CSE 4/546: Reinforcement Learning
Spring 2023

Instructor: Alina Vereshchaka

Assignment 1 - Defining & Solving RL Environments

Checkpoint: February 16, Thu, 11:59pm

Due Date: March 2, Thu, 11:59pm

1 Assignment Overview

The goal of the assignment is to acquire experience in defining and solving reinforcement learning environments, following OpenAI 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 [Total: 30 points] - Defining RL Environments

1.1 Deterministic Environment [15 points]

Define a deterministic environment, where $P(s', r|s, a) = \{0, 1\}$. Run a random agent for at least 10 timesteps to show that the environment logic is defined correctly.

Environment requirements:

- Min number of states: 12
- Min number of actions: 4
- Min number of rewards: 4

Environment definition should follow OpenAI Gym structure, which includes the following 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
```

1.2 Stochastic Environment [15 points]

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. Run a random agent for at least 10 timesteps to show that the environment logic is defined correctly.

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

Steps:

1. Apply **two tabular methods** to solve both the deterministic and stochastic environments that were defined in Part 1. You need to implement **Q-learning and any other tabular algorithm of your choice** (e.g. SARSA, Double Q-learning, Monte Carlo or n-step bootstrapping).
2. Save the Q-table/Policy table as a pickle file for both algorithms implemented and attach it to your assignment submission.
3. **Hyperparameter Tuning:**
   For Q-leanring algorithm implemented in Part 2.1 provide the analysis after tuning at least **two hyperparameters** listed below (You can use the Optuna library):
   - 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. Provide the reward graphs and your explanation for each of the results. In total you should have at least 6 graphs and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup.

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.

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

In 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 initial 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.

3.1 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 504 entries starting 02/01/2021 to 01/31/2023. 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 Structure:

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 or test the performance of the agent.
3. number_of_days_to_consider = Integer representing whether 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.

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 whether or not the episode has terminated.
4. truncated: - Boolean describing whether a truncation condition outside the scope of the MDP is satisfied.
5. info: - A dictionary that can be used to provide additional implementation information.

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.

Recommended graphs that can help to debug the code

- Total reward per episode during training.
- Total reward per episode during evaluation. Ideally the graph is around the max reward that the agent can get within the episode. For stochastic environments there can be some fluctuations.

- Percentage of episodes in which the agent achieves the goal. E.g. for 1000 training iterations, provide the percentage of goal achievements for every 100 iterations. You can use any size-window applicable to your setup.
- Average number of timesteps per episode.
- Average number of times our agent receives a penalty by going into the 'bad' state, if applicable.

Extra Points [max +10 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 5 commits on your A1 project on GitHub for the checkpoint and 5 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 your code as a Jupyter notebook that was executed on CCR. The submission should include the following commands in the first cells:
  - pwd - This must show that you are in your UBIT directory.
  - !which python - This must show that you are using the your Anaconda version and not the default one from CCR. (This is required as you don't have permission to install libraries on CCR's version of Anaconda). For more information please refer our CCR Guide.
- **Grid-World Scenario Visualization [5 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.

4 Deliverables

There should two parts in your submission:

4.1 Report

The report should be delivered as a separate pdf file, and it is recommended for you to use the NIPS template as a report structure. You may include comments in the Jupyter Notebook, however you will need to duplicate the results in the separate pdf file. For the final submission, combine the reports for both Part 1, Part 2, and Part 3 into one file. All the references can be listed at the end of the report.

4.2 Code

Python is the only code accepted for this project. You can submit the code in Jupyter Notebook or Python script. Ensure that your code follows a clear structure and contains comments for the main functions and some specific attributes related to your solution. You can submit multiple files, but they all need to have a clear name. After executing command python main.py in the first level directory or Jupyter Notebook, it should generate all the results and plots you used in your report and print them out in a clear manner. Additionally you must submit the trained parameters as a pickle file, so that the grader can fully replicate your results. For the final submission you can combine the code from all parts into one.

5 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)

6 Checkpoint Submission [Due date: February 16]

Complete Part 1 and submit the code and draft report. To submit your work, add your pdf, ipynb/python script to zip file $YOUR\_UBIT\_assignment1\_checkpoint.zip$ (e.g. $avereshc\_assignment1\_checkpoint.zip$ and upload it to UBlearns (Assignments section). Checkpoint will be evaluated after the final submission.

7 Final Submission [Due date: March 2]

Add your combined pdf, ipynb/python script, and pickle files containing the Q-table/Policy table for Part 1, Part 2, and Part 3 to a zip file $YOUR\_UBIT\_assignment1.zip$ (e.g. $avereshc\_assignment1.zip$) and upload it to UBLearns (Assignments section). After the assignment is graded, you may be asked to demonstrate it to the instructor if your results or reasoning in your report are not clear enough.

8 Late Days Policy

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

9 Important Dates

February 16, Thu 11:59pm - Checkpoint is Due

March 2, Thu, 11:59pm - Assignment 1 is Due