SOW-BKI258 Reinforcement Learning Assignment

Contents

1 Introduction

To apply and broaden the knowledge acquired during this course, there will be a project consisting of weekly parts. Each week, new material is covered in the lecture. Alongside these lectures, there are practical sessions in which you will work on implementing and furthering your understanding of the material for that week. The project is an accumulation of the weekly work you will do during the practical session.

Throughout the course, you will choose or create a tabular Reinforcement Learning (RL) environment (see Section 4), implement various RL algorithms for this environment (see Section 2), and write a report comparing the RL algorithms (see Section 3).

The **deadline** for handing in the project is set at **March 28th, 23:59**. Only one group member needs to hand in the project.

You will be working on these projects in **groups of 4**. Enrollment in these groups is done through Brightspace: go to the course page (2425 Reinforcement Learning PER 3 V) → Administration → Groups

There is a Grading Rubric available which details the contents of the project, but it should at least contain the following:

- A report (1000-1500 words) (see Section 3 for more details)
- A Python file or Jupyter Notebook containing:
  - The realization of your environment
  - The implementations of the RL algorithms*:
    1. Dynamic Programming: Policy Evaluation, Policy Improvement, Value Iteration
    2. Monte Carlo methods: Monte Carlo prediction, Monte Carlo control
    3. Temporal Difference Learning: TD(0), SARSA, Q-Learning
    4. *And also a random (baseline) agent

2 Roadmap

To guide you through project, we have set up a road map. Each week, a different topic is covered in the lecture and this road map provides a short overview of how we expect you to apply the material. The final product, the project, will be a combination of all the subparts in this roadmap.

As the project builds on the parts of the previous weeks, we recommend programming in **Jupyter Notebook (.ipynb)**. Furthermore, while programming each part every week, we recommend immediately writing the corresponding part of the report.

2.1 Week 1 & 2: set up environment and agent

Please note that there is no in-person work group in the week 1!

During weeks 1 and 2, you are expected to work on setting up an environment which you will use for the other parts of the project. As mentioned in the lecture you can either 1.) use an existing Gymnasium environment or 2.) Create a custom environment. It is allowed to switch environments during the course. Thus, if you initially decided to use an existing environment, but wanted to switch to a custom made environment, then you are allowed to do so.

Note: There is nothing wrong with using an existing environment, however, be aware that the selected environment plays a role in the Grading Rubric and will therefore affect your final grade (see Section 5).

2.1.1 Option 1: Gymnasium environment

For option 1, you have to choose an existing Reinforcement Learning environment from Python's Gymnasium library (Towers et al., 2024). This means you will not have to program a custom environment and agent behavior, but you will still have to program the Reinforcement Learning algorithms and write a report. As this option is less complex and requires less work, there will be a restriction on the amount of points you can obtain for the environment part in the rubric. See the Grading Rubric (Section 5), for exact specifics.

If you chose option 1, the steps are straightforward (see Section 4):

- Install Gymnasium
- Create a Jupyter Notebook or Python file
- Import Gymnasium and try to figure out how to handle Gymnasium environments, for example by trying some random actions.

2.1.2 Option 2: Guided environment creation

For option 2, you have to program your own tabular environment. You can find some examples of environments together with constraints and guidelines for your custom environment in Section 4. This option will have you program the environment and agent behavior from scratch besides also implementing the Reinforcement Learning algorithms and writing a report. As this option requires more work, a section in the Grading Rubric covers the environment. The points you can obtain for this part depend on your environment's complexity, originality and correctness. See the Grading Rubric (Section 5), for exact specifics.

If you choose option 2, you will have to code your own class for the environment. More on this can be found in Section 4.

2.2 Week 3: Dynamic Programming algorithms

This week you have to implement all three algorithms covered in the lecture and apply them to the environment you have selected or made the week before. The following algorithms should be implemented:

- Policy Evaluation, which can be used as ground truth to compare the state values to other algorithm outputs.
- Policy Improvement.
- Value Iteration.

Try to experiment with different values for γ and θ and see how those affect the algorithm's policies. Furthermore, we would like you to create some informative plots. For example, you could plot:

- The state values for each state
- The evolution of the state values over the episodes
- The final policy found by each algorithm
- The reward evolution (rewards over time) when following a policy
- A comparison between the final policies of the different algorithms and their reward evolution

Note: If you implemented an environment of your own, then you might not be certain if the rewards you defined lead to a good or logical policy. Therefore, if your algorithms return policies that are wrong or incorrect, it might be worthwhile to test your algorithm in a Gymnasium environment as well to determine if your implementation of the algorithm or your custom environment is the source of your troubles.

2.3 Week 4: Monte Carlo algorithms

For this week, you have to program two algorithms:

- On-policy, first-visit Monte Carlo prediction.
- On-policy, first-visit Monte Carlo control.

Again, find the correct hyperparameters, and create informative plots.

2.4 Week 5: Temporal Difference algorithms

For this week, you have to program three algorithms:

- TD(0) (TD prediction).
- Sarsa (on-policy TD control).
- Q-learning (off-policy TD control).

Yet again, find the correct hyperparameters, and create informative plots.

2.5 Week 6 & 7: Report

Note that there is no lecture and work group in week 6 because of Carnival!

Weeks 6 and 7 (and further into the exam week until the deadline) can be used as 'overflow week' if you ran behind on schedule, and are also meant to work on your report. Please refer to Section 3 for the required contents of the report.

3 Report

Besides programming various RL algorithms, you need to write a report containing the following sections:

- **Introduction.** Describe your environment and the problem the agent has to solve. Moreover, describe the objective of the report (e.g. comparing various RL algorithms), and how you are going to accomplish this (research question).
- **Dynamic Programming algorithms.** *Describe how the algorithms work, how the various algorithms differ, plot results and/or policies, etc. **Note:** describing is **not** copying your code into the report.
- **Monte Carlo algorithms.** *
- **Temporal Difference algorithms.** *
- **Discussion.** Finally, compare all algorithms. Do certain algorithms have certain strengths or limitations? What limitations does the environment impose? Furthermore, think about how you can show the differences between the algorithms.
- **Conclusion.** Conclude the project.

Note: Although it is not required, we strongly encourage you to write the corresponding parts of the report concurrently with the implementation of the algorithms.

4 Project environments

Before you get started with a predefined or self-build environment, ensure you have installed the Gymnasium library:

```
# Install Gymnasium
pip install gymnasium
```

Import necessary modules:

```
import gymnasium as gym
import numpy as np
```

4.1 Option 1: Predefined environment

On the Gymnasium (Towers et al., 2024) website (gymnasium.farama.org) you can find a list of predefined environments. For the techniques discussed in this course and summarized in Section 2, you need to use environments where a **tabular approach** is feasible. The predefined environments that we recommend are highlighted in Table 1.

Environment Type	Examples
Toy Text	Blackjack, FrozenLake, CliffWalking, Taxi

Table 1: Discrete Gymnasium Environments

The following example details how a Gymnasium environment can be imported:

```
# Import Gymnasium and create FrozenLake environment
import gymnasium as gym

# Create the FrozenLake environment
env = gym.make('FrozenLake-v1')

# Reset the environment to start
obs, info = env.reset()

# Display the initial state
print("Initial␣Observation:", obs)
```

4.2 Option 2: Make your own environment

Make sure you additionally import `spaces` form Gymnasium:

```
import gymnasium as gym
from gymnasium import spaces
import numpy as np
```

Step 1: Define the Environment Class

Create a subclass of `gym.Env` and implement the required methods.

```python
class CustomEnv(gym.Env):
    def __init__(self):
        super(CustomEnv, self).__init__()

        # Define action and observation spaces
        self.action_space = spaces.Discrete(2)  # Example:
            ↪ two actions
        self.observation_space = spaces.Box(low=0, high=10,
            ↪ shape=(1,), dtype=np.float32)

        self.state = None  # Initialize state

    def reset(self):
        """Reset the environment to an initial state."""
        self.state = np.array([5.0])  # Example initial
            ↪ state
        return self.state, {}

    def step(self, action):
        """Apply an action and return results."""
        reward = 1 if action == 1 else 0
        self.state = self.state + (action - 0.5)
        done = self.state[0] > 10 or self.state[0] < 0
        return self.state, reward, done, False, {}

    def render(self):
        """Render the environment (optional)."""
        print(f"Current␣state:␣{self.state}")

    def close(self):
        """Clean up resources (optional)."""
        pass
```

Step 2: Register the Environment

Register your environment to use it with Gymnasium.

```python
from gymnasium.envs.registration import register

register(
    id='CustomEnv-v0',
    entry_point='__main__:CustomEnv',
)
```

This is necessary to be able to use your environment natively in Gymnasium.

Step 3: Test the Environment

Use Gymnasium's API to test your custom environment.

```
# Create the environment
env = gym.make('CustomEnv-v0')

# Interact with the environment
obs, info = env.reset()
for _ in range(10):
    action = env.action_space.sample()  # Random action
    obs, reward, done, truncated, info = env.step(action)
    env.render()
    if done:
        break

env.close()
```

Requirements for custom environments

- Ensure that both the state space and the action space of the environment are **discrete**.
- The size of the action space A and size of the state space S should not be very large ($A \cdot S < 600$).
- Depending on your environment the amount of terminal states can vary. Some problems will only require one, while others might require more. Keep the number of terminal states to a reasonable amount that fits logically with your environment.
- Ensure that all Markov decision process (MDP) properties are met.

Examples

- Grid-based environments, such as Gridworld, mazes, or 3D Gridworlds
- Navigation in real-world-like settings, e.g., the campus of Radboud University, daily activities
- Any general state-transition model
- Games, such as Tic-Tac-Toe, Snakes and Ladders, Black Jack, etc.

Tips and Best Practices

- Use `spaces.Discrete` for discrete actions.
- Due to the nature of some algorithms, it is required for your environment to be **episodic** and therefore have terminal state(s).

- Decide if the environment is a stochastic process or a deterministic process.
- Clearly define the reward structure. In this sense, think about how a specific reward will impact the agent's behavior and how immediate rewards and cumulative rewards play a role. What is the objective of your environment? What do you want the agent to learn? and how do get it to show this behavior?
- Test your environment extensively to ensure that the RL algorithms are capable of learning an optimal policy.
- It can be helpful to use additional logging capabilities within Gymnasium, see Custom Gymnasium wrappers.

For more detailed instructions on building a custom Gymnasium environment, read this documentation or ask the teaching assistants.

Lastly, here is a list containing various functions that could be convenient for your own environment class:

- An initialization function of the class (def __init__(self)).
- A function to reset the environment and agent (def reset(self)).
- A function that lets the agent take one step given a current state s and an action a, returning a new state s' and a reward r. (def step(self)
- A function that computes (or sets) the value for $p(s', r|s, a)$.
- A function that samples an action a from the policy $\pi(s)$, given a state s.
- A function that samples a full episode (sequence of (s, a, r) tuples) until termination or maximum time T.
- A function returning the nonterminal states.
- A function returning the terminal states.
- A function returning a Boolean whether you are on a terminal state.
- A function that converts a state-action values to state values.
- A function that converts state values into state-action values.
- A function that prints the current state (can be in ASCII), or just textually.
- A function that prints (or plots) a given policy.
- A function that prints the state values for each state.
- A function that prints the state-action values for each state-action pair.

Note: Not all of these functions might be equally necessary, depending on your environment.

5 Grading rubric

The project and report will be graded according to the following Grading Rubric:

Component	Criteria	Points	Content
Environment	***Note: Readymade Gymnasium environments are ineligible for environment points***		
	Originality	1 pt.	Environment choice and complexity, the rationale behind rewards and transition probabilities
	Correctness	1 pt.	Implementation, functioning, completeness
Dynamic Programming	Code	1 pt.	functionality, correctness, performance
	Report	1 pt.	description, documentation, plotting
Monte Carlo	Code	1 pt.	functionality, correctness, performance
	Report	1 pt.	description, documentation, plotting
Temporal Difference	Code	1 pt.	functionality, correctness, performance
	Report	1 pt.	description, documentation, plotting
Written report	Introduction	0,5 pt.	Description of the environment, problem, research question
	Results and choice of metrics	0,5 pt.	Justification of chosen metrics, clear overview and description of results
	Discussion, Conclusion	0,5 pt.	Discussion and comparison of results, conclusion of findings and project
	Writing style, grammar, formatting, etc	0,5 pt.	
Bonus	***Note: The final grade you can obtain for the project will be capped at a 10.0***		
	Deep Reinforcement Learning Implementation	0,5 pt.	
	Exceptional environment	0,5 pt.	

References

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*.