

CS 4122: Reinforcement Learning and Autonomous Systems

Programming Assignment #4 (for Module 4)

Release date: 12th November, 2023, 11:59 pm (IST)

Due date: 3rd December, 2023, 11:59 pm (IST)

Maximum Score: 100 marks (this assignment is graded)

Read the following before you move forward:

1. This programming assignment is worth **10% of the total marks** of this course.
2. **Late policy:** You can be late by a maximum of 3 days. Your assignment will not be accepted more than 3 days after the due date. The **actual score** and **received score** (received score is actual score minus penalty for late submission) are related as follows:

$$\text{received score} = \begin{cases} \text{actual score} & ; \text{late by 0 days} \\ 0.9 \cdot \text{actual score} & ; \text{late by 1 day} \\ 0.7 \cdot \text{actual score} & ; \text{late by 2 days} \\ 0.5 \cdot \text{actual score} & ; \text{late by 3 days} \\ 0 & ; \text{late by more than 3 days} \end{cases}$$

3. **Plagiarism, if detected, no matter how big or small, will lead to a score of ZERO** for all the team member; no exceptions! **It is ok to take help from AI tools** or **talk to other teams**. But, **you cannot copy from another team or any AI tools**.
 - a. It is down right stupid to copy-paste from AI tools and think that is acceptable.
 - b. Also, it is a bad idea to keep open another teams work in front of you while you do the assignment. While you may think you are just taking help, your reports and your codes will get heavily influenced by the other team to the point that it will qualify as being plagiarized.
4. This is a team project. You have to do the project with the team that you selected in the excel sheet that was shared with you.
 - a. **Only one submission per team.**
 - b. Submission must be made using the google drive link that was shared with you.
 - c. Your submission in the google drive link must contain **ONLY FIVE files**:
 - i. report.pdf
 - ii. lunar_dataset.csv
 - iii. training.py
 - iv. The final trained model
 - v. testing.py

Read the deliverables carefully to understand what needs to be submitted in the report and in the Python codes. DON'T submit any other files. You MUST NOT zip/compress these five files. You MUST NOT put these five files in a subfolder inside the google drive link. Directly drop these five files in the google drive folder corresponding to your team.

Deep Q-Learning for Lunar Lander

In this programming assignment, you will be implementing **double Deep Q Learning** for the Lunar Lander environment of OpenAI Gym (https://www.gymnasium.dev/environments/box2d/lunar_lander/).

VERY IMPORTANT:

1. You cannot use existing Deep RL libraries like Keras RL, Stable Baselines, Tensorflow Agents etc. You can use deep learning libraries though.
2. You have to code **DQN architecture 1** in this assignment NOT DQN architecture 2.

If any of the above two rules are broken, **you will get ZERO for section 5 of this assignment. There will be NO MERCY in this regard.** I will keep on repeating the above two points a few times so that there is no room for mistake from your side.

Section 1: Sharing workload

Unfortunately, sharing workload in this assignment is quite tough. This is because there is only one main task which you will encounter in section 5, i.e. to write the code to train a double DQN model for lunar lander environment. **My sincere advice is that at least two team members should write their own code for section 5 INDEPENDENTLY.** This is because depending on the coding style and the data structures used, the training speed of two different codes can differ significantly.

Section 2: Useful resources

1. To clear the fundamentals of Deep Q-Learning, check the slides of lectures 26, 27, 29, 30, and 31. You will be implementing **Double Deep Q-Learning** in this assignment whose pseudocode is there in the end of the lecture slides mentioned above.
2. In this programming assignment folder, there is a Python script [CartPole_DQN_training.py](#). This Python script trains **DQN architecture 2** (you have to code **DQN architecture 1**) for the **cart pole environment** of OpenAI Gymnasium using **Keras/Tensorflow** libraries. Also, **this code DOES NOT use double DQN**. It just trains DQN using replay buffer and target network, i.e. **algorithm 3** in the slides of lectures 26, 27, 29, 30, and 31.
 - **IMPORTANT:** Pay attention to lines 47, 72, and 88 of this [CartPole_DQN_training.py](#). Try to understand the significance of [\(1-terminated_batch\)](#) in line 88.

Section 3: Installation

If you are using **Spyder** that comes with **Anaconda**, execute the pip command in **Anaconda prompt** to install **OpenAI Gymnasium** library and few other supporting libraries required for this assignment (for **Colab**, **pip** should be replaced with **!pip**):

[pip install gymnasium](#)

pip install swig

pip install gymnasium[box2d]

pip install pygame

2nd line: **swig** is a library to interface C/C++ libraries with Python. 3rd line: **Box2D** package of Gymnasium contains the Lunar Lander environment. 4th line: **pygame** is a library that is required for animation.

Few things to note:

1. Install OpenAI Gymnasium and NOT OpenAI Gym. Gym and Gymnasium are two different libraries that contains environments for reinforcement learning. Gymnasium is the newer version of Gym and supports more environments than Gym.
2. You need a **deep learning library** for this assignment. So, install either **Keras**, **Tensorflow**, or **Pytorch**. The YouTube video which I mentioned in section 2 uses Keras/Tensorflow.

Section 4: Get familiar with the Lunar Lander environment

There are two steps to get familiar with the lunar lander environment of OpenAI Gym.

Step 1: Go through the following webpage which explains the lunar lander environment:

https://www.gymnasium.dev/environments/box2d/lunar_lander/

There is a section titled **Arguments**. You may choose to skip that.

Deliverables (6 marks): Answer the following questions about the lunar lander environment:

1. How many actions are there in the action space? Very brief answer only.
2. Is it possible to fire both main engine and left/right orientation engine at the same time? Justify your answer.
3. How many observations are there? Very brief answer only.
4. How many observations are there in the observation space? Very brief answer only. NOTE: Questions 3 and 4 are different.
5. What are the sources of randomness for this environment? Answer should not exceed 7 lines.
6. While training a DQN, there is a “convergence criteria” after which we should stop training. What should be this convergence criteria according to the lunar lander webpage?

Your answers to the above questions **MUST** indicate that you read the webpage. Otherwise, you will get **zero** for that specific question. The answers to the above questions should be there **ONLY** in **report.pdf**.

Step 2: It's game time!!! You are given a Python script [LunarLander_PlayGame.py](#). You can run this to play the game using keyboard. **Don't start playing the game immediately!** Read through step 2, complete the deliverables, and then play the game.

The idea of this step is three folds. *First*, to get you acquainted with the feature of Gymnasium that allows you to control an environment using keyboard. *Second*, to get you comfortable playing the game as it is required for step 3 where we will do some data collection. *Third*, to get you to appreciate the

difficulty/ease of the game so that you can understand if the model that you will train in Section 3 is doing well.

CAUTION: If you run this in Jupyter notebooks like that in Kaggle and Colab, it may flag an error and also the graphics may not appear. Better use Spyder.

When you run `LunarLander_PlayGame.py`:

1. A window titled “pygame window” appears where you can play the game. You have to use **a**, **s**, and **d** keys to play the game.
2. Unless you click the × sign on the right side of the “pygame window”, episodes will run one after the other. **Clicking the × sign is the correct way to stop the game.** Get in the habit of clicking the × sign to stop the game rather than doing it abruptly in any other way. **In case, you abruptly stopped the game, execute the code in line 34 to close the pygame window.**
3. After every episode, the **total reward** will be displayed in the **console**.

Deliverables (4 marks): Consider line 27 of `LunarLander_PlayGame.py`. It uses a function named `gymnasium.utils.play.play`. The description of the function is given in the following webpage:

<https://www.gymnasium.dev/api/utils/>

Based on your reading of the above webpage, answer the following questions:

1. Associate the keys **a**, **s**, and **d** with the respective actions (fire main engine, do nothing, fire left orientation engine, fire right orientation engine).
2. In line 27, what is the function of `noop`?
3. In line 27, what is the function of `callback=total_reward_func`?
4. Suppose the game is too fast for you, what should you change in line 27 and how?

The answer to the above questions should be there **ONLY** in **report.pdf**.

Play the game until you can get a total reward of around 20-50 pretty consistently.

Step 3: Recall that Q-Learning and hence Deep Q-Learning is an **off-policy** algorithm. This means that we can use data collected from other resources, in this case from human actions, to train out Deep-Q Network (DQN). That is why it is important to some extent that you get comfortable playing the game before starting this step (this is not to be taken too seriously).

You are given a Python script titled `LunarLander_CollectOfflineData.py` and an empty csv file `lunar_dataset.csv`. The Python script is similar to that in Step 2. Just that it collects the data acquired during gameplay and saves it in the csv file. The csv file has the following columns:

Play #	State	Action	Reward	Next State	Terminated
--------	-------	--------	--------	------------	------------

The columns *State*, *Action*, *Reward*, and *Next State* needs no further explanation as it has been discussed time and again during lectures. *Terminated* is required because we are dealing with an episodic setup. *Play #* is the episode number.

Your task: Collect around 100 plays of the game in `lunar_dataset.csv`. You may collect less also but not too less. Please keep the following points in mind while collecting the dataset:

1. **You don't have to collect all 100 plays at once.** You can run [LunarLander_CollectOfflineData.py](#), play a few episodes, and then close the pygame window. When you run the Python script again, the new data that is collected will get **appended** onto the existing dataset.
2. **All the team mates can play the game and then concatenate the dataset.** This will speed-up data collection.
3. It is highly suggested that you use **×** **sign** on the right side of the pygame window to stop the game play. In case you close it any other way, you have to run lines 55 to 67 of the Python script on your own.

Deliverables: Submit `lunar_dataset.csv` that contains the data collected during game play. **No points** for this step. But, if you don't submit `lunar_dataset.csv`, you will **lose 10 points**.

Section 5: Train DQN Model

In this section you will train two **double DQN models** of **Architecture type 1**. The first double DQN model should be without the data collected in step 3 of section 4 and the second one uses the data. I am repeating the following again:

VERY IMPORTANT:

1. **You cannot use existing Deep RL libraries** like Keras RL, Stable Baselines, Tensorflow Agents etc. **You can use deep learning libraries** though.
2. You have to code **DQN architecture 1** in this assignment NOT DQN architecture 2.

If any of the above two rules are broken, **you will get ZERO for section 5 of this assignment. There will be NO MERCY in this regard.**

Deliverables (75 marks): You are given a Python script [training.py](#). This script contains the bare basic skeleton of the DQN training code along with a function that loads the data collected in step 3 of section 4. **You must NOT change the overall structure of the skeleton.** There are two functions in [training.py](#): [DQN_training](#) and [plot_reward](#). Your task is to write the code for these two functions. Few instructions about [DQN_training](#):

1. This function should implement a **double Q-learning** algorithm for the lunar lander environment. **DQN architecture must be type 1**, i.e. the DQN should accept a state-action pair as input and the output of the DQN should be the Q-value of the state-action pair given in the input. The pseudocode for double DQN is given in **Algorithm 5 (not Algorithm 4)** in the slides of lectures 26, 27, 29, 30, and 31. The output of the function is the **final trained predict DQN model**, and a Numpy array containing **total reward per episode**.
2. In function has an argument called [use_offline_data](#). If this variable is **True** then the data collected in step 3 should be used for training; else not.

- If `use_offline_data=False`, then it is business as usual, i.e. your code will be similar to that in the resources folder.
 - If `use_offline_data=True`, then we will initialize the replay buffer with the data collected offline. For the first E episodes, you will NOT append any data collected from the interaction with the environment onto the replay buffer. After E episodes, the data collected from the interaction with the environment should be appended to the replay buffer. E should not be greater than 100. The exact value of E is an hyperparameter. In many ways it depends on how good the data collected in step 3 is. If you got high total rewards in step 3, E can be high. In this regard note that one of the argument of the function `load_offline_data` is `min_score`. Only those episodes/play # will be loaded whose total reward is \geq `min_score`. So, the higher the `min_score`, the quality of data increases but the amount of data decreases. By default, `min_score` is set to $-\infty$ and hence all the episodes are loaded.
3. **The final trained model should be saved and submitted.** You should **submit only one model**, either for `use_offline_data=False` or `use_offline_data=True`. Submit the one that is performing better. **The size of your model should not be greater than 2 MB.**

The function `plot_reward` should plot the following in the same graph: (i) total reward per episode, and (ii) moving average of the total reward. The plots for both `use_offline_data=False` and `use_offline_data=True` should be included in report.pdf.

The following points may also be of significant help:

1. **IMPORTANT:** Remember that action is one of the inputs to the double DQN. Should the action be ordinal or one-hot-encoded?
2. For your own benefit, don't write two separate code for `use_offline_data=False` or `use_offline_data=True`. The difference between them is not much.
3. Don't forget to **save the model periodically** using an automated code. This is because your laptop, Kaggle/Colab notebooks can switch off (or go to sleep) if there is prolonged inactivity which is often the case when you are training a model for a long time. If you save your model, you can load it and start from where your progress stopped.
4. **Don't used any complicated neural network** model. It will take a lot of time to train it. A neural network model with 2-3 hidden layers and not more than 128 neurons per hidden layer is more than enough. In fact, 128 neurons is too much and so is the size of 2 MB mentioned move. The size of my model is less than 100 KB.
5. **GPU will NOT increase the training speed significantly.** This is one of the curses of Deep RL (unless you are using advanced techniques like multi-agent RL).

Section 6: Testing the final model

Deliverables (15 marks): Finally, you will write a code to test the final model that you have trained. You are given a Python script `testing.py`. This script is supposed to **show the animation of one complete episode** of lunar lander environment using the final model that you have trained in Section 5. `testing.py` contains clear instructions about writing the required code. Fill `testing.py` based on the instructions. You have to submit `testing.py`. I should be able to run and see the desired output in the click of a button.