October 16, 2023

Department of Computing
Department of Bioengineering

Reinforcement Learning – Prof Aldo Faisal and Dr Nicole Salomons
Coursework design: Prof Aldo Faisal & Manon Flageat

Assessed Coursework 1

The Coursework should be submitted on Scientia by November 3, 10 am (this date and time ensures that students have an equal number of computer labs), and consist in:
- A **PDF** of your written report, imperatively named ***coursework1_report.pdf***.
- A **Python file** of the final code used to generate your results, based on the provided code, imperatively named ***coursework1.py***.
The Scientia submission will open on October 18th. Please ensure that you are familiar with it well before the deadline. Unfortunately, emailed or printed submissions cannot be accepted.

Report: Your report should not be longer than 7 single-sided A4 pages with at least 2-centimeter margins all around and 12pt font, preferably typeset, ideally in Latex. Appendixes are not allowed. 7 pages is a **maximum** length, shorter Courseworks are fine, and Courseworks over the page limit will incur a penalty. Questions that should be answered in the Report are indicated in the following as **[Report]**, questions indicated as **[Notebook]** do not need to be answered in the report but only code in the Jupyter Notebook. Submissions that violate these requirements will receive up to a 20-mark penalty.

Code: Your code should follow the guidelines detailed below to ensure it can be automatically marked. **DO NOT** add any non-optional inputs to the functions and methods to ensure your code can be automatically marked. Make sure that the functions and agent classes you write are fully self-contained (i.e. they don't rely on global variables) and only access external class attributes that you are explicitly told you can access in the question. You should only modify those classes and functions that you are explicitly told to modify in the specifications. You are welcome to reuse code you developed in the lab assignments and interactive computer labs, but your code should be your own. When handing in your code, please keep the code you write to run experiments or generate plots within a conditional statement `if __name__ == '__main__':` so that experiments are not ran and plots are not generated when the code is imported by the automated code marking scripts. All object and function definitions that involved in a **[Notebook]** tag should come before this condition so that they can be imported by the automatic marking scripts. Before handing in your code, ensure that importing your ***coursework1.py*** script does not run lengthy experiments (e.g. hyperparameter experiments) or any plotting code.

You are encouraged to discuss the general Coursework questions with other students, but your answers should be yours, i.e., written by you, in your own words, showing your own understanding. Your report and code will be automatically verified for plagiarism. Written answers should be clear, complete and concise. Figures should be clearly readable, labelled, captioned and visible. Poorly produced answers, incomplete figures, irrelevant text not addressing the point and unclear text may lose points.

Marks are shown next to each question. Note, that the **marks are only indicative** and the maximum marks achievable are 70 pts. Please note that this coursework counts 25% of your final grade, and so carries a substantially higher grade than on regular coursework – which is why we can offer a light-touch exam at the end of term. If you have questions about the coursework please make use of the computer labs or EdStem. Please note that the Graduate Teaching Assistants cannot provide you with answers that directly solve the coursework but are very happy to help you with questions.

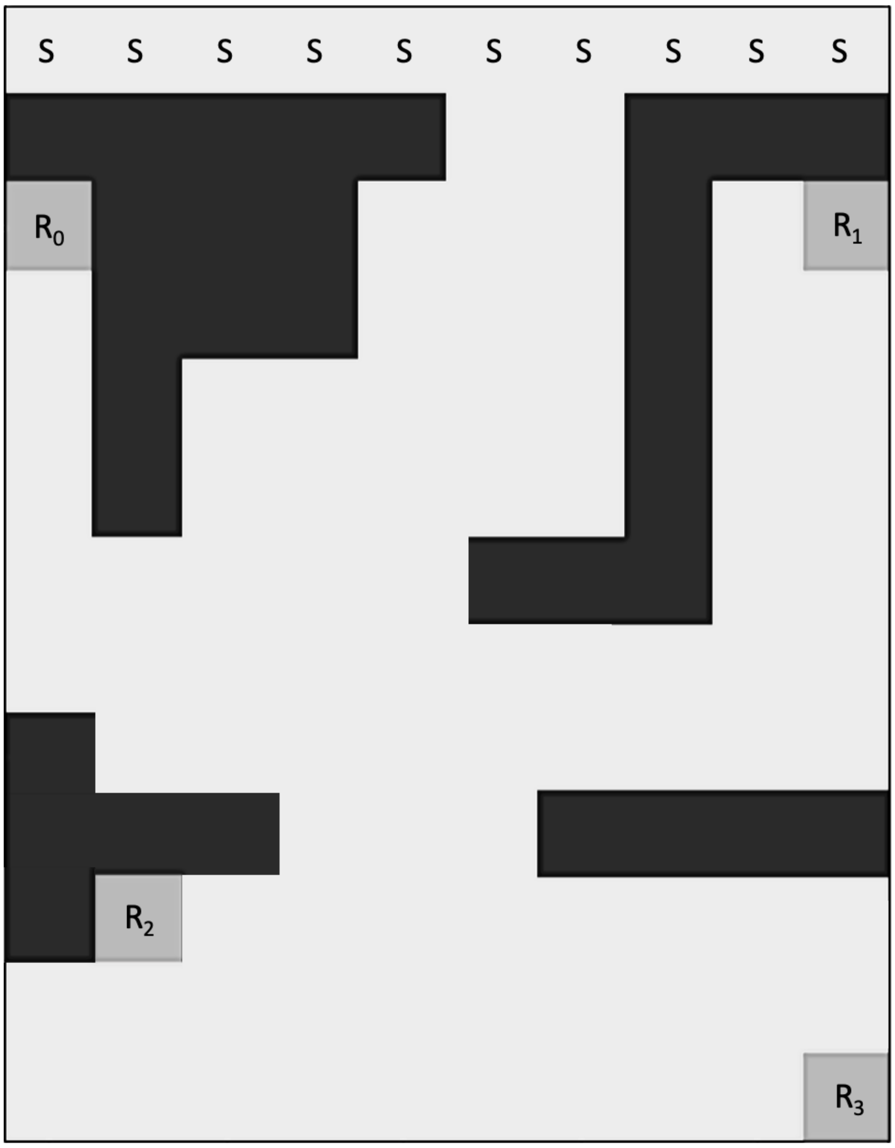

Figure 1: Illustration of the Maze environment, the focus of this Coursework. The absorbing states are indicated as "R_i", they correspond to absorbing rewards. The "S" indicate the potential start states.

Description of the Maze environment

The focus of this Coursework is the resolution of a Maze environment, illustrated in Figure 1 and model as a Markov Decision Process (MDP). In this illustration, the black squares symbolise obstacles, and the dark-grey squares absorbing states, that correspond to specific rewards. Absorbing states are terminal states, there is no transition from an absorbing state to any other state.

This environment shares a lot of similarities with the Grid World environment studied in Tutorial 2, you are allowed to reuse any code you might have developed for this tutorial. However, unlike the Tutorial, this Coursework aims to help you implement RL techniques that do not require full knowledge of the dynamic of the environment, namely Monte-Carlo and Temporal-Difference learning. In the following, we will thus assume we don't have access to the transition matrix T and reward function R of the environment, and try to find the optimal policy π by sampling episodes in it.

Consider an agent moving inside this maze, trying to get the best possible reward. To do so, it can choose at any time-step between four actions:

- $a_0 =$ *going north* of its current state
- $a_1 =$ *going east* of its current state
- $a_2 =$ *going south* of its current state
- $a_3 =$ *going west* of its current state

The chosen action has a probability p to succeed and lead to the expected direction. If it fails, it has equal probability to lead to any other direction. For example, if the agent chooses the action $a_0 =$ *going north*, it is going to succeed and go north with probability p, and it has probability $\frac{1-p}{3}$ to go east, probability $\frac{1-p}{3}$ to go south and probability $\frac{1-p}{3}$ to go west. p is given as a hyperparameter, defining the environment.

If the outcome of an action leads the agent to a wall or an obstacle, its effect is to keep the agent where it is. For example, if the agent chooses the action $a_1 =$ *going east* and there is a wall east but no wall in the other directions, it is going to stay in place in his current state with probability p and to go north, south or west with the same probability $\frac{1-p}{3}$.

Any action performs by the agent in the environment gives it a reward of -1.

The episode ends when the agent reach one of the absorbing state $(R_i)_{0 \leq i \leq 3}$ or if it performs more than 500 steps in the environment.

Finally, the starting state is chosen randomly at the beginning of each episode from the possible start states with equal probability $1/N_{start-states}$. The possible start states are indicated on the Maze illustration as "S".

This environment is personalised by your College ID (CID) number, specifically the last 2 digits (which we refer to as y and z), as follow:

- The probability p is set as $p = 0.8 + 0.02 \times (9 - y)$.
- The discount factor γ is set as $\gamma = 0.8 + 0.02 \times y$.
- The reward state is R_i with $i = z \ mod \ 4$ (meaning that if $z = 0$ or $z = 4$ or $z = 8$, the reward state would be R_0, if $z = 1$ or $z = 5$ or $z = 9$, it would be R_1, etc).
- The reward state R_i gives a reward $r = 500$ and all the other absorbing states gives a reward $r = -50$. As stated before, any action performed by the agent in the environment also gives it a reward of -1.

Jupyter Notebook

This Coursework is based on the provided Jupyter Notebook, also available on Colab: `https://colab.research.google.com/drive/1G3KJGCqR13NBt4F7oH5s6KYFS9UBigD1?usp=sharing`. It defines most of the Maze structure. In the following questions, you will be asked to progressively complete the functions of the Maze class indicated with the tag "[Action required]".

The Notebook has the following structure:

1. `get_CID()` and `get_login()` functions definition. These functions should return your CID and short login. They are used by the auto-marking script, make sure to fill them in.
2. `GraphicsMaze` class definition: allow all the graphical visualisation of the Maze. You **DO NOT NEED** to read or understand this class, you can simply call its methods when needed.

3. `Maze` class definition: this class is the main Maze class, you will be asked to complete it in the following questions. All attributes and methods of this class starting with `_` such as `env._T` or `env._build_maze()` are private and should NOT be accessed outside the `Maze` class. `Get` functions such as `get_T()` are defined to access some of them from outside the class.
4. `DP_agent` class definition: this class defined a Dynamic Programming agent to solve the Maze, you will be asked to complete it in the following questions.
5. `MC_agent` class definition: this class defined a Monte-Carlo Learning agent to solve the Maze, you will be asked to complete it in the following questions.
6. `TD_agent` class definition: this class defined a Temporal-Difference Learning agent to solve the Maze, you will be asked to complete it in the following questions.

Question 0: Defining the environment – 2 pts

We are first going to define the Maze environment used in the following. This question is only code-based and does not need to be part of your Report.

[Notebook] – 2 pts

Fill in the following parts of the Jupyter Notebook:

- The functions `get_CID()` and `get_login()` to return your CID number and you short login. Make sure you fill in these functions as they will be used by the auto-marking script.
- The `__init__()` method of the `Maze` class, to set you personalised reward state, absorbing reward, p, and γ in the environment.

Question 1: Dynamic programming – 18 pts

For comparison purposes, we are first going to assume we have access to the transition T and reward function R of this environment and consider a Dynamic Programming agent to solve this problem.

[Notebook] – 4 pts

Complete the `solve()` method of the `DP_agent` class. You can choose any Dynamic Programming method and you are allowed to reuse code from the lab assignment. Note that *for this agent only*, you are allowed to access the transition matrix through `env.get_T()`, the reward matrix through `env.get_R()` and the list of absorbing states through `env.get_absorbing()` of the `Maze` class. You might also need to use `env.get_action_size()`, `env.get_state_size()` and `env.get_gamma()`.

You can define any additional methods inside the `DP_agent` class, but only the `solve()` method will be called for automatic marking.**DO NOT** add any inputs or outputs to `solve()`.

[Report] – 14 pts

1. State which method you choose to solve the problem and justify your choice. Give and justify any parameters that you set, design that you chose or assumptions you made. **– 4 pts**
2. Report the graphical representation of your optimal policy and optimal value function. **– 6 pts**
3. Discuss how the value of your γ and p have influenced the optimal value function and optimal policy in your personal Maze. In particular, you may investigate the effect of having a value of $p < 0.25$, $p = 0.25$ or $p > 0.25$, and similarly $\gamma < 0.5$ or $\gamma > 0.5$. **– 4 pts**

Question 2: Monte-Carlo Reinforcement Learning – 25 pts

We are now assuming that we do not have access to the transition T and reward function R of this environment. We want to solve this problem using a Monte-Carlo learning agent.

[Notebook] – 5 pts

Complete the `solve()` method of the `MC_agent` class. Note that for this agent you are only allowed to use the `env.reset()` and `env.step()` methods of the `Maze` class, as well as `env.get_action_size()`, `env.get_state_size()` and `env.get_gamma()`. **DO NOT** use the transition matrix `env.get_T()`, the reward matrix `env.get_R()` and the list of absorbing states `env.get_absorbing()`, or any other `env` attribute not mentioned above.

You can define any additional methods inside the `MC_agent` class, but only the `solve()` method will be called for automatic marking. **DO NOT** add any inputs or outputs to `solve()`.

[Report] – 20 pts

1. State which method you choose to solve the problem and justify your choice. Give and justify any parameters that you set, design that you chose or assumptions you made. **– 8 pts**
2. Report the graphical representation of your optimal policy and optimal value function. **– 6 pts**
3. Plot the learning curve of your agent: the total non-discounted sum of reward against the number of episodes. The figure should picture the mean and standard deviation across your replications on the same graph (using the replication number you stated in the previous question). **– 6 pts**

Question 3: Temporal Difference Reinforcement Learning – 25 pts

We are now going to assume we do not have access to the transition T and reward function R of this environment. We want to solve this problem using a Temporal-Difference learning agent.

[Notebook] – 5 pts

Complete the `solve()` method of the `TD_agent` class. Note that for this agent you are only allowed to use the `env.reset()` and `env.step()` methods of the `Maze` class, as well as `env.get_action_size()`, `env.get_state_size()` and `env.get_gamma()`. **DO NOT** use the transition matrix `env.get_T()`, the reward matrix `env.get_R()` and the list of absorbing states `env.get_absorbing()`, or any other `env` attribute not mentioned above.

You can define any additional methods inside the `TD_agent` class, but only the `solve()` method will be called for automatic marking. **DO NOT** add any inputs or outputs to `solve()`.

[Report] – 20 pts

1. State which method you choose to solve the problem and justify your choice. Give and justify any parameters that you set, design that you chose or assumptions you made. **– 8 pts**
2. Report the graphical representation of your optimal policy and optimal value function. **– 6 pts**
3. How does varying the exploration parameter ϵ and the learning rate α of your algorithm impact your learning curves? Briefly explain what you find and relate it where possible to the theory you learned. We encouraged you to use relevant additional plots to illustrate your findings in this question. **– 6 pts**