CSE 4/574: Introduction to Machine Learning, Sections A&C&D

Spring 2023

**Instructor: Alina Vereshchaka**

# Assignment 3

# Defining and Solving Reinforcement Learning Task

**Checkpoint: Apr 27, Thu, 11:59pm**

**Due Date: May 11, Thu, 11:59pm**

## Description

Welcome to the third assignment for this course. The goal of this assignment is to acquire experience in defining and solving a reinforcement learning (RL) environment, following Gym standards.

The assignment consists of three parts. The first part focuses on defining an environment that is based on a Markov decision process (MDP). In the second part, we will apply a tabular method SARSA to solve an environment that was previously defined. In the third part, we apply the Q-learning method to solve a grid-world environment.

## Part I: Define an RL Environment [30 points]

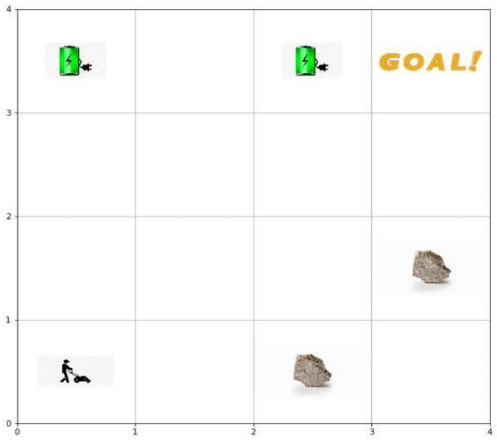
In this part, we will define a grid-world reinforcement learning environment as an MDP. While building an RL environment, you need to define possible states, actions, rewards and other parameters.

**STEPS:**

1. Choose a scenario for your grid world. You are welcome to use the visualizationdemo as a reference to visualize it.

An example of idea for RL environment:

- **Theme**: Lawnmower Grid World with batteries as positive rewards and rocks as negative rewards.
- **States:** {S1 = (0,0), S2 = (0,1), S3 = (0,2), S4 = (0,3), S5 = (1,0), S6 = (1,1), S7 = (1,2), S8 = (1,3), S9 = (2,0), S10 = (2,1), S11 = (2,2), S12 = (2,3), S13 = (3,0), S14 = (3,1), S15 = (3,2), S16 = (3,3)}
- **Actions**: {Up, Down, Right, Left}
- **Rewards**: {-5, -6, +5, +6}
- **Objective**: Reach the goal state with maximum reward




2. Define an RL environment following the scenario that you chose.

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

   Environment definition should follow the OpenAI Gym structure, which includes thebasic methods. You can use the “Defining RL env” demo as a base code.

```
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
     usingmatplotlib is sufficient
```

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.

### In your report for Part I:

1. Describe the environment that you defined. Provide a set of actions, states, rewards, main objective, etc.
2. Provide visualization of your environment.
3. Safety in AI: Write a brief review (~ 5 sentences) explaining how you ensure the safety of your environment. E.g. how do you ensure that the agent chooses only actions that are allowed, that agent is navigating within defined state-space, etc

## Part II: Solve your environment using – SARSA [40 points]

In this part, we implement SARSA (State-Action-Reward-State-Action) algorithm and apply it to solve the env defined in Part 1.

SARSA is an on-policy reinforcement learning algorithm. The agent updates its Q-values based on the current state, action, reward, and next state, action pair. It uses an exploration-exploitation strategy to balance between exploring new actions and exploiting the knowledge gained so far.

**STEPS:**

1. Apply SARSA algorithm to solve the environment that was defined in Part I.

**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

2. Try hyperparameter tuning on at least two parameters to get better results for SARSA. You can explore hyperparameter tuning libraries, e.g. Optuna or make it manually. Parameters to tune:

    a. Discount factor (γ)
    b. Epsilon decay rate
    c. Epsilon min/max values
    d. Number of episodes

e. Max timesteps

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

3. Provide the reward graphs and your explanation for each result. In total, you should have at least 3 graphs and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup.

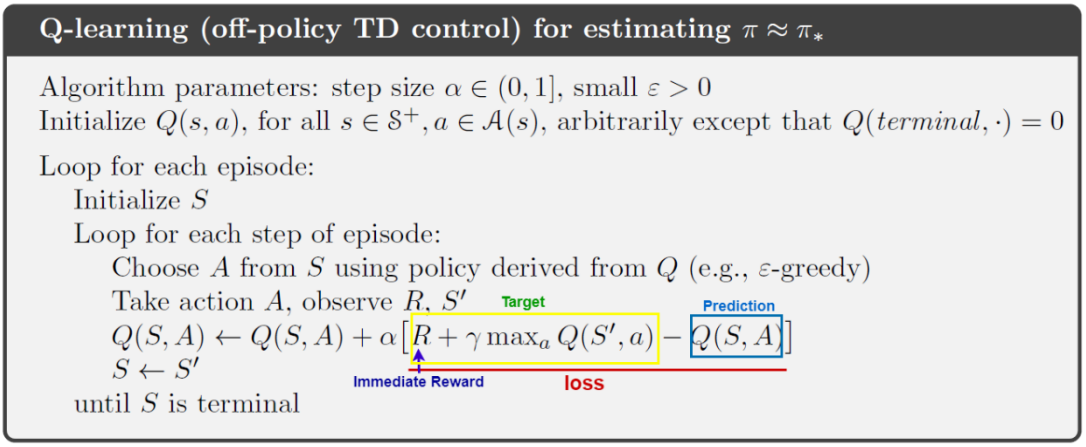
## Part III: Solve your environment using Q-learning [30 points]

In this part, we apply Q-learning algorithm to solve the env defined in Part 1.

Q-learning is an off-policy reinforcement learning algorithm. It has many similarities with SARSA. The algorithm involves taking action in the current state, observing the reward and next state, and updating the Q-value estimate of the previous state-action pair based on the maximum expected Q-value of the next state.

**STEPS:**

1. Apply Q-learning algorithm to solve the environment that was defined in Part I. You can modify your code from Part II.




2. Try hyperparameter tuning on at least two parameters to get better results for SARSA. You explore some of the hyperparameter tuning libraries, e.g. Optuna or make it manually. Parameters to tune:

f. Discount factor (γ)
g. Epsilon decay rate
h. Epsilon min/max values
i. Number of episodes
j. Max timesteps

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

3. Provide the reward graphs and your explanation for each result. In total, you

should have at least 3 graphs and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup

## In your report for Part II & III:

1. Briefly explain the tabular methods that were used to solve the problems. Provide their update functions and key features. What are the advantages/disadvantages?

2. Show and discuss the results after:
   - Applying SARSA to solve the environment defined in Part 1. Plots should include epsilon decay and total reward per episode.
   - Applying Q-learning to solve the environment defined in Part 1. Plots should include epsilon decay and 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 learned policy. Plot should include the total reward per episode.

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

4. Provide the analysis after tuning at least two hyperparameters from the list above. Provide the reward graphs and your explanation for each of the results. In total, you should have at least 6 graphs for each implemented algorithm and your explanations. Make your suggestion on the most efficient hyperparameters values for your problem setup.

### Tips on completing Part II & III

Recommended graphs that can help to debug the code

- Cumulative reward per episode. Ideally, we want this to increase linearly over time
- Total reward per episode during training.
- Total reward per episode during evaluation. Ideally, the graph is around the maxreward that the agent can get within the episode.
- Average number of timesteps per episode.
- Average number of times our agent receives a penalty by going into the ’bad’ state, if applicable.

## Bonus task

**n-step SARSA [5 points]**

Modify your SARSA and implement a 2-step bootstrapping SARSA. Compare the results with SARSA.

## ASSIGNMENT STEPS

## 1. Register your team

You may work individually or in a team of up to 2 people. The evaluation will be the same for a team of any size.

Register your team at UBLearns (UBlearns > Tools > Groups). Your teammates should be different for A1 & A2.

If you joined the wrong group, make a private post on piazza.

## 2. Submit checkpoint (Apr 27)

- Complete Part I & Part II.
- For the checkpoint, it is ok if SARSA algorithm is not fully converging. You can finalize and improve it for the finalsubmission.
- For your checkpoint submission, it is ok if your report is not fully completed. You can finalize it for the final submission.
- Submit the code in a .ipynb and your draft report in pdf at UBLearns > Assignments.
- The code of your implementations should be written in Python. You can submit multiple files, but they all need to be labeled clearly.
- Submit your file named as: UBIT TEAMMATE1_TEAMMATE2 _assignment3_checkpoint.zip (e.g., avereshc_ atharvap_ assignment3_checkpoint.zip).

## 3. Submit final results (May 11)

- 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: UBIT TEAMMATE1_TEAMMATE2 _assignment3_final.zip (e.g. avereshc_ atharvap_ assignment3_final.zip).
- Your Jupyter notebook should be saved with the results. Do not submit any python/JSON/HTML file, these files will not be graded.
- Include all the references that have been used to complete the assignment.
- If you are working in a team, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in the form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

| Team Member | Assignment Part | Contribution (%) |
|---|---|---|
| | | |
| | | |

## Academic Integrity

This project can be done in a team of up to two people. Your teammates should be different for A1&A2.

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.

## Late Days Policy

You can use up to 7 late days throughout the course that can be applied to any assignment related due dates. You do not have to inform the instructor, as the late submission will be tracked in UBlearns.

If you work in teams, the late days used will be subtracted from both partners. In other words, you have 4 late days and your partner has 3 late days left. If you submit one day after the due date, you will have 3 days and your partner will have 2 late days left.

## Important Dates

**Apr 27, Thu** - Checkpoint is Due

**May 11, Thu** - Final Submission is Due