Assignment 3: Reinforcement Learning

March 20, 2023

0. INTRODUCTION

The overall aim of the 3 assignments of the course is to create a robot that can do groceries by itself. For this, it is important that your robot can learn how to optimally navigate the supermarket, for example, to a product or to the cash register.

In this assignment, you will help the robot learn such a route based on rewards. The supermarket will be modeled as a labyrinth (further referred to as "maze") where the target location yields a reward. Your robot will learn to navigate from a fixed starting point to this rewarded target through many rounds of exploration. We take an approach based on Reinforcement Learning (RL): **you will implement Q-learning**, a model-free off-policy version of RL. To that end, you will use a tabular representation for $Q(s,a)$ - a matrix with two dimensions S and A, where S represents all possible states (locations in the maze), and A represents all actions possible in each state (`up`, `down`, `left`, `right`). A value in the matrix at position s,a represents the value $Q(s,a)$, i.e. the value of an action a in state s.

The problem that you need to solve boils down to learning an $S \times A$ matrix of values, each value representing the utility of an s,a pair. Every step taken by your robot updates *one* such value, namely the last visited s,a pair. As we use Q-learning (an off-policy where updates are not based on the actually executed actions), we will update this value based on the value of the best possible next action $Q(s', a_{max})$. The update rule that you will implement is:

$Q(s,a)_{new} = Q(s,a)_{old} + \alpha(r + \gamma Q(s', a_{max}) - Q(s,a)_{old})$,
where $Q(s', a_{max})$ is the Q value of the best action so far in state s'.

Your robot will learn by updating $Q(s,a)$ each time a random action is taken. You will have to experiment with the actual action selection algorithm (the policy). For these experiments you will implement the ϵ-Greedy method, which either takes a random action with a probability

of ϵ (exploration) or a greedy action with a probability of $1-\epsilon$ (exploitation). A random action is selected out of the 4 possible actions. A greedy action is an action with the highest predicted value so far, so the action with the highest $Q(s,a)$ for the state s we are now in. By varying the ϵ you can thus control how much the robot explores. By varying the α and γ you can control the speed of learning and the discount factor respectively.

ASSIGNMENT

You and your team will have to write a fully functioning Q-learning algorithm with ϵ-Greedy action selection. For this, we have provided you with a Python template (see Appendix A), you only need to focus on the actual algorithms of action selection and learning. You will also have to write a report explaining what you did. Please include answers to all the questions from Section 1 in a structured fashion (number your answers to the corresponding questions). **Whenever you make a design decision, motivate your choice and where possible, include plots and graphics to support your answers**.

You will conduct your experiments primarily on the `toy_maze.txt` provided with this exercise. Your robot's starting location is fixed at the top left of the maze; your target location is fixed at the bottom right of the maze and has a reward of 10. Start and target locations in the `easy_maze.txt` are also in the top left and bottom right corners respectively.

DELIVERABLES

Create a report to answer the questions from this document. Besides this, you have to clean up your code and **upload its final working version to the course Vocareum** in the corresponding assignment. Your submission must include **a Jupyter Notebook** which starts the training sequence. You should check if the notebook works as expected before pressing the `Submit` button. Altogether, the following files should be delivered:

- all of your code files;
- a Jupyter Notebook which starts the training sequence;
- a report in PDF format with answers to all questions. Your report must be no longer than 8 pages (including visuals) + the title page and references. Please structure your report based on the numbered questions in this document:

  `"<group_number>_report.pdf"`

Note 1: Your code will not be directly graded - do not rely on it to support your answers. Nonetheless, as the code will be inspected for irregularities remember to keep your code clean, use proper indentation, provide useful comments, name your variables logically, etc.

Note 2: To implement your algorithm you may only use Python with `NumPy` and `pandas`.

We want you to deliver your work exactly as outlined above. **If (a) your files are not of this format or (b) the main notebook does not run, your work will not be graded.**

Fraud will not be tolerated. You are allowed to discuss concepts and ideas with colleagues from other groups, but you are not allowed to share code outside your own group. The same applies to submissions from previous years. You are highly encouraged to make use of the provided Gitlab repositories. You may ask questions on Answers-EWI (while keeping in mind that it's a public forum, so please don't share partial solutions), or to a TA during the labs.

The deadline for the assignment is Tuesday **11$^{\text{th}}$ of April 2023 at 18:00**.

1. QUESTIONS

The following section will guide you in the implementation of your *Q-learning* algorithm. Give answers to all of the questions in a structured manner in your report.

1.1. DEVELOPMENT

In this sections you will implement the algorithm step by step.

1. (1 point) Ensure your agent can move in the maze. Implement `get_random_action()`, `get_best_action()`, and `get_egreedy_action()` in `MyEGreedy`. How do you prevent the agent from being biased to select the same action in `get_best_action()` over and over before it has learned something about the environment?

2. (1 point) Implement the agent's cycle in `Main` (selecting actions, executing them, updating the Q-table, ...). Note that the agent will not be able to learn, as you have not implemented `update_q()` in `MyQLearning` yet. Remember that you need to reset the agent when it reaches it's goal location. Explain the cycle in your report.

3. (0 points) Implement the stopping criterion at 30000 steps. Verify if your agent stops correctly. If you wish to conduct your experiments on easy maze, it may require more steps because the maze is larger.

4. (2 points) Run your agent. It should print to a list of numbers representing the steps taken in a single episode (in other words a trial, the period between two resets). You should be able to observe that the number of steps the agent takes per episode does not decrease. Make a plot of the average of 10 runs (one run is one launch of `Main`) to show that the agent indeed *does not* learn. The x-axis should represent the consecutive numbers of the episodes, the y-axis should represent the average number of steps in each episode. Note that you might not have the same number of episode per run if your stopping criterion does not include the number of episodes per run. Make this plot for both mazes given. Explain your plots.

5. (1 point) Implement `update_q()` in `MyQLearning`. Set $\alpha = 0.7$, $\gamma = 0.9$ and $\epsilon = 0.1$. This should be a basic implementation of Q-Learning. Explain your method in the report.

6. (2 point) Now run your agent again on the mazes. You should be able to see that the numbers decrease over time during a run. If you don't, first verify if you allow the robot to try long enough. If it still does not learn, make sure you have set α, γ and ϵ to the default values. Again, make a plot of the average of 10 runs to show that the agent indeed *does* learn. Make this plot for both mazes given. Explain your plots.

1.2. TRAINING

In this section you will experiment with the action selection and training.

7. (2 points) Play around with the ϵ parameter of your action selection algorithm. Choose a value of ϵ between 0 and 1 at least four times. For each of those values, make a plot of 10 run averages (similar as above) to study the effect of varying epsilon. Explain what you see. Be precise in your observations.
8. (1 point) What are the trade-offs between a high and a low ϵ?
9. (2 points) Now, play around with the learning rate α of your algorithm. Again, choose a value of α between 0 and 1 at least four times. Make a plot for each of these values. Explain what you notice. Why is that the case?

1.3. OPTIMIZATION

In this section you will optimize the γ and ϵ. **Do the following exercises on the toy maze**.

10. (1 point) Add a second reward, sized 5, at the location `(9,0)` (top right). Remember to add this location to the condition to reset the agent (it is a second goal). Run your program again a couple of times. What do you observe? How can you explain this?
11. (2 points) Invent a way to mitigate the previous problem by modifying the values of ϵ (they can change with each episode). Explain your method. Show with a plot that your agent converges to the optimal solution.
12. (1 point) Can you, using your previous solution, experimentally find a value of γ for which the optimal solution is in fact to go up for the smaller reward, rather than go down for the larger one? Explain and show the plots for different values of γ.

1.4. REFLECTION

For these questions, you should not think about your specific implementation. Instead, we ask you to consider Reinforcement Learning as a general technique.

13. (1 point) Can you think of possible downsides of having an algorithm that is greedy with regard to its action selection?
14. (1 point) When and why can reward functions cause problems in the society? This question is *not* about difficulty of finding a good reward function or discrimination/bias but rather about agents "understanding" the point of a reward function.
15. (2 points) Can you think of a way to overcome these problems for all reward functions? If not, why do you think that the problem is unsolvable?

1.5. PEN AND PAPER

Finally, we have three questions for you to be solved on paper which serve as the preparation for the final exam. They are independent from the rest of the questions in this document. You are asked to submit only one set of answers but we strongly suggest that each team member solves them separately, so that you all can practice with the calculations. For all questions use the problem description below. If you make any assumptions, explain them.

- You would like to have some fruit for dessert, so you send your robot to the store.
- The store consists of three locations: `entrance`, `aisle`, and `checkout`.
- The robot must navigate between the locations in the given order (meaning that it cannot move directly between `entrance` and `checkout`) using actions `forward` and `back`.
- Whenever your robot tries to move (`forward` or `back`), there is a 10% probability that its actuators fail and the robot stays in the same location suffering a penalty of -1.
- In the aisle, the robot may choose to `pick up apples` with 100% success rate.
- It may also choose to `pick up berries` (up to two boxes). When the robot picks up the first box of berries, the action always succeeds. If the robot is already carrying one box of berries, it can try to pick up another box of berries with 80% success rate.
- If picking up fails, the robot suffers a penalty of -2 for spilling the fruit.
- The robot **cannot** try to pick up both apples and berries in the same episode.
- Whenever the robot is in the `aisle`, it can decide to move to the checkout and `pay` for the groceries it is holding; the payment always succeeds.
- After payment, the problem terminates.
- If the robot pays while holding apples, it receives a reward of $+5$. If it pays while holding one box of berries, it receives a reward of $+3$. If it pays while holding two boxes of berries, it receives a reward of $+10$. Otherwise the reward is 0.
- There are no other action effects. For example, performing the `pick up berries` action at the entrance means that the state will not change. If the robot performs an action in the state where it does not have any effects, it suffers a penalty of -1.
- All possible actions, locations, and rewards are specified above.
- The problem begins at the `entrance` when the robot is not holding anything.

The questions which you must answered are given on the next page.

16. (2 points) Draw a graph representing the transitions of the MDP for the story above. The nodes represent states, the arrows **annotated with actions and probabilities** represent the transitions between the states. Actions that lead to no change in the state of the robot can be omitted to avoid clutter. Make sure to name all states.

17. (3 points) Apply the Bellman equation as given below

$$v_{k+1}(s) := \sum_{s'} P(s'|s,\pi(s)) \left[R(s,\pi(s),s') + \gamma v_k(s') \right]$$

to determine the $k = 2$-steps-to-go value function $v_2(s)$ for the following policy:

- If the robot is holding two packs of berries, move to the checkout and pay.
- If the robot is **not** holding two packs of berries, do `pick up berries`.

In your calculations use a discount factor of $\gamma = 0.9$. Make sure to **first write down the tabular representation of the policy**.

18. (1 point) Explain how we could determine the optimal policy for the robot. In other words, how do we decide whether the robot should get a bag of apples, one box of berries, or try to pick up two boxes of berries? You **should not** perform any calculations.

A. INSTRUCTIONS FOR THE TEMPLATE

The project consists of 6 predefined classes:

- `Action` - encodes the possible actions.
- `Agent` - maintains the current coordinates (`x` and `y`) of the robot. Contains the methods to (1) retrieve the current state of the agent (`get_state(...)`), and (2) set the robot back to its starting location which is fixed when the agent is created. This will also print the number of steps taken since the previous reset into the console (`reset()`). Finally, (3) it allows you to execute a selected action on the maze (`do_action(...)`).
- `Maze` - allows you to load a maze from a file by using its constructor which parses a text file and constructs a 2-dimensional maze. This class additionally has helper functions to (1) retrieve valid actions (`get_valid_actions(...)`, (2) set reward for a particular state (`set_reward(...)`, and (3) retrieve the reward for a state (`get_reward(...)`).
- `State` - encodes the information about each state.
- `QLearning` - contains methods to (1) get a Q-value for a state-action pair (`get_q(...)`), (2) return values associated with specified actions (`get_action_values(...)`), and (3) set the Q-value for a state-action pair (`set_q(...)`).

There are further three files that you have to extend with your implementation:

- `MyEGreedy` - it should contain all functionalities related to the ϵ-Greedy algorithm:
  - `get_random_action(...)` – to select an action at random in a given state
  - `get_best_action(...)` – to select the best action currently known in a state
  - `get_egreedy_action(...)` – to select between random or best action
- `MyQLearning` - inherits all functionality from `QLearning`. It should contain method `update_q(...)` to update the Q-values according to the Q-learning algorithm.
- `Main` - should implement the agent cycle.

You only need to program in these three files for the assignment. You do not need to make changes to the other files or to modify the function signatures.