ULiège
Prof. Damien Ernst

INFO8003-1
Optimal decision making
for complex problems

Assignment 1

Reinforcement Learning in a Discrete Domain

1 Domain

We describe the domain below:

- State space: $X = \{(x, y) \in \mathbb{N}^2 | x < n, y < m\}$,
- Action space: $U = \{(1, 0), (-1, 0), (0, 1), (0, -1)\}$,
- Dynamics: $(x_{t+1}, y_{t+1}) = (min(max(x_t + i, 0), n - 1), min(max(y_t + j, 0), m - 1))$ where $(i, j) \in U$,
- Reward signal: $r((x_t, y_t), (i_t, j_t)) = R(g, x_{t+1}, y_{t+1})$,
- Discount factor: $\gamma = 0.99$,
- Time horizon: $T \to +\infty$

where $g \in \mathbb{R}^{n \times m}$ for any $n, m \in \mathbb{N}$ and $R(g, i, j)$ returns the value of the cell located at (i, j) in g. Note that the reward signal is bounded by $B = \max_{(i,j)} R(g, i, j)$. The domain is deterministic. We also provide a stochastic setting which differs from the deterministic setting by its dynamics, redefined below:

$$(x_{t+1}, y_{t+1}) = \begin{cases} (min(max(x_t + i, 0), n - 1), min(max(y_t + j, 0), m - 1)) & \text{if} \quad w_t \leq 1 - \beta \\ (0, 0) & \text{otherwise,} \end{cases}$$

where $0 \leq w_t \leq 1$ is drawn according to a uniform distribution, for each time step $t = 0, 1, 2...T$ and β is chosen equal to 0.5 for this assignment.
Figure 1 describes an instance of the domain that you should use inside your experimental protocols.
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

-3	1	-5	0	19
6	3	8	9	10
5	-8	4	1	-8
6	-9	4	19	-5
-20	-17	-4	-3	9

Figure 1: Domain instance. The number in each cell represents the reward obtained while reaching it after a move (e.g., if at time t the agent moves up from the red cell, it receives a reward equal to 5). $(0, 0)$ corresponds to the upper-left corner of the grid. The two coordinates respectively refer to the line and to the column number. The initial state of this instance is the red cell located at $(3, 0)$.

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* and *matplotlib*. We should also be able to execute your code and understand easily the results displayed.

2 Implementation of the Domain (2 points)

Implement the different components of the domain. Implement a rule-based policy of your choice (e.g., always go left, select actions always at random...). Simulate the policy in the domain and display the trajectory.

3 Expected Return of a Policy (2 points)

Implement a routine to estimate J^μ in this domain, where $\mu : X \to U$ is a stationary policy. The implementation should exploit the dynamic programming principle where a sequence of functions J^N for $N = 0, 1, 2...$ is computed using the Bellman Equation. Test your implementation with your rule-based policy of Section 2. Choose a N which is large enough to approximate well the infinite time horizon of the policy and motivate your choice. Display $J^N_\mu(x)$ for each state x.

4 Optimal Policy (3 points)

Implement a routine generated from a random uniform policy which computes $p(x'|x, u)$ and $r(x, u)$ that define the equivalent MDP of the domain, together with γ and a trajectory of size T. Afterwards, implement a routine which computes $Q(x, u)$ from these components using the dynamic programming principle. Determine the smallest T such that a greater value does not change the policy inferred from Q. Derive directly μ^* from Q. Display $J^N_{\mu^*}$ for each state x.

5 System Identification (5 points)

Implement a routine which estimates $r(x, u)$ and $p(x'|x, u)$ from a given trajectory $h_t = (x_0, u_0, r_0, x_1, u_1, r_1, ..., u_{t-1}, r_{t-1}, x_t)$. Display the convergence speed towards p and r. Compute $\widehat{Q}$ by using $\widehat{r}(x, u)$ and $\widehat{p}(x'|x, u)$ that estimate the MDP structure. Derive $\widehat{\mu}^*$ from $\widehat{Q}$. Display $J^N_{\widehat{\mu}^*}$, along with $J^N_{\mu^*}$, for each state x.
Run your implementation over several trajectories, generated from a random uniform policy, of different lengths. Explain the influence of the length on the quality of the approximation $\hat{Q}$ using an infinite norm.

6 Q-learning in a Batch Setting (8 points)

6.1 Offline Q-learning (2 points)

Sample trajectories from a random uniform policy with a finite, large enough time horizon T. Implement a routine which iteratively updates $\widehat{Q}(x_k, u_k)$ with *Q-learning* from these trajectories. Run your routine with a constant learning rate $\alpha = 0.05$. Derive directly $\widehat{\mu}^*$ from $\widehat{Q}$. Display $J^N_{\widehat{\mu}^*}$, along with $J^N_{\mu^*}$, for each initial state x.

6.2 Online Q-learning (4 points)

Implement an intelligent agent which learns Q with Q-learning using an ϵ-greedy policy.

The first experimental protocol is the following. The agent trains over 100 episodes having 1000 transitions in the domain instance described in Figure 1. An episode always starts from the initial state (see Figure 1). The learning rate α is equal to 0.05 and the exploration rate ϵ is equal to 0.25. The values of α and ϵ are both constant over time. The function $\hat{Q}$ is updated after every transition. The transitions are used only once for updating $\hat{Q}$.

The second experimental protocol differs from the first one only by the learning rate which is such that $\alpha_0 = 0.05$ and $\forall t > 0, \alpha_t = 0.8\alpha_{t-1}$.

The third experimental protocol is identical from the first one except that the one-step system transitions are stored in a replay buffer and that at each time-step, the function $\hat{Q}$ is updated ten times by drawing ten transitions at random from the replay buffer.
Run the three experimental protocols and display for each of them, after each episode, the value of $\|\hat{Q} - J^N_{\mu^*}\|_\infty$. Compare the results obtained.

6.3 Discount Factor (2 points)

Rerun the first abovementioned experimental protocols with $\gamma = 0.4$. What do you observe in terms of convergence rate of the algorithm expressed as the speed at which $\hat{Q}$ converges to Q (in infinite norm)?

6.4 Q-Learning with Another Exploration Policy (Bonus, 2 points)

Implement another exploration/exploitation policy presented in the scientific literature. Rerun the first experience from 6.2 by replacing the ϵ-greedy policy with this new policy. Tune the parameters of your policy so as to to maximize the expected return over a single infinite time horizon. Discuss the results obtained at the light of the exploration/exploitation tradeoff.