Model-free Lunar Lander

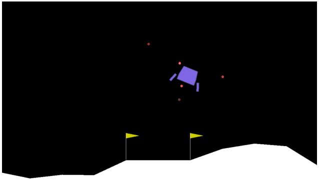
Marco Carollo

Reinforcement Learning
DSSC
Università degli Studi di Trieste

January 25, 2024

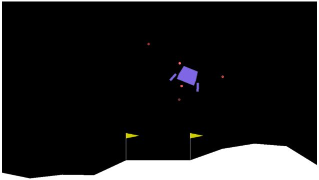
The problem at hand

This environment is a classic rocket trajectory optimization problem.



The problem at hand

This environment is a classic rocket trajectory optimization problem.



Goal: direct the agent to the landing pad as softly and fuel-efficiently as possible.

The Lunar Lander environment

The state space is continuous as in real physics, but the action space is discrete.

The Lunar Lander environment

The state space is continuous as in real physics, but the action space is discrete.
Complexity of the state space: $O(n^6 \times 2 \times 2)$

$$\text{state} = \begin{cases} \text{first coordinate of the lander} \\ \text{second coordinate of the lander} \\ \text{horizontal velocity} \\ \text{vertical velocity} \\ \text{tilt (angle)} \\ \text{angular velocity} \\ \text{leg 1 touching the ground} \\ \text{leg 2 touching the ground} \end{cases}$$

The Lunar Lander environment

The state space is continuous as in real physics, but the action space is discrete.
Four discrete actions are available: do nothing, fire left orientation engine, fire right orientation engine, and fire main engine.

Rewards

After every step a reward is granted, in this fashion:

- positive/negative the closer/further the lander is to the landing pad.
- positive/negative the slower/faster the lander is moving.
- more negative the more the lander is tilted.
- increased for each leg that is in contact with the ground.
- decreased each time an engine is firing.

Rewards

After every step a reward is granted, in this fashion:

- positive/negative the closer/further the lander is to the landing pad.
- positive/negative the slower/faster the lander is moving.
- more negative the more the lander is tilted.
- increased for each leg that is in contact with the ground.
- decreased each time an engine is firing.

The episode receives an additional positive/negative reward for landing/crashing.

Rewards

After every step a reward is granted, in this fashion:

- positive/negative the closer/further the lander is to the landing pad.
- positive/negative the slower/faster the lander is moving.
- more negative the more the lander is tilted.
- increased for each leg that is in contact with the ground.
- decreased each time an engine is firing.

The episode receives an additional positive/negative reward for landing/crashing.
The lander starts at the top center of the space with a random initial force applied to its center of mass. The landing pad is always at coordinates (0,0).

Discretization of the state space

Due to the state space being continuous, discretization techniques are implemented.

Naive state aggregation: A simple discretization of 10 values for each continuous variable leads to 400000 states!

Discretization of the state space

Effective state aggregation[1]: all the coordinates far from the center can be generalized into one single state because the agent will always tend to move in one direction, which helps reduce the state space.

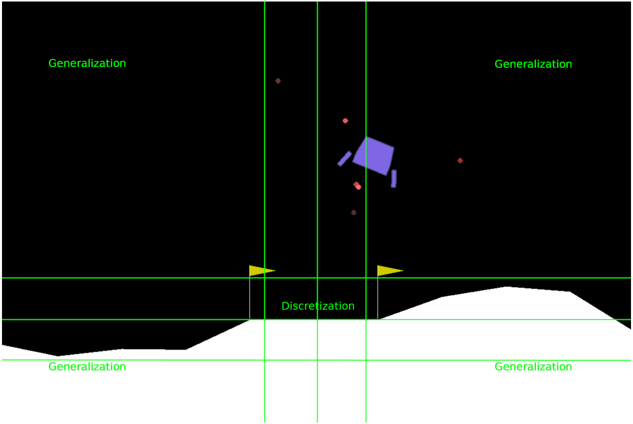


Figure: Example of discretization on the first two variables

Model-free setting

These are the algorithms used to tackle the problem at hand:

- Monte Carlo Control
- TD(0) Control: SARSA[1], Expected SARSA, Q-learning[1].
- TD(λ) Control: SARSA(λ), Q(λ)

All implemented methods utilize an $\varepsilon-$greedy policy.

[1] Implemented as a special case of TD(λ)

Monte Carlo Control

The implemented method is a first-visit MC. The current estimate of the action-value function is updated in the following way:

$$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t))$$

where $\text{Returns}(Q(S_t, A_t))$ stores the first-visit return of (S_t, A_t) for all the episodes generated so far.

Monte Carlo Control

The implemented method is a first-visit MC. The current estimate of the action-value function is updated in the following way:

$$Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t))$$

where $\text{Returns}(Q(S_t, A_t))$ stores the first-visit return of (S_t, A_t) for all the episodes generated so far.
Note that the update happens at the end of each episode.

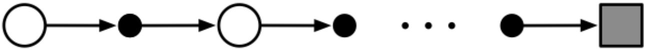


Figure: Backup diagram of MC Control. [2]

TD(0) Control

In these methods, the update of the action-value function happens at each time step.

TD(0) Control

In these methods, the update of the action-value function happens at each time step.
SARSA(0):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha_t[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

The update refers only to a single next state.

TD(0) Control

In these methods, the update of the action-value function happens at each time step.
Expected SARSA:

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

The update refers to all possible next states (at a single action distance).

TD(0) Control

In these methods, the update of the action-value function happens at each time step.

Q-learning:

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

The update refers only to the next state following a greedy policy. Note that this is not the actual method's policy. Q-learning is, in fact, *off-policy*.

TD(0) Control

In these methods, the update of the action-value function happens at each time step.

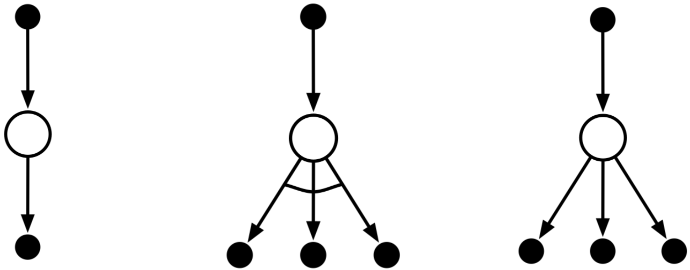


Figure: Backup diagrams of the TD(0) Control methods presented.[2]

TD(λ) Control

TD(λ) uses a return that includes *all* n-step returns, weighted by an exponentially decaying factor λ.
$TD(\lambda)$ is equivalent to $TD(0)$ if $\lambda = 0$ or to MC if $\lambda = 1$.

TD(λ) Control

TD(λ) uses a return that includes *all* n-step returns, weighted by an exponentially decaying factor λ.
$TD(\lambda)$ is equivalent to $TD(0)$ if $\lambda = 0$ or to MC if $\lambda = 1$.
Eligibility traces are an effective way to *approximate* this update:

$$e_t(S) = \begin{cases} \lambda\gamma e_t(S) & \text{if } S \neq S_t \\ \lambda\gamma e_t(S) + 1 & \text{if } S = S_t \end{cases}$$

Convergence

The learning rate α_t must satisfy:

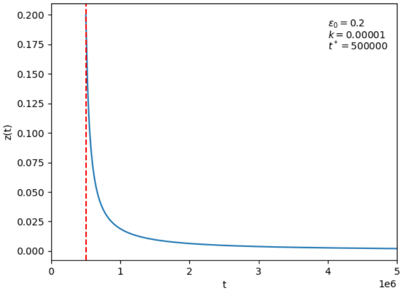
- $\sum_t^\infty \alpha_t^2$ converges
- $\sum_t^\infty \alpha_t$ diverges.

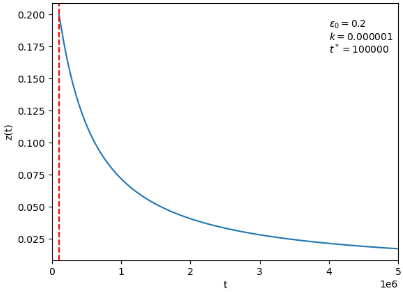
The same conditions must hold for ε_t. The chosen update function is the following:

$$z_t = \frac{z_0}{1 + k(t - t^*)^{0.75}}$$

where $z_t = \alpha_t$ or ε_t, $z_0 = \alpha_0$ or ε_0 and $k = k_\alpha$ or k_ε.
k determines the decreasing speed, z_0 the starting point and t^* the starting point of the decrease.

Convergence





Hyperparameters

t^*, γ and α_0 are fixed.
The following parameters will be tuned:

- ε_0: to decide *how greedy* the policy is at the beginning, due to the high amount of states.
- λ: to set the decay of the eligibility traces.
- k_α and k_ε. [2]

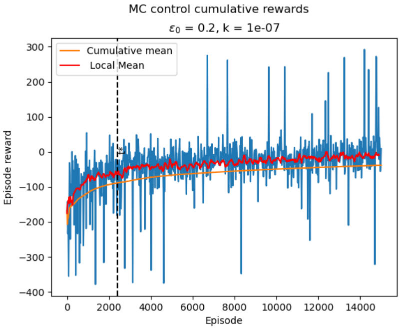
Hyperparameters

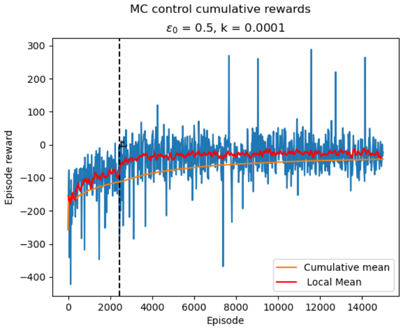
t^*, γ and α_0 are fixed.
The following parameters will be tuned:

- ε_0: to decide *how greedy* the policy is at the beginning, due to the high amount of states.
- λ: to set the decay of the eligibility traces.
- k_α and k_ε. [2]

Best parameters are chosen based on the greatest mean return of 2000 episodes *after training*, using a greedy policy.

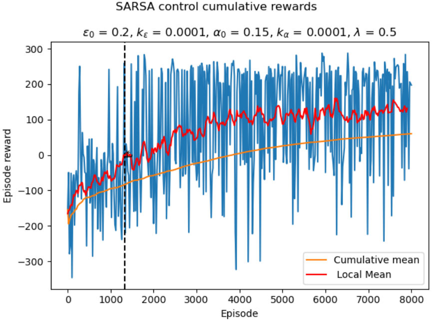
Results - MC Control

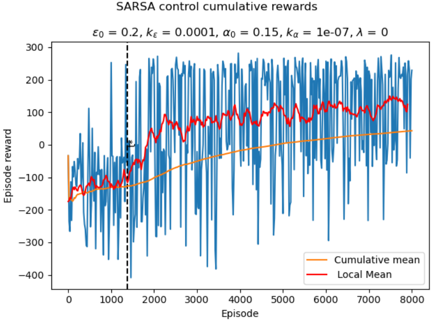




ε_0	k_ε	Mean	SD
0.2	0.0000001	33.333181	108.369116
0.5	0.0001	-24.237290	53.122977

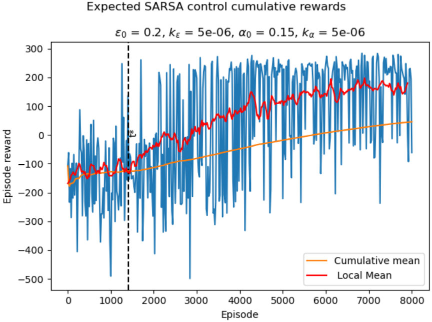
Results - SARSA

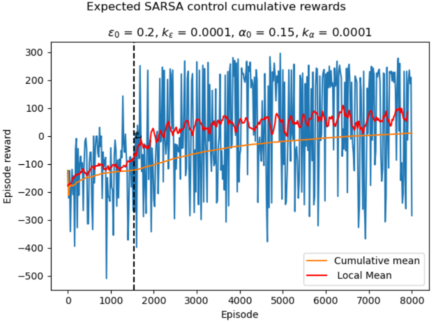




λ	k_α	k_ε	ε_0	Mean	SD
0.5	0.0001	0.0001	0.2	126.963339	146.207947
0	0.0000001	0.0001	0.2	120.464891	154.265832

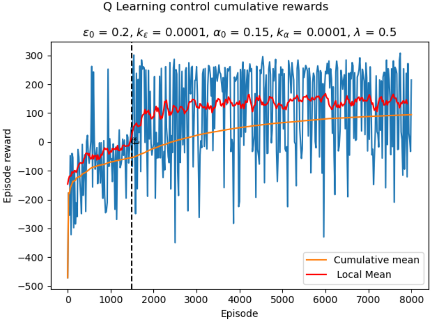
Results - Expected SARSA

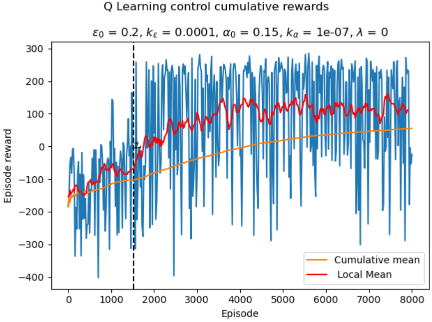




k_α	k_ε	ε_0	Mean	SD
0.000005	0.000005	0.2	163.010306	133.534936
0.0001	0.0001	0.2	87.824704	169.574288

Results - Q-learning





λ	k_α	k_ε	ε_0	Mean	SD
0.5	0.0001	0.0001	0.2	137.566266	147.619052
0	0.0000001	0.0001	0.2	119.539369	128.535647

Comparison between best results

λ	k_α	k_ε	ε_0	Mean	SD
-	0.000005	0.000005	0.2	163.010306	133.534936
0.5	0.0001	0.0001	0.2	137.566266	147.619052
0.5	0.0001	0.0001	0.2	126.963339	146.207947
-	-	0.0000001	0.2	33.333181	108.369116

References

[1] Soham Gadgil, Yunfeng Xin, and Chengzhe Xu. Solving the lunar lander problem under uncertainty using reinforcement learning, 2020.

[2] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction.

Note: the algorithms have been implemented with the course tutor's code as a starting point.