CS 4122: Reinforcement Learning and Autonomous Systems

Programming Assignment # 5 (for Module 5)

Release date: 3rd December, 2024, 8:00 pm (IST)

Due date: 22nd December, 2024, 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:

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

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.
    - It is down right stupid to copy-paste from AI tools and think that is acceptable.

    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.
    - Only **one submission per team**.
    - Submission must be made using the google drive link that was shared with you.
    - **Your submission must contain ONLY FOUR files:**
        i. **report.pdf.**
        ii. **training_level1.py.**
        iii. **training_level2.py.**
        iv. **training_level3.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 FOUR files. **You MUST NOT put these four files in a subfolder inside the google drive link.** Directly drop these four files in the google drive folder corresponding to your team.

Using Stable Baselines

Section 1: Introduction

Unlike machine Learning and deep Learning that have well established Python libraries Scikit Learn, Tensorflow, Keras, and Pytorch, deep reinforcement learning (RL) does not that well established libraries. Among all the libraries for Deep RL, my understanding is that **Stable Baselines** is perhaps the most established library for Deep RL. Few **advantages** of Stable Baseline: *(i)* Fast training, *(ii)* Can handle various kind of OpenAI Gym environments., *(iii)* Easy to use. The only **disadvantage** according to me is that it only works with **Pytorch**. But, for this assignment you don't need a lot of Pytorch knowledge (if any).

This is less of an assignment and more of **guided learning** to accomplish the following goals:

1. Learning to **basics of Stable Baselines**.
2. Learning how to create **custom reward functions** using **wrappers** on OpenAI Gym.
3. Learning how to **deal with observations instead of states** using **wrappers** on OpenAI Gym.

In order to achieve these four goals, we will be dealing with a **variant** of the simplest Gym environment, the **Cartpole**. That said, in section 3, we will briefly deal with the **lunar lander** environment.

Section 2: Installation

If you are using **Spyder** that comes with **Anaconda**, execute the pip command in **Anaconda prompt** to install the libraries required for this assignment (for **Kaggle** or **Colab**, pip should be replaced with !pip):

```
pip install gymnasium

pip install swig

pip install gymnasium[box2d]

pip install gymnasium[classic-control]

pip install pygame

pip install torch

pip install stable-baselines3[extra]
```

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: **classic-control** package of Gymnasium contains the Cart Pole environment. 5th line: **pygame** is a library that is required for animation. 5th line: **pytorch** is a library for deep learning; stable baselines use Pytorch models. 6th line: **stable-baseline3** is a library for deep RL.

Section 3: Get familiar with Stable Baselines

This section has two objectives. *First*, it is a warm up exercise to get you familiar with stable baselines. *Second*, it shows how easy it is to use stable baselines compared to writing your own code for deep RL.

Coding tasks: You are given a Python file titled **training_level1.py** that contains most of the code required to train and test the same **lunar lander** environment that you coded in programming assignment 3. You have to write only four lines of code! Do the following **in order**:

1. Before you start the deliverables, make a copy of the original **training_ level1.py** that was given to you. I will refer to line numbers of the original **training_ level1.py** file throughout this section.

2. Learn how to train the DQN model of stable baselines by going through the "example" in this webpage: https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html

3. Do the following to **replicate** the code in the above website for lunar lander environment:
   a. Check comment in **line 4** of **training_ level1.py** and do the needful.
   b. Check comment in **line 29** of **training_ level1.py** and do the needful.

   Don't train the DQN model yet!

4. Scroll down in the above webpage. You will find a section titled "Parameters". Learn about the following parameters of the DQN class: *train_freq, learning_rate, learning_starts, batch_size, target_update_interval, buffer_size, exploration_initial_eps, exploration_final_eps*, *policy_kwargs*. A bit more about *policy_kwargs*:
   a. You will see a term called 'MlpPolicy'. This means "multi-layer perceptron" policy. So basically this is the Deep Neural network.
   b. *policy_kwargs* is used to specify the architecture of the neural network. To see how, go the following website and do Ctrl+F to search for "policy_kwargs": https://stable-baselines3.readthedocs.io/en/master/guide/examples.html
   c. To know more about adjusting the neural network architecture, check the following link: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html
      But I am not going to bother you with it because it requires some Pytorch knowledge which some of you may not have. I pointed out the link for educational purpose.

5. Based on what you learned in the above step, modify necessary parameters in the two lines of code that you wrote below **line 29** of **training_ level1.py**. How you set the training parameters is upto you. That said, **your neural network should contain only two hidden layers and not more than 64 neurons per hidden layer**. Now train the DQN model. I feel 200000 timesteps of training should be good enough if train_freq is 1. Training should take around 10 minutes.

6. After training the DQN model, you have to test it. To do so, check comment in **line 36** of **training_ level1.py** and do the needful. Now you can run this line of code that you just wrote to visualize the performance of your trained DQN model any number of times.

Deliverables (20 marks): Submit training_level1.py after you follow steps 3 and 5.

Section 4: A deeper dive

In this section we will learn more about Stable Baselines and also about OpenAI **Gymnasium**. In order to do so, we will use a custom environment called **CartPoleEnv_Continuous** that is given to you as a Python file **cartpole_continuous.py**.

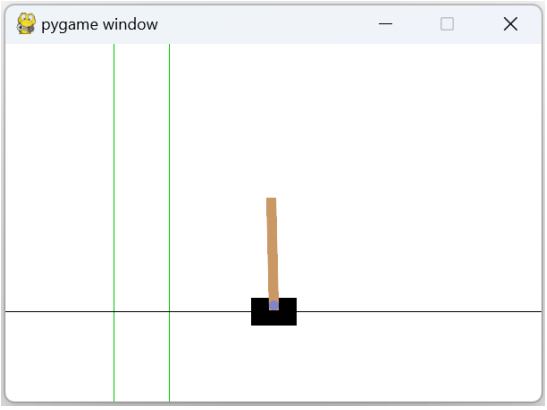
IMPORTANT: You must not change anything in **cartpole_continuous.py** throughout this assignment.

Go through the following subsections **in order**. **Not all subsections have deliverables!**

Subsection 4.1: Getting to know the custom environment

The custom environment is a variant of OpenAI Gymnasium's CartPole environment. Glance through the following webpage to learn about the CartPole environment:

https://gymnasium.farama.org/environments/classic_control/cart_pole/



A snapshot of the Pygame window for the custom environment is shown above for easy understanding. The custom environment is different from the original environment in the following ways:

1. The original environment has **discrete action space**, i.e. push right (1) or push left (0). The custom environment has **continuous action space**, i.e. a force between -10 to 10 Newtons.
2. In the original environment, the observations are the four states. In the custom environment:
    a. The observation is a **noisy measurement** of the four states.
    b. There is a **fifth state**, the **desired cartpole position**.
3. In the original environment, the cartpole position should be between -2.4 to 2.4 and the pole angle should be between -12° to 12° to get a reward of 1. Otherwise, the reward is 0. In the custom environment, the following conditions should be meet to get a reward of 1:
    a. As usual, the pole angle should be between -12° to 12°.
    b. **The cartpole position should be between $\pm$0.25 of the desired cartpole position**. This is shown using the green lines in the above figure.

    Otherwise, the reward is 0. **The termination/truncation criteria of both the original and the custom environment is the same.** When you are initializing the custom environment, you have to select a parameter called **control_mode**. **control_mode** can assume the following values:
        a. *control_mode='regulatory'*: The desired cartpole position will always be **0**.
        b. *control_mode='setpoint'*: The desired cartpole position will be a **random number between -1.5 to 1.5** that is selected from an uniform distribution.

Subsection 4.2: Dealing with observations (instead of states) and continuous action space

The custom cartpole environment is challenging for two reasons:

1. We have to deal with observation instead of states. So, in essence it is a **partially observable Markov decision process**. We discussed multiple times during lecture that in such cases we just can't rely on the current observation to make "good" decisions; we need **current observation** and **past observation**. Off course you can write your own code to store the past observations but OpenAI Gymnasium makes this very easy. There is a **wrapper** called **FrameStack** in OpenAI Gymnasium that creates a new environment whose observation is the past $Nframes$ observations of the input environment. You can find more about **FrameStack** and an **example** to use it in the following link:
   https://gymnasium.farama.org/api/wrappers/observation_wrappers/

   Note: For the sake of your knowledge, you can learn more about wrappers and different kind of wrappers in OpenAI Gymnasium.

2. We have to deal with continuous action space. As we discussed during lecture Deep Deterministic Policy Gradient (DDPG). There are many others. There is a very neat table in the following link that concisely describes the capabilities of various deep RL algorithms:
   https://stable-baselines3.readthedocs.io/en/master/guide/algos.html

   Those algorithms that has a tick mark in the "Box" column can deal with continuous action space. Now, just like DQN, you can learn about DDPG from the following link (**check the example** and the **parameters of DDPG**):
   https://stable-baselines3.readthedocs.io/en/master/modules/ddpg.html

Coding tasks: You are given a Python file titled **training_level2.py** that contains most of the code required for this section. You have to write around nine lines of code! Do the following **in order**:

1. Before you start the deliverables, make a copy of the original **training_ level2.py** that was given to you. I will refer to line numbers of the original **training_ level2.py** file throughout this section.

2. You will see something called **control_mode** in **line 47** of **training_level2.py**. For this subsection, **control_mode** should be set to **'regulatory'**.

3. Based on your understanding of **FrameStack**, check comment in **line 48** of **training_ level2.py** and do the needful.

4. Based on your understanding of the **example of DDPG**, check comments in **lines 5 and 52** of **training_ level2.py** and do the needful.

5. Now it is time to train DDPG. How you set the training parameters of DDPG is upto you. That said, **your neural network should contain only two hidden layers and not more than 64 neurons per**

hidden layer. Now train the DDPG model. I feel 50000 timesteps of training should be good enough. Training should take around 5 minutes (even for $Nframes$ as high as 10).

6. After training the model, you have to test it. To do so, check comment in **lines 62 and 65** of **training_level2.py** and do the needful. Now you can run these lines of code that you just wrote to either 'analyze' or 'visualize' the performance of your trained model.

7. Train and analyze DDPG models (analyze model means to calculate average total reward of the trained model over 100 episodes) for $Nframes = 1, 2, 4, 6, 8, 10$. In **report.pdf**, plot a graph of $Nframes$ vs average total reward. You should see an increasing trend.

Deliverables (40 marks): Submit training_level2.py after you follow steps 3, 4, 5, and 6. Also, report.pdf that contains the graph mentioned in step 7.

Subsection 4.3: Setpoint control

To do: We start this sub section by going back to **training_level2.py** that you have already coded. Now, set **control_mode** to **'setpoint'**. Then train, analyze, and visualize the performance of the DDPG model. How is the reward compared to **'regulatory' control_mode**?

Most likely the performance of the DDPG model is bad when **control_mode** is **'setpoint'**.

Coding task: You are given a Python file titled **training_level3.py**. This is exactly the same as **training_level2.py** with one difference: there is a **Custom_Wrapper** class that you can use to implement a **modified reward** function. In fact, you can import all your code from the previous sub section to **training_level3.py**. Your task in this section is simple; train a deep RL in order to get good performance for **control_mode='setpoint'**. You are not limited to any approach but you MUST use Stable Baselines only. Some possible approaches can be:

1. Try other deep RL models like TD3, PPO, SAC etc.
2. Try to code your own custom reward function. The skeleton of implementing custom reward function is given in **training_level3.py**. You just have to replace **modified_reward = reward** in **line 54** with your own code to calculate a modified reward.

Why code a custom reward?: The reason is similar to why in classification problems you don't optimize accuracy directly; you optimize a loss function. In other words, it is easier to optimize loss function compared to accuracy. Similarly, for this environment, optimizing the actual reward may be tough. Hence, you design a custom reward function such that:

A. It is easy to optimize.
B. Optimizing the custom reward function indirectly leads to optimizing the true reward function.

Why optimizing true reward function is difficult?: It is because the reward for **control_mode='setpoint'** is SPARSE, i.e. the agent gets **reward=1** only after reaching very close to the desired cartpole position. Until then, the agent gets **reward=0** and hence does not know if the actions it is taking is correct. Sparse reward is a challenging aspect of RL.

A viable choice of custom reward function is:

$$r(x, v, \theta, \omega, x_o) = \begin{cases} -(x - x_o)^2 - \alpha_v v^2 - \alpha_\theta \theta^2 - \alpha_\omega \omega^2 & ; \ terminated = False \\ -\alpha_T & ; \ terminated = \ True \end{cases}$$

where $\alpha_v, \alpha_\theta, \alpha_\omega, \alpha_T \geq 0$ are hyperparameters to be set. Please remember you are not limited to this reward function. Neither are you limited to any approaches other than that you have to use Stable Baselines.

Deliverables (40 marks): Submit training_level3.py that contains the FINAL code to train a deep RL model when control_mode='setpoint'. In report.pdf, you have to briefly describe the final model and its training process (not more than a page).