Reinforcement Learning Tutorial 3, Week 4
— with solutions —
Dynamic Programming / Monte Carlo

Pavlos Andreadis, Adam Jelley, Sanjay Rakshit

February 2024

Overview: The current tutorial questions relate to material taught in weeks 2/3 of the 2023-24 Reinforcement Learning course. They aim at encouraging engagement with the course material and facilitating a deeper understanding.

This week you are presented with a couple of exercises related to Dynamic Programming and Monte Carlo methods asking you to do some computations by hand. For the Dynamic Programming problem, there is no need to run these till convergence. The solution of **Problem 1** is intended to show you enough to see how it works, and to give you the details on what it converged to and after how many steps. **Problem 1** requires two outer loops of the *Policy Iteration* algorithm, and the first *Policy Evaluation* converges after 15 updates (4 updates are enough to understand what is going on).

If you are so inclined, you could also write a quick script to tackle Problem 1. This can be a good exercise in understanding how the algorithm works and will help to prepare you for the coursework.

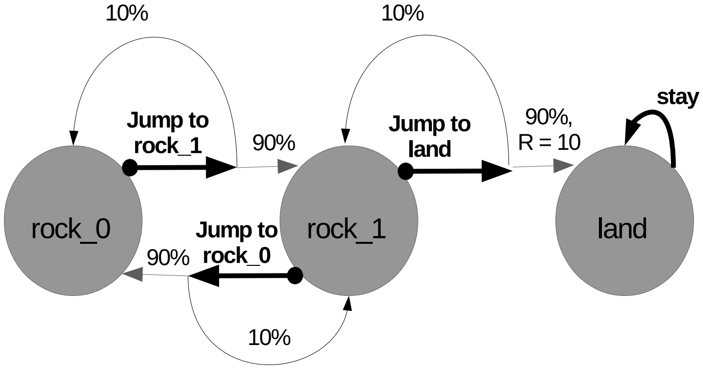
Generally, when considering Reinforcement Learning algorithms, note that most of them[1] can be understood as a specific instance of the *Generalised Policy Iteration* procedure: We iterate, until the policy converges, over one step of some variant of *Policy Evaluation* (which needs to improve the evaluation of the current policy at least by a little) and one step of some form of *Policy Improvement* (which we generally want to be at least somewhat greedy so that convergence is guaranteed). After the Policy Improvement step, we get a new policy whose evaluation (via state-value or action-value function) we can improve upon by the next Policy Evaluation step, and so forth until convergence. See Sutton and Barto [2018], section 4.6, page 86 for details.

The Monte Carlo policy evaluation methods in **Problem 2** could replace the Iterative Policy Evaluation step (a Dynamic Programming algorithm) of the Policy Iteration algorithm. [*Though we wouldn't usually do this while evaluating the state-value function. Why?*]

[1] Policy gradient methods don't generally follow this pattern.

Problem 1 - Dynamic Programming

Consider the Hop Along MDP from tutorial 2. Set the discount factor to $\gamma = 0.9$, and assume a reward of 10 for reaching land (and 0 otherwise). Apply two iterations of *Policy Iteration*, starting from a uniform initial policy.



Do the final values and policy agree with your intuition?

Answer:

Start with a uniform policy:
$\pi_0(rock_0, jump\_to\_rock_1) = 1$
$\pi_0(rock_1, jump\_to\_rock_0) = 0.5$
$\pi_0(rock_1, jump\_to\_land) = 0.5$

and an initial evaluation for this policy of:
$V_0(rock_0) = 0$
$V_0(rock_1) = 0$

$[V(land)$ is always 0. Why?]

We will use a convergence check of no change to any value state function greater than $\theta = 0.01$.

1st Iteration: *Policy Evaluation*

$$V_1(rock_0) = \pi_0(rock_0, jump\_to\_rock_1)(0.1{*}[0{+}\gamma V_0(rock_0)]{+}0.9{*}[0{+}\gamma V_0(rock_1)])$$
$$= 0$$

$$V_1(rock_1) = \pi_0(rock_1, jump\_to\_rock_0)(0.1{*}[0{+}\gamma V_0(rock_1)]{+}0.9{*}[0{+}\gamma V_0(rock_0)])$$
$$+\pi_0(rock_1, jump\_to\_land)(0.1 * [0 + \gamma V_0(rock_1)] + 0.9 * [10 + \gamma V(land)])$$
$$= 4.5$$

$max_{s \in \{rock_0, rock_1\}}[V_1(s) - V_0(s)] \geq \theta$

$$V_2(rock_0) = \pi_0(rock_0, jump\_to\_rock_1)(0.1{*}[0{+}\gamma V_1(rock_0)]{+}0.9{*}[0{+}\gamma V_1(rock_1)])$$
$$= 0.9 * 0.9 * 4.5 = 3.645$$

$$V_2(rock_1) = \pi_0(rock_1, jump\_to\_rock_0)(0.1{*}[0{+}\gamma V_1(rock_1)]{+}0.9{*}[0{+}\gamma V_1(rock_0)])$$
$$+\pi_0(rock_1, jump\_to\_land)(0.1 * [0 + \gamma V_1(rock_1)] + 0.9 * [10 + \gamma V(land)])$$
$$= 0.5 * (0.1 * [0 + 0.9 * 4.5])$$
$$+ 0.5 * (0.1 * [0 + 0.9 * 4.5] + 0.9 * [10 + 0.9 * 0])$$
$$= 0.405 + 4.5 = 4.905$$

$max_{s \in \{rock_0, rock_1\}}[V_2(s) - V_1(s)] \geq \theta$

$$V_3(rock_0) = \pi_0(rock_0, jump\_to\_rock_1)(0.1{*}[0{+}\gamma V_2(rock_0)]{+}0.9{*}[0{+}\gamma V_2(rock_1)])$$
$$= 0.1 * [0.9 * 3.645] + 0.9 * [0.9 * 4.905]$$
$$= 0.32805 + 3.97305 = 4.3011$$

$$V_3(rock_1) = \pi_0(rock_1, jump\_to\_rock_0)(0.1{*}[0{+}\gamma V_2(rock_1)]{+}0.9{*}[0{+}\gamma V_2(rock_0)])$$
$$+\pi_0(rock_1, jump\_to\_land)(0.1 * [0 + \gamma V_2(rock_1)] + 0.9 * [10 + \gamma V(land)])$$
$$= 0.5 * (0.1 * [0 + 0.9 * 4.905] + 0.9 * [0 + 0.9 * 3.645])$$
$$+ 0.5 * (0.1 * [0 + 0.9 * 4.905] + 0.9 * [10 + 0.9 * 0])$$
$$= 0.5 * (0.44145 + 2.95245) + 0.5 * (0.44145 + 9)$$
$$= 0.44145 + 1.476225 + 4.5 = 6.417675$$

$max_{s \in \{rock_0, rock_1\}}[V_3(s) - V_2(s)] \geq \theta$

$$V_4(rock_0) = \pi_0(rock_0, jump\_to\_rock_1)(0.1{*}[0{+}\gamma V_3(rock_0)]{+}0.9{*}[0{+}\gamma V_3(rock_1)])$$
$$\simeq 0.1 * [0.9 * 4.3011] + 0.9 * [0.9 * 6.4177]$$
$$\simeq 0.3871 + 5.1983 = 5.5854$$

$$V_4(rock_1) = \pi_0(rock_1, jump\_to\_rock_0)(0.1{*}[0{+}\gamma V_3(rock_1)]{+}0.9{*}[0{+}\gamma V_3(rock_0)])$$
$$+\pi_0(rock_1, jump\_to\_land)(0.1 * [0 + \gamma V_3(rock_1)] + 0.9 * [10 + \gamma V(land)])$$
$$\simeq 0.5 * (0.1 * [0 + 0.9 * 6.4177] + 0.9 * [0 + 0.9 * 4.3011])$$
$$+ 0.5 * (0.1 * [0 + 0.9 * 6.4177] + 0.9 * [10 + 0.9 * 0])$$
$$= 6.8195$$

$max_{s \in \{rock_0, rock_1\}}[V_4(s) - V_3(s)] \geq \theta$

Eventually, we have at time step $t = 14$:
$V_{14}(rock_0) = 7.2596$,
$V_{14}(rock_1) = 8.1681$;
and at time step $t = 15$:
$V_{15}(rock_0) = 7.2695$,

$V_{15}(rock_1) = 8.1753$;
which is enough to terminate the policy evaluation step with a convergence check for no value state change greater than 0.01.

Policy Improvement (1 step)

$\pi_1(rock_0, jump\_to\_rock_1) = 1$
$$\pi_1(rock_1, \text{argmax}_{a \in \{jump\_to\_rock_0, jump\_to\_land\}} \sum_{s' \in \{rock_0, rock_1, land\}} p(s'|rock_1, a) * [r(rock_1, a, s') + \gamma V(s')]) = 1$$

Therefore:
$\pi_1(rock_1, jump\_to\_rock_0) = 0$
$\pi_1(rock_1, jump\_to\_land) = 1$

2nd Iteration: *Policy Evaluation*

We can and will continue from our evaluations of the previous policy. After all, the evaluation of the new policy will be higher at all states. [*Why?*]

This time we converge at time step $t = 5$ with:
$V_5(rock_0) = 8.8028$,
$V_5(rock_1) = 9.8901$.

Policy Improvement (1 step)

The policy improvement step will not alter the policy (i.e. the policy is stable), and we terminate returning the evaluation above, and the policy:
$\pi^*(rock_0, jump\_to\_rock_1) = 1$
$\pi^*(rock_1, jump\_to\_rock_0) = 0$
$\pi^*(rock_1, jump\_to\_land) = 1$,
or, equivalently (with the common slight abuse of notation):
$\pi^*(rock_0) = jump\_to\_rock_1$
$\pi^*(rock_1) = jump\_to\_land.$

Hopefully, this optimal policy is what you would have intuitively expected. The optimal values should be (approximately) too: they are both slightly less than 10 due to discounting and the 10% chance of falling off the rock at each timestep, with $V^*(rock_0) < V^*(rock_1)$ as there is more discounting and chance of falling off before reaching the reward from rock 0 than from rock 1. The optimal policy and value function may not always be so intuitive, particularly as the MDPs we consider start to scale up!

This problem is intended to be answered by hand, to develop intuition for the inner workings of policy iteration. However, to provide an additional perspective, and to help bridge the gap between the tutorials and the coursework, an interactive notebook solution to this tutorial problem is also available as a GitHub Gist:
`https://gist.github.com/AdamJelley/d1e872426b0186980a15f1b6421250c2`

Problem 2 - Monte Carlo

Compute the state-value function for a given policy π and MDP without access to the MDP's model, using the following four episodes, in order:

$$\begin{gathered}
\text{rock}_0, 0, \text{rock}_0, 0, \text{rock}_1, 10, \text{land} \\
\text{rock}_1, 0, \text{rock}_0, 0, \text{rock}_1, 10, \text{land} \\
\text{rock}_0, 0, \text{rock}_0, -100, \text{sea} \\
\text{rock}_1, 0, \text{rock}_0, -100, \text{sea}
\end{gathered}$$

Note: The discount factor used here is $\gamma = 1$. [*Why is this acceptable here?*]

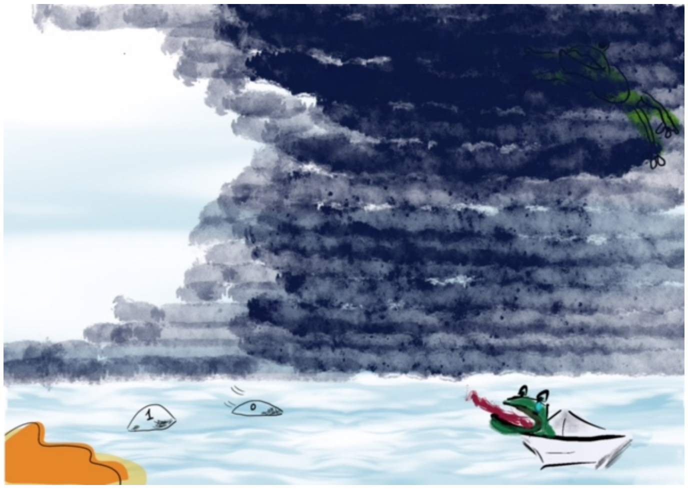


Figure 1: "Frog with More Problems." Image and title used with permission from Yana Knight and Andreadis [2021]

Part 1

Use first-time visit Monte Carlo to evaluate the state-value function at each state. Show your calculation.

Answer:

[*Wait! How come we don't need to know what the policy is?*]

Here we assume a discount factor of $\gamma = 1$. This is safe to do as the episodes terminate. Note that I asked for an application of the algorithm. As such, simply averaging across samples to get the final result will not suffice here. (I am putting the sequence of rewards for new samples in parentheses).

$V_0(rock_o) = 0$
$V_0(rock_1) = 0$ (optionally: $V(land) = V(sea) = 0$)

After episode 1:

$V_1(rock_o) = (0 + 0 + 10)/1 = 10$
$V_1(rock_1) = (10)/1 = 10$

After episode 2:

$V_2(rock_o) = [10 + (0 + 10)]/2 = 10$
$V_2(rock_1) = [10 + (0 + 0 + 10)]/2 = 10$

After episode 3:

$V_3(rock_o) = [10 + 10 + (0 - 100)]/3 = -26.7$
$V_3(rock_1) = V_2(rock_1) = 10$

After episode 4:

$V_4(rock_o) = [10 + 10 - 100 + (-100)]/4 = -45$
$V_4(rock_1) = [10 + 10 + (0 - 100)]/3 \simeq -26.7$

Part 2

Use every-time visit Monte Carlo to evaluate the state-value function at each state. Show your calculation.

Answer:

$V_0(rock_o) = 0$
$V_0(rock_1) = 0$ (optionally: $V(land) = V(sea) = 0$)

After episode 1:

$V_1(rock_o) = [(0 + 0 + 10) + (0 + 10)]/2 = 10$
$V_1(rock_1) = (10)/1 = 10$

After episode 2:

$V_2(rock_o) = [10 + 10 + (0 + 10)]/3 = 10$
$V_2(rock_1) = [10 + (0 + 0 + 10) + (10)]/3 = 10$

After episode 3:

$V_3(rock_o) = [10 + 10 + 10 + (0 - 100) + (-100)]/5 = -34$
$V_3(rock_1) = V_2(rock_1) = 10$

After episode 4:

$V_4(rock_o) = [10 + 10 + 10 - 100 - 100 + (-100)]/6 = -45$
$V_4(rock_1) = [10 + 10 + 10 + (0 - 100)]/4 = -17.5$

Part 3

How would you expect the state-values estimated by both first-time visit Monte Carlo and every-time visit Monte Carlo to change as the number of episodes considered goes to infinity? (No mathematical proofs required, just a short description of the expected values).

Answer:

The estimates of the state-values from both first-time visit and every-time visit Monte Carlo should converge to the true state value function v_π for the behaviour policy as the number of episodes considered goes to infinity (assuming both the policy and MDP are stationary).

If interested, for a full comparison and the proof of convergence for the more complex case of every-visit Monte Carlo, see Theorem 7 in Ref. Singh and Sutton [1996].

Part 4

Which are the absorbing states?

Answer:

The states *land* and *sea*, as they terminate the episodes.

However, this assumes that we have not terminated any episodes prematurely. This is something we might have to do, if we were to use these algorithms over an MDP without absorbing states. This is obviously beyond the scope of this quesitons, but imagine you are given a sample trajectory, such as:

$$\text{rock}_0, 0, \text{rock}_0$$

Would you be confident now that *land* and *sea* are absorbing states? No worries, we will came back to this in later tutorials.

References

Yana Knight and Pavlos Andreadis. ”Story of Yana”. http://storyofyana.com/, 2021.

Satinder P Singh and Richard S Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1):123–158, 1996.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.