Q(A, a_{LT}) \leftarrow Q(A, a_{LT}) + \alpha[R_{t+1} + \gamma \cdot \max_a Q(T, a) - Q(A, a_{LT})]$$

Since action "L" transitions to the terminal state T with a reward of 0.7 and $Q(T,a) = 0$ for any action a, we have:

$$Q(A, a_{LT}) \leftarrow 0.7 + 0.2[0.7 + 0 - 0.7] = 0.7$$

- If we execute action "R" in state A, the Q-value update would be:

$$Q(A, a_{RB}) \leftarrow Q(A, a_{RB}) + \alpha[R_{t+1} + \gamma \cdot \max_a Q(B, a) - Q(A, a_{RB})]$$

Since action "R" transitions to state B with a maximum reward of 0 and all actions from state B have an initial Q-value of 0.7, and assuming a $\gamma = 1$, we have:

$$Q(A, a_{RB}) \leftarrow 0.7 + 0.2[0 + 1 \times 1 - 0.7] = 0.76$$

For expected SARSA:

- If we execute action "L" in state A, the Q-value update would be:

$$Q(A,a_{LT}) \leftarrow Q(A,a_{LT}) + \alpha[R_{t+1} + \gamma \cdot \sum_a \pi(a|T)Q(T,a) - Q(A,a_{LT})]$$

Since $\pi(a|T) = 0$ and $Q(T,a) = 0$ for any action a, we have:

$$Q(A,a_{LT}) \leftarrow 0.7 + 0.2[0.7 - 0.7] = 0.7$$

- If we execute action "R" in state A, the Q-value update would be:

$$Q(A,a_{RB}) \leftarrow Q(A,a_{RB}) + \alpha[R_{t+1} + \gamma \cdot \sum_a \pi(a|B)Q(B,a) - Q(A,a_{RB})]$$

Since $\pi(a|B) = \frac{1}{4}$ for any action a, and assuming $\gamma = 1$, we have:

$$Q(A,a_{RB}) \leftarrow 0.7 + 0.2[0 + \frac{1}{4}(Q(B,a^1_{RB}) + Q(B,a^2_{RB}) + Q(B,a^3_{RB}) + Q(B,a^4_{RB})) - 0.7]$$

So:

$$Q(A,a_{RB}) \leftarrow 0.7 + 0.2[\frac{1}{4}(0 + 1 + 0.5 + 0.5) - 0.7] = 0.66$$

3. What are the true state-action values that we would expect to get (after convergence) if we continued sampling episodes.

If we continued sampling episodes, the final values would be:

$$Q(A,a_{LT}) = 0.7$$

$$Q(A,a_{RB}) = 0.5$$

$$Q(B,a^1_{RB}) = 0.5$$

$$Q(B,a^2_{RB}) = 0.5$$

$$Q(B,a^3_{RB}) = 0.5$$

$$Q(B,a^4_{RB}) = 0.5$$

4. This problem suffers from maximization bias. Explain where this can be observed. Do both Q-learning and SARSA suffer from this bias in this example? Why/why not?

In the true state-action scenario, we know that the value of $Q(A, a_{RB})$ is $\frac{1}{2}$. This is because it represents an average over all possible state-action values of B. Ideally, we would choose $Q(A, a_{LT}) = 0.7$, as it's greater than $Q(A, a_{RB})$.

However, Q-learning might lead us to estimate $Q(A, a_{RB})$ as 1 and, hence, consider action a_{RB} as optimal. This happens due to the max_a operator in its formula, which can introduce this bias.

On the contrary, Expected SARSA doesn't have this bias. It estimates the average directly and, thus, provides a more accurate representation of the state-action values.

5. To circumvent the issue of maximization bias, we can apply Double Q-learning. Use the given example to explain how Double Q-learning alleviates the problem of maximization bias.

We can introduce two Q-learning functions: Q_1 and Q_2. In this approach, one function, Q_1, is used to choose the maximizing action, while the other, Q_2, estimates its value. This dual estimation reduces the risk of having overly optimistic value estimates. In the given example, where Q-learning overestimated $Q(A, a_{RB})$ as 1, Double Q-learning would distribute overestimations more evenly between Q_1 and Q_2, providing a more accurate state-action value estimation, mitigating the bias.

6. What are the update equations for Double Expected SARSA with an "$\varepsilon - greedy$" target policy? Provide both equations for Q-value update and policy update.

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma \cdot \sum_a \pi(a|S_{t+1}) \cdot Q_2(S_{t+1}, a) - Q_1(S_t, A_t)]$$

$$Q_2(S_t, A_t) \leftarrow Q_2(S_t, A_t) + \alpha \cdot [R_{t+1} + \gamma \cdot \sum_a \pi(a|S_{t+1}) \cdot Q_1(S_{t+1}, a) - Q_2(S_t, A_t)]$$

$$\pi(a|S_t) = \begin{cases} \frac{\varepsilon}{|A(S_t)|} + 1 - \varepsilon & \text{if } a = \arg\max_a [Q1(S_t, a) + Q2(S_t, a)] \\ \frac{\varepsilon}{|A(S_t)|} & \text{otherwise} \end{cases}$$

Homework: Gradient Descent Methods

1. Given the following Stochastic Gradient Descent (SGD) method for state-value prediction:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha_t \cdot [U_t - \hat{v}(S_t, w_t)] \nabla \hat{v}(S_t, w_t)$$

with:

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Determine which of the following estimators are unbiased estimators of $v^{\pi}(s)$:

- $U_t = G_t$
- $U_t = G_{t:t+n}$
- $U_t = R + \hat{v}(S', w_t)$
- $U_t = \sum_{a,s',r} \pi(a|S_t) p(s', r|S_t, a)[r + \gamma \hat{v}(s', w_t)]$

Afterwards, answer the following questions:

a. Relate these four estimators to reinforcement learning algorithms

b. Among the provided estimators, given the previously mentioned assumption about the learning rate, which one guarantees convergence to a local optimum? Why?

c. Give an example of a learning rate that guarantees convergence to a local optimum for one of the estimators that guarantees such convergence?

d. Why do we continue to use biased estimators with function approximation? Can you provide an example of an environment where using a biased estimator could be more advantageous than using an unbiased one?

First part of the question:

To determine an unbiased estimator U_t for $v^{\pi}(s)$, the estimator needs to fulfil $\mathbf{E}[U_t|S_t = s] = v^{\pi}(s)$, for each t.

- For $U_t = G_t$:

This is the Monte Carlo target and by definition, it is an unbiased estimator:

$$v^{\pi}(s) \doteq \mathbf{E}[G_t|S_t = s]$$

- For $U_t = G_{t:t+n}$:

This is an estimator for Bootstrapping targets with n-step returns. This estimator depends on the vector $\mathbf{w}_t$, which makes it a biased estimator.

- For $U_t = R + \hat{v}(S', \mathbf{w}_t)$:

This estimator uses $\mathbf{w}_t$ as an approximation of $v^{\pi}(s)$, which also makes it dependent on $\mathbf{w}_t$, and therefore biased.

- For $U_t = \sum_{a,s',r} \pi(a|S_t) p(s', r|S_t, a)[r + \gamma \hat{v}(s', w_t)]$:

This is the Dynamic Programming (DP) target, and also depends on the vector $\mathbf{w}_t$, making it a biased estimator.

Second part of the question:

a.)

- $U_t = G_t$: Monte Carlo target, returns.

- $U_t = G_{t:t+n}$: Bootstrapping targets with n-step returns

- $U_t = R + \hat{v}(S', \mathbf{w}_t)$: TD(0)

- $U_t = \sum_{a,s',r} \pi(a|S_t) p(s', r|S_t, a)[r + \gamma \hat{v}(s', w_t)]$: Dynamic Programming (DP), using bellman equations.

b.) Among the provided estimators, only the first one, $U_t = G_t$, guarantees convergence to a local optimum. This is because it is an unbiased estimator of the true value function, and the SGD method converges to a local optimum if the expected update is in the direction of the negative gradient of a convex function. Since the other estimators are biased because they depend on the weight vector $\mathbf{w}_t$, which changes over time, they do not satisfy the conditions for convergence of SGD.

c.) A decreasing alpha that satisfies the standard stochastic approximation conditions. So, $\alpha = \frac{1}{n}$ satisfies this.

d.) We use biased estimators in semi-gradient methods (bootstrapping), and some reasons to use them are that they typically enable significantly faster learning, and enable learning to be continual and online, without waiting for the end of an episode.

An example environment could be the Mountain Car problem, in which an underpowered car must drive up a steep hill. The car cannot simply accelerate up the hill

because gravity is stronger than the engine, and thus, the car has to learn to use its momentum by driving back and forth on the opposite hill before reaching the goal at the top of the right hill. Initially, with this problem, the car will oscillate back and forth at the bottom of the hill, without actually reaching the top of the hill, this is because moving forward seems to yield better rewards, but it's not possible due to insufficient force. Therefore, using the Monte Carlo approach might take an extensive amount of time to even have a single episode for learning. However, it can gradually converge to the optimal policy with semi-gradient updates at each timestep (bootstrapping methods) and good initialization.

2. Gradient Monte-Carlo approximates the gradient of the mean squared value error (see e.g. equations 9.1, 9.4, and 9.5 in the book). Similarly, derive a weight update that minimizes the mean squared temporal difference error. Compare the result to Semi-Gradient TD, and comment on the origin of the name Semi-Gradient method.

Equation 9.1 from the book:

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in S} \mu(s)[v_\pi - \hat{v}(s,\mathbf{w})]^2$$

Equation 9.4 from the book:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t - \frac{1}{2}\alpha_t \nabla [v_\pi(S_t) - \hat{v}(s,\mathbf{w})]^2$$

Equation 9.5 from the book:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t [v_\pi(S_t) - \hat{v}(s,\mathbf{w})] \nabla \hat{v}(s,\mathbf{w})$$

For TD:

The semi-gradient update rule is:

$$w_{t+1} = w_t + \alpha[R + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)] \nabla \hat{v}(S_t, w)$$

The TD error can be written as:

$$TDE(w) = \sum_{s \in S} \mu(s) \delta_t^2$$

where δ is: $\delta_t = R_{t+1} + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(s, \mathbf{w})$

The mean squared temporal difference error can then be written as:

$$\overline{VE}(\mathbf{w}) \doteq \sum_{s \in S} \mu(s)[R_{t1} + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(s, \mathbf{w})]^2$$

To minimize this error using gradient descent, we update the weights w as:

$$\Delta w = -\alpha \nabla \text{TDE}(w)$$

Using the chain rule, we can derive the gradient:

$$\nabla \text{TDE}(w) = 2\delta_t \nabla \delta_t$$

Given δ_t as above, the gradient $\nabla \delta_t$ is:

$$-\nabla \hat{v}(S_t, w)$$

The weight update becomes:

$$\Delta w = \alpha[R + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)] \nabla \hat{v}(S_t, w)$$

$$w_{t+1} = w_t + \alpha[R + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)] \nabla \hat{v}(S_t, w)$$

The naming comes from the fact that the update rule does not follow the true gradient, but it approximates it with the biased estimator $R + \gamma \hat{v}(S', \mathbf{w}_t)$, which, as seen before, depends on $\mathbf{w}_t$. The semi-gradient method takes into account the effect of changing the weight vector w_t on the estimate but ignores its effect on the target.