ULiège
Prof. Damien Ernst

INFO8003-1
Optimal decision making
for complex problems

Assignment 2

Reinforcement Learning in a Continuous Domain

Domain

We describe the domain below:

- State space: $X = \{(p, s) \in \mathbb{R}^2 | |p| \leq 1, |s| \leq 3\}$ and a *terminal state*[1].
  - A terminal state is reached if $|p_{t+1}| > 1$ or $|s_{t+1}| > 3$.
- Action space: $U = \{4, -4\}$.
- Dynamics: $\dot{p} = s$, $\dot{s} = \frac{u}{m(1+Hill'(p)^2)} - \frac{gHill'(p)}{1+Hill'(p)^2} - \frac{s^2 Hill'(p) Hill''(p)}{1+Hill'(p)^2}$, where $m = 1$, $g = 9.81$ and

$$Hill(p) = \begin{cases} p^2 + p & \text{if} \quad p < 0 \\ \frac{p}{\sqrt{1+5p^2}} & \text{otherwise.} \end{cases}$$

  - The discrete-time dynamics is obtained by discretizing the time with the time between t and $t+1$ chosen equal to $0.100s$.
- Integration time step: 0.001.
- Reward signal:

$$r(p_t, s_t, u_t) = \begin{cases} -1 & \text{if} \quad p_{t+1} < -1 \quad \text{or} \quad |s_{t+1}| > 3 \\ 1 & \text{if} \quad p_{t+1} > 1 \quad \text{and} \quad |s_{t+1}| \leq 3 \\ 0 & \text{otherwise.} \end{cases}$$

- Discount factor: $\gamma = 0.95$.

[1] A terminal state can be seen as a regular state in which the system is stuck and for which all the future rewards obtained in the aftermath are zero.

- Time horizon: $T \to +\infty$.
- Initial state: $p_0 \sim \mathcal{U}([-0.1, 0.1])$, $s_0 = 0$.

This domain is a *car on the hill* problem, and will be referred by this name from now on. Figure 1 shows an illustration of the domain.

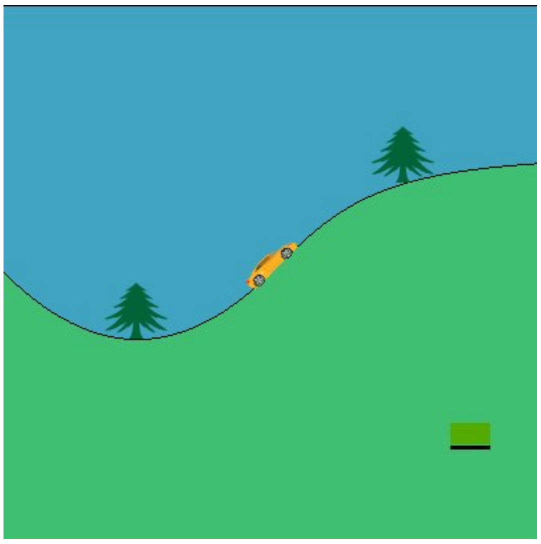

Figure 1: Display of the position $p = 0$ and the speed $s = 1$ of the car.

INSTRUCTIONS

You are expected to deliver (i) your source code (in Python 3.7, a file per section and named as *sectionK.py* where *K* is the section number - each missing one will cost you a one point penalty) and (ii) a report which is structured according to the next sections. We insist on the fact that the report is mandatory and that the source code needs to use only standard programming librairies plus, possibly, *NumPy*, *matplotlib* and those referred in the next sections. We should also be able to execute your code and understand easily the results displayed.

1 IMPLEMENTATION OF THE DOMAIN (2 POINTS)

Implement the different components of the *car on the hill* problem. Your implementation of the dynamics should exploit the Euler integration method using the integration time step specified in the domain. Make sure your implementation properly handles the terminal state case. Implement a rule-based policy of your choice (e.g., always accelerate, select actions always at random...) and describe it formally in your report. Simulate the policy in the domain from an initial state and display the trajectory as a sequence of tuples $(x_0, u_0, r_0, x_1), \ldots, (x_{10}, u_{10}, r_{10}, x_{11})$ (e.g., by displaying a screenshot from your favourite terminal).

2 Expected Return of a Policy in Continuous Domain (3 points)

Implement a routine which estimates the expected return of a policy for the car on the hill problem. Your routine should exploit the Monte Carlo principle. Choose a N which is large enough to approximate well the infinite time horizon of the policy and motivate your choice. Display, from 0 to N in a curve plot in your report, the expected return of your rule-based policy defined in Section 1 starting from 50 initial states drawn from the distribution specified in the domain statement.

3 Visualization (2 points)

Implement a routine which produces a video from any *car on the hill* trajectory. Your routine needs to produce the same output image as the function implemented in this *Python script* (or an equivalent implementation) which produces an image of a given state from the *car on the hill* problem. You may use your preferred Python programming library to generate your video file. Simulate the policy in the domain from the state ($s_0 = 0, p_0 = 0$) and save the video in a GIF file. Join it to your deliverable archive.

4 Fitted-Q-Iteration (7 points)

Implement a routine which computes $\widehat{Q}_N$ for $N = 1, 2, 3 \ldots$ using *Fitted-Q-Iteration*. Use the following supervised learning techniques:

- Linear regression,
- Extremely Randomized Trees,
- Neural networks.
  - You need to build yourself and motivate your neural network structure.

These techniques are implemented in the *scikit-learn* and *Keras* programming libraries. Propose two strategies for generating sets of one-step system transitions that will be used in your experiments and motivate them. Propose two stopping rules for the computation of the $\widehat{Q}_N$-functions sequence and motivate them. Display $\widehat{Q}_N$ in a colored 2D grid for each action (from red to blue as values increase, and with a resolution of 0.01 for the state space display) in your report. Derive the policy $\widehat{\mu}^*_N$ from $\widehat{Q}_N$ and display it in a colored 2D grid (red for action $u = -4$ and blue for action $u = 4$ and with a resolution of 0.01 for the state space display) in your report. Estimate and display the expected return of $\widehat{\mu}^*_N$ - in a table for each supervised learning algorithm, one-step system transitions generation strategy and stopping rule - in your report. Discuss the impact on the results of the supervised learning algorithm, the stopping rules and the one-step system transitions generation strategies.

5 Parametric Q-Learning (6 points)

Implement a routine which computes a parametrized approximation of the Q-function via the Parametric Q-Learning algorithm. Use a neural network as the approximation architecture, and motivate its structure. Derive the policy $\widehat{\mu}_*$ from $\widehat{Q}$ and display it in a colored 2D grid (red for action $u = -4$ and blue for action $u = 4$ and with a resolution of 0.01 for the state space display) in your report. Estimate and show the expected return of $\widehat{\mu}^*$ in your report. Design an experimental protocol to compare *FQI* and *parametric Q-learning* through a curve plot where the x-axis is the number of one-step system transitions and the y-axis is the expected return. Discuss the results obtained by running this experimental protocol.

Normalised Parametric Q-Learning (Bonus, 5 points)

Implement the online Q-iteration algorithm as seen in the lecture and run it through the experimental protocol you have designed in Section 5 with a normalised update term of the Q-function. Specify in the report the norm you have used. Discuss the impact of this normalised term on the performances of the online Q-iteration algorithm.
⚠This bonus is conditional on the general quality of your work.