CSE 4/546: Reinforcement Learning
Fall 2024

Instructor: Alina Vereshchaka

Assignment 1 - Defining & Solving RL Environments

Checkpoint: September 12, Thu, 11:59pm

Due Date: September 26, Thu, 11:59pm

1 Assignment Overview

The goal of the assignment is to acquire experience in defining and solving reinforcement learning environments, following Gymnasium standards. The assignment consists of three parts. The first focuses on defining deterministic and stochastic environments that are based on Markov Decision Process. In the second part we will apply two tabular methods to solve environments that were previously defined. In the third part we will apply Q-learning algorithm to solve a stock-trading environment.

Part 1: Defining RL Environments [30 points]

In this section we are defining an RL environment following Gymnasium standards.

1.1 Deterministic Environment [20 points]

Steps:

1. Refer to our the real-world scenarios listed at Section 2.1. Pick one scenario that you will work to define a deterministic and stochastic environment. You can implement multiple scenarios, however only one will be evaluated.
2. Define a deterministic environment, where $P(s', r|s, a) = \{0, 1\}$. Refer to the suggested deterministic environment setup for your chosen scenario. Refer to Section 2.1 for more details.
3. Run a random agent for at least 10 timesteps to show that the environment logic is defined correctly. Print the current state, chosen action, reward and return your grid world visualization for each step.

1.2 Stochastic Environment [10 points]

Steps:

1. Define a stochastic environment, where $\sum_{s',r} P(s', r|s, a) = 1$. A modified version of the environment defined in Part 1.1 should be used. Refer to the suggested stochastic environment setup for your chosen scenario.
2. Run a random agent for at least 10 timesteps to show that the environment logic is defined correctly. Print the current state, chosen action, reward and return your grid world visualization for each step.

2 Supplementary Materials

2.1 Environment Requirements

Environment definition should follow the Gymnasium structure, which includes the basic methods:

```
def __init__:
    # Initializes the class
    # Define action and observation space

def step:
    # Executes one timestep within the environment
    # Input to the function is an action

def reset:
    # Resets the state of the environment to an initial state

def render:
    # Visualizes the environment
    # Any form like vector representation or visualizing using matplotlib will be sufficient
```

Environment Scenarios

1. Warehouse Robot

Scenario: A robot operates in a warehouse grid where its task is to pick up items from specified locations and deliver them to designated drop-off points. The warehouse has shelves that act as obstacles, and the robot must navigate around them efficiently.

Environment setup:

- **Grid size:** 6x6 grid
- **Obstacles:** Shelves (static obstacles) that block the robot's path
- **Goal:** The robot must pick up an item from an assigned location and deliver it to another specified location
- **Actions:** Up, Down, Left, Right, Pick-up, Drop-off
- **Rewards:**

  +20pts for successful delivery

  -1pt for each step taken to encourage efficiency

  -20pts for hitting an obstacle
- **Terminal state:** The robot successfully delivers the item
- **Suggested deterministic environment setup:** All the obstacles, drop-off points, and the outcome of agent's actions are fixed.
- **Suggested stochastic environment setup.** Choose one or more from the following:
  - The outcomes of agent's actions are not fixed. E.g., if the agent takes the action to go right, the agent ends up in the grid block on the right with a probability of 0.9 (90%), but stays in the same grid block with a probability of 0.1 (10%).

- Introduce other robots that are randomly moving in the environment. Two robots cannot occupy the same grid location.

Requirements:

- The robot (RL Agent) should be able to navigate from any start position to any target position.
- Implement collision detection to prevent the robot from moving through shelves.

Bonus [4 points]

- The robot should be capable of multiple pick-up and drop-off tasks in a single episode. The items have to be picked up from randomly assigned locations.

2. Traffic Light Control

Scenario: A traffic light controller operates at a 4-way intersection. The goal is to minimize the average wait time of cars by optimizing the traffic light switching strategy.

Environment Setup:

- **Grid Size:** 4x4 grid representing the intersection.
- **Cars:** Cars arrive at the intersection and must wait until they can move forward.
- **Goal:** Minimize the average wait time of cars at the intersection.
- **Actions:** Switch to Red, Green, or Yellow for each of the four directions.
- **Rewards:**
  - -1 for each second a car waits.
  - +5 for each car that successfully passes through the intersection.
- **Terminal State:** Defined by a time limit or a certain number of cars processed.
- **Suggested Deterministic Environment Setup:** Traffic flow is fixed, meaning cars arrive at regular intervals in each direction. The timing and number of cars arriving are predictable.
- **Suggested Stochastic Environment Setup:** Traffic flow is random, with cars arriving at irregular intervals.

Requirements:

- The controller (RL Agent) must handle varying traffic conditions (e.g., rush hour vs. off-peak times).
- The system (RL Agent) should handle at least 50 cars in a single episode.

Bonus:

- Implement a more complex intersection with two lanes where vehicles can make turns. Update the action-space to include traffic lights for turns.
- Ensure that the RL agent can't perform illegal action sequences such as 1. Green 2. Yellow 3. Green.

3. Autonomous Drone Delivery

Scenario: An autonomous drone operates in a small city grid. The drone's task is to deliver packages to various locations while avoiding no-fly zones.

Environment Setup:

- **Grid Size:** 6x6 grid representing the city layout.
- **Obstacles:** No-fly zones (static obstacles) that the drone cannot pass through.
- **Goal:** Deliver a package from a warehouse (starting point) to a customer's location.
- **Actions:** Up, Down, Left, Right, Pickup, Drop-off.
- **Rewards:**
  - +20 for successful delivery.
  - -1 for each step taken.
  - -20 for entering a no-fly zone.
- **Terminal State:** The drone successfully delivers the package.
- **Suggested Deterministic Environment Setup:**
  - The drone's movements are exact, with no deviation from its intended path. For instance, when the drone moves up, it always moves exactly one grid square up.
  - No-fly zones are static and do not change during the course of the delivery, allowing the drone to plan its route perfectly.
- **Suggested Stochastic Environment Setup:** Choose one or more from the following:
  - The drone's movements are subject to external factors such as wind. For example, if the drone attempts to move up, it successfully moves up with a probability of 0.9 (90%) but may deviate to an adjacent grid block with a probability of 0.1 (10%).
  - No-fly zones may appear or disappear randomly, forcing the drone to adapt its route in real-time.

Requirements:

- The drone (RL Agent) should be able to reroute dynamically if a no-fly zone is encountered.
- The drone (RL Agent) should be able to navigate from any start position to any target position.

Bonus:

- Implement scenarios where the drone must make multiple deliveries in one episode.

4. Smart Elevator System

Scenario: A smart elevator system in a multi-story building must transport people efficiently. The elevator should minimize wait times and the number of stops.

Environment Setup:

- **Grid Size:** A 10x1 grid representing 10 floors.
- **Requests:** People requesting the elevator on different floors, either going up or down.
- **Goal:** Minimize the total wait time and travel time for all people.
- **Actions:** Move Up, Move Down, Open Door, Close Door, Idle.
- **Rewards:**
  - +5 for each person who successfully reaches their destination.
  - -1 for each second a person waits.

  - -2 for unnecessary stops.

- **Terminal State:** After a fixed number of people are serviced or a time limit is reached.
- **Suggested Deterministic Environment Setup:**
  - Elevator requests are deterministic, meaning they occur at regular intervals and at fixed floors.
  - The elevator moves between floors at a constant speed, with no variation in travel time.
- **Suggested Stochastic Environment Setup:** Choose one or more from the following:
  - Elevator requests are stochastic, occurring at random times and from random floors, making it difficult to predict the next request.

Requirements:

- The elevator (RL Agent) should prioritize requests to minimize the total wait and travel time.
- The system (RL Agent) should be able to handle both peak and off-peak scenarios.

Bonus:

- Implement scenarios where the elevator has a maximum occupancy limit of 6 people, however, there are more people who would want to get on.

In your report for Part 1:

1. Describe the deterministic and stochastic environments, which were defined (set of actions/states/rewards, main objective, etc).
2. Provide visualizations of your environments.
3. How did you define the stochastic environment?
4. What is the difference between the deterministic and stochastic environments?
5. **Safety in AI**: Write a brief review ($\sim$ 5 sentences) explaining how you ensure the safety of your environments. E.g. how do you ensure that agent choose only actions that are allowed, that agent is navigating within defined state-space, etc.

Part 2 [Total: 40 points] - Applying Tabular Methods

2.1 Q-learning

Steps:

1. Apply **Q-learning** to solve both the deterministic and stochastic environments that were defined in Part 1.
2. Save the Q-table/Policy table as a pickle file or h5 of the trained model and attach it to your assignment submission.
3. Provide the evaluation results:
   (a) Print the initial Q-table and the trained Q-table
   (b) Plot the total reward per episode graph (x-axis: episode, y-axis: total reward per episode).
   (c) Plot the epsilon decay graph (x-axis: episode, y-axis: epsilon value)
   (d) Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learned policy. Include a plot of the total reward per episode.

4. Hyperparameter Tuning: Select at least two hyperparameters to tune to get better results for Q-learning. You can explore hyperparameter tuning libraries, e.g. Optuna or make it manually. Parameters to tune (select 2):

   - Discount factor (γ)
   - Epsilon decay rate
   - Epsilon min/max values
   - Number of episodes
   - Max timesteps

   Try at least 3 different values for each of the parameters that you choose

5. Provide the evaluation results (refer to Step 3) and your explanation for each result for each hyperparameter. In total, you should complete Step 3 seven times [Base model (step 1) + Hyperparameter #1 x 3 difference values & Hyperparameter #2 x 3 difference values]. Make your suggestion on the most efficient hyperparameters values for your problem setup.

2.2 Other tabular method

Steps:

1. Apply **any other tabular method** to solve both the deterministic and stochastic environments that were defined in Part 1. E.g. SARSA, Double Q-learning, Monte Carlo, n-step bootstrapping, etc.
2. Save the Q-table/Policy table as a pickle file or h5 of the trained model and attach it to your assignment submission.
3. Provide the evaluation results:
   (a) Print the initial Q-table and the trained Q-table
   (b) Plot the total reward per episode graph (x-axis: episode, y-axis: total reward per episode).
   (c) Plot the epsilon decay graph (x-axis: episode, y-axis: epsilon value)
   (d) Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learned policy. Include a plot of the total reward per episode.

In your report for Part 2:

1. Show and discuss the results after:

   - Applying Q-learning to solve the deterministic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.
   - Applying Q-learning to solve the stochastic environment defined in Part 1. Plots should include epsilon decay and total reward per episode.
   - Applying any other algorithm of your choice to solve the deterministic environment defined in Part 1. Plots should include total reward per episode.
   - Applying any other algorithm of your choice to solve the stochastic environment defined in Part 1. Plots should include total reward per episode.
   - Provide the evaluation results. Run your environment for at least 10 episodes, where the agent chooses only greedy actions from the learnt policy. Plot should include the total reward per episode.

2. Compare the performance of both algorithms on the same deterministic environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.

3. Compare how both algorithms perform in the same stochastic environment (e.g. show one graph with two reward dynamics) and give your interpretation of the results.
4. Briefly explain the tabular methods, including Q-learning, that were used to solve the problems. Provide their update functions and key features.
5. Briefly explain the criteria for a good reward function. If you tried multiple reward functions, give your interpretation of the results.

Important note on reusing your previous submissions
If you have completed a similar task as part of your other CSE course. While you can reuse your code, we would like to see you come up with a couple of improvements in your submission:

- We expect you to come up with a new environment definition, e.g. different board configuration, reward function, objective, etc
- You may use your logic for the algorithms implemented before, but we would like you to optimize your algorithms if applicable.
- If you are using the logic/implementation from your previous assignment for this submission, make sure to add relevant references. E.g. *"Part I is based on my (John Smith) Assignment 2 submission for CSE 574, Fall 2023."*

Part 3 [Total: 30 points] - Solve Stock Trading Environment

Task:

In this part, you need to apply a Q-learning agent that you implemented in Part 2.1 to learn the trends in stock price and perform a series of trades over a period of time to end up with a profit. You can modify your Q-learning code, if needed.

In each trade you can either buy/sell/hold. You will start with an investment capital of $100,000 and your performance is measured as a percentage of the return on investment. Save the Q-table as a pickle file and attach it to your assignment submission.

Stock trading Environment

This environment is based on the dataset on the historical stock price for Nvidia for the last 2 years. The dataset has 523 entries starting 07/01/2022 to 07/31/2024. The features include information such as the price at which the stock opened, the intraday high and low, the price at which the stock closed, the adjusted closing price and the volume of shares traded for the day.

The environment which calculates the trends in the stock price is provided to you along with the documentation in the .ipynb file. Your task is to use the Q-learning algorithm to learn a trading strategy and increase your total account value over time.

Environment Description

Init method: This method initializes the environment.
Input parameters:

1. file_path: Path of the CSV file containing the historical stock data.
2. train: - Boolean indicating whether the goal is to train the agent or evaluate the performance of the agent.
3. number_of_days_to_consider: Integer representing the number of days the for which the agent considers the trend in stock price to make a decision.

Reset method: This method resets the environment and returns the observation.

Returns:

1. observation: - The environment observation is an integer in the range of 0 to 3 derived from the following vector: [price_increase, stock_held]. It represents the four possible observations that the agent can receive. The observation depends upon whether the price increased on average in the number of days the agent considers, and whether the agent already has the stock or not. E.g.,

```
if [price_increase, stock_held] == [True, False]:
    observation = 0
```

2. info: - A dictionary that can be used to provide additional implementation information (Is empty for our environment).

Step method: This method implements what happens when the agent takes the action to Buy/Sell/Hold. Input parameter: action: - Integer in the range 0 to 2 inclusive. 0 = Buy, 1 = Sell, 2 = Hold.

Returns:

1. observation: - The environment observation is an integer in the range of 0 to 3 derived from the following vector: [price_increase, stock_held]. It represents the four possible observations that the agent can receive. The observation depends upon whether the price increased on average in the number of days the agent considers, and whether the agent already has the stock or not. E.g.,

```
if [price_increase, stock_held] == [True, True]:
    observation = 1
```

2. reward: - Integer/Float value that's used to measure the performance of the agent. The reward is calculated based on the change in stock price and a penalty is added for illegal actions (such as trying to sell when the agent doesn't hold any stock).
3. terminated: - Boolean describing if the episode has terminated.
4. truncated: - Boolean describing if a truncation condition outside the scope of the MDP is satisfied (Always returns 'False' for our environment).
5. info: - A dictionary that can be used to provide additional implementation information (Empty for our environment).

Render method: This method renders the agent's total account value over time. Input parameter: mode: 'human' renders to the current display or terminal and returns nothing.

Note: You can't make changes to the environment.

In your report for Part 3:

1. Show and discuss the results after applying the Q-learning algorithm to solve the stock trading problem. Plots should include epsilon decay and total reward per episode.
2. Provide the evaluation results. Evaluate your trained agent's performance (you will have to set the train parameter set to False), by only choosing greedy actions from the learnt policy. Plot should include the agent's account value over time. Code for generating this plot is provided in the environment's render method. Just call environment.render after termination.

Extra Points [max +12 points]

- **Git Expert [2 points]**
  Git is one the essential tools to know, you can check some foundations of it here.
  - Along with your submission at UBlearns, upload your project on GitHub in a **private repository** before A1 Checkpoint Due Date
  - Add **@ub-rl** as collaborators before A1 Checkpoint Due Date.
  - Make at least 7 commits on your A1 project on GitHub for the checkpoint and 7 commits for the final submission.
  - In your report include a link to your private GitHub project and a snapshot of your commits history. Reports without the link and snapshot provided, will not be considered for this bonus.
- **CCR Submission [3 points]**

  Submit a screenshot of successful execution of your code on CCR. Depending on which method you are using to access CCR resources, requirements for proof of successful use of CCR are as follows:
  - VsCode Session:
    * Open the terminal on VsCode and run your code.
    * Provide a screenshot of the entire VsCode window with the terminal visible. The terminal should show successful execution of your code.
  - Desktop Session:
    * Open the terminal and run your code.
    * Provide a screenshot of the entire Desktop window with the terminal open. The terminal should show successful execution of your code.
  - Jupyter Notebook Session:
    * Since there is no terminal here, print your working directory (!pwd) showing that you are in your own home folder. It should have your ubitname somewhere in the path.
    * Send a screenshot of the entire screen showing the output of this command and successful execution of your code (just the final results will suffice).
- **Grid-World Scenario Visualization [3 points]**
  Your grid has to contain different images to represent:
  - Agent - At least two images dependent on what the agent is doing.
  - Appropriate background for your scenario (Not the default one)
  - Images representing each object in your scenario.
- **Complex Environment Scenario [4 points]**
  Implement and solve a more complex environment scenario as shown in the "Bonus" section highlighted in Green for the various scenarios in Part 2.1.

4 Assignment Steps

4.1 Submit Checkpoint [Sep 12]

- Complete Part I & Part 2.1 (Q-learning)
- For the checkpoint, it is okay if your report is not fully completed. You can finalize it for the final submission.

- The code of your implementations should be written in Python. You can submit multiple files, but they all need to be labeled clearly.
- Add your code as .ipynb and your draft report in pdf to the zip folder. Submit your .zip folder named as: $YOUR\_UBIT\_assignment1\_checkpoint.zip$ (e.g. $avereshc\_assignment1\_checkpoint.zip$
- Submit at UBLearns > Assignments

4.2 Submit Final results [Sep 26]

- Fully complete all parts of the assignment and submit to UBLearns > Assignments.
- The code of your implementations should be strictly written in a Python notebook (ipynb file). You can submit multiple files, but they all need to be labeled clearly.
- In case you submit multiple files, all assignment files should be packed in a ZIP file named: $YOUR\_UBIT\_assignment1.zip$ (e.g. $avereshc\_assignment1.zip$).
- Your Jupyter notebook should be saved with the results. Do not submit JSON/HTML files, as these will not be graded.
- Additionally submit the trained parameters as a pickle file or .h5, so that the grader can fully replicate your results.
- For the final submission you can combine the code from all parts into one.
- Include all references that have been used to complete the assignment.

5 Grading Details

Checkpoint Submission

- Graded on a 0/1 scale.
- A grade of "1" is awarded for completing more than 80% of the checkpoint-related parts and "0" is assigned for all other cases.
- Receiving a "0" on a checkpoint submission results in a 30-point deduction from the final assignment grade.

Final Submission

- Graded on a scale of X out of 100 points, with potential bonus points (if applicable).
- All parts of the assignment are evaluated during final evaluation.
- Ensure your final submission includes the final version of all assignment parts.

Important Notes

1. Only files submitted on UBLearns are considered for evaluation.
2. Files from local devices, GitHub, Google Colab, Google Docs, or other locations are not accepted.
3. Regularly submit your work-in-progress on UBLearns and always verify submitted files by downloading and opening them.

6 References

- RL Handbook
- NIPS Styles (docx, tex)
- Overleaf (LaTex based online document generator) - a free tool for creating professional reports
- GYM environments
- Lecture slides
- Richard S. Sutton and Andrew G. Barto, "Reinforcement learning: An introduction" (pdf)

7 Late Days Policy

You can use up to 5 late days throughout the course toward any assignment's checkpoint or final submission. You don't have to inform the instructor, as the late submission will be tracked in UBLearns.

8 Academic Integrity

This assignment should be done individually.
The standing policy of the Department is that all students involved in any academic integrity violation (e.g. plagiarism in any way, shape, or form) will receive an F grade for the course. The catalog describes plagiarism as "Copying or receiving material from any source and submitting that material as one's own, without acknowledging and citing the particular debts to the source, or in any other manner representing the work of another as one's own.". Refer to the Office of Academic Integrity for more details.

9 Important Dates

September 12, Thu 11:59 pm ET - Checkpoint is Due

September 26, Thu, 11:59 pm ET - Assignment 1 is Due