SOW-BKI258 Reinforcement Learning Assignment

Contents

1 Introduction **2**

2 Course schedule **3**

3 Roadmap **3**

3.1 Workgroup 1: set up environment and agent 3

3.1.1 Option 1: Gymnasium environment 4

3.1.2 Option 2: Guided environment creation 4

3.2 Workgroup 2: Dynamic Programming algorithms 4

3.3 Workgroup 3: Monte Carlo algorithms 5

3.4 Workgroup 4: Temporal Difference algorithms 6

3.5 Workgroup 5: Overflow week 6

4 Report (in the notebook) **6**

5 Coding in your notebook **7**

6 Project environments **8**

6.1 Option 1: Predefined environment 8

6.2 Option 2: Make your own environment 8

7 Grading rubric **12**

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 6), implement various RL algorithms for this environment (see Section 3), and write a report in a notebook comparing the RL algorithms (see Section 4).

The **deadline** for handing in the project is set at **April 5th, 23:59**. Only one group member needs to hand in the project. This is a strict deadline; late submissions will not be accepted.

You will be working on these projects in **groups of strictly 4 students**. If you need to find group members, utilize the dedicated Discord channel. Enrollment in these groups is done through Brightspace: go to the course page (2526 Reinforcement Learning PER 3 V) → Administration → Groups. Your group number will also decide your room allocation, as displayed in the group enrollment page. Group enrollment will open after the first lecture. You should be enrolled in a group to be able to view and hand in the assignment.

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

- Three Python files, each containing a class of Reinforcement Learning algorithms, listed in the following point.
- A requirements file which you update along your project. (Do **not** hand in your virtual environment!)
- A Jupyter Notebook (see the template on Brightspace) 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
  - Your report, which is made in the Jupyter Notebook. It should consist of around 1000-1500 words. See Section 4 for more details.

2 Course schedule

Can be subject to change! Any changes will be communicated via Brightspace announcements.

Date	Type	Description
30-01-2026	Lecture	Lecture 1: Introduction
06-02-2026	Lecture	Lecture 2: Markov Decision Processes
09-02-2026	Workgroup	Workgroup 1: create environment
13-02-2026	Lecture	Lecture 3: Dynamic Programming
20-02-2026	Lecture	Lecture 4: Monte Carlo methods
20-02-2026	Replacement workgroup	Workgroup 2: Dynamic Programming (13:30-15:15)
23-02-2026	Workgroup	Workgroup 3: Monte Carlo
27-02-2026	Lecture	Lecture 5: Temporal Difference Learning
02-03-2026	Workgroup	Workgroup 4: Temporal Difference Learning
06-03-2026	Lecture	Lecture 6: Tabular methods, Summary and Exam
09-03-2026	Workgroup	Workgroup 5: overflow week
13-03-2026	Lecture???	Q&A????
25-03-2026	Exam	Exam
05-04-2026	Report	Deadline report
23-06-2026	Resit exam	Resit exam

3 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.

3.1 Workgroup 1: set up environment and agent

February 9th. Please note that there is no in-person work group the Monday after the first lecture; there is a workgroup the Monday after the second lecture.

During the first weeks, 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 7).

3.1.1 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 in the notebook. 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 7), for exact specifics.

If you chose option 1, the steps are straightforward (see Section 6):

- 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.

3.1.2 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 6. This option will have you program the environment and agent behavior from scratch besides also implementing the Reinforcement Learning algorithms and writing a report in the notebook. 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 7), 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 6.

3.2 Workgroup 2: Dynamic Programming algorithms

There will be no in-person workgroup this week due to Carnival.

This week you have to implement two algorithms covered in the lecture and apply them to the environment you have selected or made in the previous work-group:

1. Policy Iteration, which includes two parts:
   - **Iterative policy evaluation:** which can be used as ground truth to compare the state values to other algorithm outputs.
   - **Policy Improvement:** greedy strategy
2. 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 any one of:

- The final policy found by each algorithm (recommended)
- The state values for each state (or selected subset of states when the state space is large)
- The evolution of the state values (selected states) over iterations
- 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.

3.3 Workgroup 3: Monte Carlo algorithms

February 23rd.

For this week, you can choose to program **one of the two** algorithms below:

1. Monte Carlo Exploring Starts (See Section 5.3 in the textbook), which includes two parts
   - ***Prediction***: First-visit Monte Carlo prediction, with exploring starts. *(Our suggestion is to evaluate action values.)*
   - ***Control:*** Greedy strategy
2. Monte Carlo without Exploring Starts (See Section 5.4 in the textbook), which also includes two parts

- ***Prediction***: First-visit Monte Carlo prediction without exploring starts. *(Our suggestion is to evaluate action values.)*
- ***Control:*** ε-Greedy strategy

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

3.4 Workgroup 4: Temporal Difference algorithms

February 30th.

For this week, you need to program **both** algorithms below:

1. On-policy algorithm: Sarsa (See Section 6.4 in the textbook), which includes two parts
   - ***Prediction:*** TD(0), *(Our suggestion is to evaluate action values.)*
   - ***Control:*** ε-greedy strategy
2. Off-policy algorithm: Q-learning (See Section 6.5 in the textbook), which also includes two parts
   - ***Prediction:*** Q-learning, *(Our suggestion is to evaluate action values.)*
   - ***Control:*** ε-greedy strategy

Yet again, find the correct hyper-parameters and create informative plots.

Note: As we mentioned in our lecture, ε-greedy policies are usually not optimal and a decreasing ε overtime has a higher chance to converge to an optimal policy. Think about how to decrease ε strategically for the assignments in Weeks 4 and 5. *This is optional, but we will give you some bonus points.*

3.5 Workgroup 5: Overflow week

March 9th.

The final workgroup (and further into the exam week until the deadline) can be used as 'overflow week' if you ran behind on schedule, and is also meant to work on the comparison section. Please refer to Section 4 for the required contents of the report within the Jupyter Notebook.

4 Report (in the notebook)

Besides programming various RL algorithms, you need to write your **report in the Jupyter Notebook** (see the template!), 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.
- **Monte Carlo algorithms.** *
- **Temporal Difference algorithms.** *
- **Comparison and Discussion:** Compare different algorithms (MC and TD with plots[1]). You can choose to plot any of the following:
  - Cumulative reward: The total reward accumulated over each episode. As learning progresses, cumulative rewards are expected to increase over episodes, reflecting the agent's improving performance.
  - Root mean squared errors (between estimated and true values) averaged over states or state-action pairs, against episodes. As learning progresses, the errors are expected to decrease.
  - Sample efficiency: The number of episodes needed to achieve a certain performance level (e.g., finding an optimal policy).
  - Any other metric that you think is a fair comparison.

  Include a short discussion: what can you conclude by comparing different RL algorithms? Do they have certain strengths or limitations?
- **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.

5 Coding in your notebook

In your notebook, code your environment. Code all Reinforcement Learning algorithms in the Python files (`dp.py`, `mc.py` and `td.py`).

Now, from your notebook, call the functions from your Python files, for example: `dp.run_policy_iteration(env, args)`. This should then return your outputs, plots, etc. to display in your 'report'. For further guidance on this, utilize the template, which you can find in the assignment.

[1] You don't need to plot DP alongside MC and TD since DP is not a learning algorithm. However, DP can provide the ground truth for optimal state or action values, which can serve as a reference when evaluating MC and TD.

6 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
```

6.1 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 3, 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)
```

6.2 Option 2: Make your own environment

Make sure you additionally import `spaces` from 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.

```
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.

```
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.

```python
# 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.

7 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
Report parts	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.	Also includes use of the Jupyter Notebook, e.g. code cluttering in the notebook.
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*.