Released: 4th Mar Deadline: 17th Mar

Problem Set 4

INF368A - Reinforcement Learning
Spring 2023

IMPORTANT: Submission of this problem set is **compulsory** to be admitted to the final exam and graduate from the course.

Part I. Problems

In the first part (*Problems*), you are required to provide answers for all the *obligatory* questions, that is, questions marked by a star (*). The remaining questions are optional; however, you are invited to solve them because: (i) they are stepping-stones towards the obligatory questions; (ii) they are similar to the questions you will be asked at the oral exam.

1. Function Approximation

Exercise1 Function Approximation

In class, we have seen that we can use *linear function* both for feature engineering $\phi(s)$ and for function approximation $f(\cdot; \mathbf{w})$.

1. (*) What is the difference between using a linear function for feature engineering and function approximation?
2. Write out the semi-gradient for learning a function approximation when performing prediction with a TD(n) target.
3. (*) Consider the *policy improvement theorem* that we discussed in the context of dynamic programming. Why do the guarantees of this theorem fail to apply in case of function approximation? What step of the proof does not hold anymore?

4. (*) Suppose you have been training the Mars Rover 2.0 robot using the tabular TD(λ) algorithm and relying on off-policy data gathered by the Mars Rover 1.0. A colleague suggests that to increase the performance of the Mars Rover 2.0 you should use a neural network to approximate its action-value function. What sort of challenge would you expect to face if you were to follow the suggestion of your colleague?

2. Planning

Exercise2 Model-Free and Model-Based Learning

Consider the following environment modelling a very simple game. An agent can be in two states $\mathcal{S} = \{\texttt{home}, \texttt{uni}\}$ and it has only two actions $\mathcal{S} = \{\texttt{travel}, \texttt{take\_exam}\}$. Episodes can start in any state and end after the agent undertakes the `take_exam` action.

The agent is run according to a policy π, and you are provided with the following trajectories in the format $(state_i, action_i, reward_i)$ for each timestep i.

$$\begin{aligned}
\tau_1 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\tau_2 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\tau_3 &= \{(\texttt{uni}, \texttt{take\_exam}, 0)\} \\
\tau_4 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\tau_5 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\tau_6 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\tau_7 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\tau_8 &= \{(\texttt{home}, \texttt{travel}, 0), (\texttt{uni}, \texttt{take\_exam}, 0)\}
\end{aligned}$$

We now want to run prediction for our policy π under an arbitrary discount γ.
(All computations are simple enough to be performed on paper; still, you can run them on your computer if you prefer.)

1. What would be $v_\pi(\texttt{uni})$ and $v_\pi(\texttt{home})$ computed using MC?
2. What would be $v_\pi(\texttt{uni})$ and $v_\pi(\texttt{home})$ computed using TD?
3. (*) Are the state values computed by MC and TD identical? If not, can you explain why not?

Let us now use, instead, the data provided to learn a model of our game.

4. What quantities do you need to have a full learned model $\hat{\mathcal{M}}$ of the game?
5. Recall the definition of the transition function $\mathcal{T}$.
6. Infer $\mathcal{T}$ from the trajectories using a table.
7. Recall the definition of the reward function $\mathcal{R}$.
8. Infer $\mathcal{R}$ from the trajectories using a table.
9. How does the reward function $\mathcal{R}$ differ from the value function $v_\pi(\cdot)$.

10. Draw the graph for the MDP $\hat{\mathcal{M}}$ you have learned.
11. Using the model $\hat{\mathcal{M}}$ that you have inferred, what would be $v_\pi(\texttt{uni})$ and $v_\pi(\texttt{home})$ computed using DP?
12. (*) How do the state-values you have learned with DP compare to the ones learned by MC and TD? Are they identical or not? If not, can you explain why not?

Suppose you use $\hat{\mathcal{M}}$ for simulation. The agent interacts with $\hat{\mathcal{M}}$ following the same policy π, and generates the following trajectories

$$\begin{aligned}
\hat{\tau}_1 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\hat{\tau}_2 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\hat{\tau}_3 &= \{(\texttt{home}, \texttt{travel}, 0), (\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\hat{\tau}_4 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\hat{\tau}_5 &= \{(\texttt{uni}, \texttt{take\_exam}, 0)\} \\
\hat{\tau}_6 &= \{(\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\hat{\tau}_7 &= \{(\texttt{home}, \texttt{travel}, 0), (\texttt{uni}, \texttt{take\_exam}, 1)\} \\
\hat{\tau}_8 &= \{(\texttt{home}, \texttt{travel}, 0), (\texttt{uni}, \texttt{take\_exam}, 0)\}
\end{aligned}$$

13. Compute again $v_\pi(\texttt{uni})$ and $v_\pi(\texttt{home})$ using MC from real and simulated data.
14. Compute again $v_\pi(\texttt{uni})$ and $v_\pi(\texttt{home})$ using TD from real and simulated data.
15. (*) Did the values computed by MC and TD change? If so, how did they change and why?

Exercise3 Monte Carlo Tree Search

1. (*) Suppose you are training an agent to drive a self-driving car. A colleague, who has recently read about Monte Carlo Tree Search, suggests that the implementation of MCTS in your problem could improve the quality of the driving. What do you think of this suggestion? What considerations would you need to make to decide whether the use of MCTS is justified or not?

Part II. Report

In the second part (*Report*), you are required to provide a report of maximum 2 pages describing the experiments you have run and analyzed to answer the *obligatory* questions, that is, questions marked by a star (*). The remaining questions are optional; however, you are invited to solve them because: (i) they are stepping-stones towards the obligatory questions; (ii) they are similar to the questions you will be asked at the oral exam. You will be evaluated only on your report. Review the documents on `coding tips` and `writing tips` for advices on how to tackle this part.

Exercise4 Feature Engineering

Consider the `Mountain-Cart` environment from the *gymnasium* library. We will run episodes of this environment for 1000 steps.

The state space of this problem is continuous[1]. This hinders the possibility of using algorithms you have already developed. Let us consider an approach based on *feature engineering*.

1. Choose a uniform discretization step Δ for the states that you observe. Use it to preprocess the continuous states into discrete states and try to learn an optimal policy using SARSA or Q-learning.
2. (*) Consider a range of possible discretization steps Δ and compare the result that you obtain using SARSA or Q-learning for control.
3. (*) Is there any other form of feature engineering you would consider applying to this problem?

Exercise5 Planning

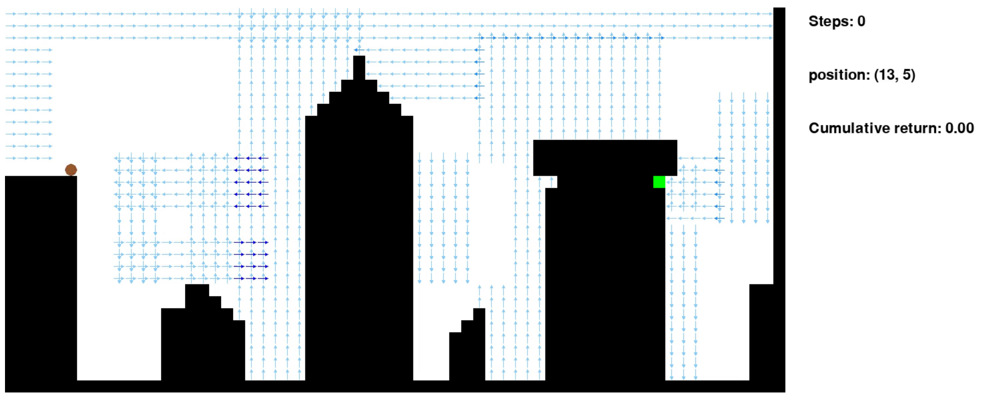
Consider a lightweight drone. You have been charged with developing a policy that would allow the drone to glide safely in the eventuality that its battery would run low. To do so, the drone has to exploit air currents in order to reach a landing zone.

We summarize below facts about the dynamics of the environment (see picture):

- *Objective:* the aim of the drone is to reach the landing zone on the top floor of the rightmost building (green square in position $(14, 54)$).
- *Representation:* the environment in which the drone can move is represented by tuples (i, j) indexing row and column of the two-dimensional environment.
- *Start state:* the drone starts from the top of the leftmost building (brown circle in position $(13, 5)$).

[1]Theoretically continuous. Its implementation is of course discrete, but we will treat it as continuous.

- *End state:* an episode ends when the drone reaches the landing zone on the top floor of the rightmost building (green square in position $(14, 54)$), or whenever colliding with an obstacle or leaving the game area.
- *Actions:* the drone has two actions: increasing its leftward speed (-3), or increase its rightward speed $(+3)$.
- *Transitions:* transitions are determined by three factors: the horizontal speed of the drone, the gravity which pulls the drone downward with constant speed (1), and air currents (blue arrows) that will push the drone in different directions.
- *Reward:* a reward of 1 is provided to the drone upon reaching the landing site; otherwise, reward is 0.
- *Violation of borders:* if the drone attempts to move beyond the map it will automatically crash.



While battery power was available, your colleagues have flown the drone across the windy skyscrapers of the city, from the starting pad to the landing position. In this way, they have been able to collect information about the environment and its wind currents. Data have been stored in the file `power_flight.txt` in the format $(i, j, a, r, i'.j')$, where (i, j) is the current state, a the action, r the reward, and (i', j') the next state. (**Caution: notice that the data in the file is 1-indexed, while the environment is 0-indexed.**)

1. Use the data collected to develop a model $\hat{\mathcal{M}}$ of the environment.
2. Implement the Dyna architecture and learn an optimal policy.
3. Plot the optimal trajectory you have learned.
4. (*) Run Dyna changing the number of simulations run by the agent using $\hat{\mathcal{M}}$, and compare the results.