ARTS1425 2024-2025 Programming Assignment

Due: 2025/01/09 23:59

Guidelines

- **Code:** It is recommended to use Jupyter Notebook (**.ipynb**) for your submission as it allows for clear presentation of both code and outputs. Ensure that your code is executable.
- **Submission:** Please compress all your codes, files and models into ZIP file as **"name_id.zip"**, and send it to **yehw2024@shanghaitech.edu.cn**. Late submissions will not be accepted.
- **Academic Integrity:** Ensure that all work submitted is **your own**. Plagiarism or any form of academic dishonesty will result in a score of 0. (PA is worth 20% of your final grade.)

Prerequisites

This assignment focuses on implementing and evaluating Deep Reinforcement Learning (DRL) algorithms using the OpenAI Gym environment. The goal is to explore and understand how DRL can be applied to a variety of tasks within a simulated environment, where agents learn through interaction and feedback. We will be utilizing Gym, a widely used toolkit for developing and comparing reinforcement learning algorithms, which provides a variety of environments for testing different models. Through this assignment, you will gain hands-on experience with reinforcement learning algorithms, which be valuable for your final project.

1. Create a virtual environment using **Anaconda**, with Python 3.10:

   ```
   conda create -n arts1425 python==3.10
   conda activate arts1425
   ```

2. Install the project dependencies by running:

   ```
   pip install -r requirements.txt
   ```

3. Install **PyTorch** based on your computer system to accelerate your code.
4. Familiarize yourself with the frequently-used functions of the Gym library, including:
   - **gym.make(env_name)**: Creates an environment previously registered in Gym.
   - **env.reset()**: Resets the environment to an initial internal state.
   - **env.step(action)**: Run one timestep of the environment using the agent action.
5. There are three **.ipynb** files in the assignment folder, corresponding to problem 1, 2, and 3.
   - dqn.ipynb (Problem 1)
   - reinforce.ipynb (Problem 2)
   - actor_critic.ipynb (Problem 3)

   You need to complete the above three files first.
6. You can directly evaluate and visualize the model's results by running:

   ```
   python evaluate.py
   ```

Problem 1: DQN

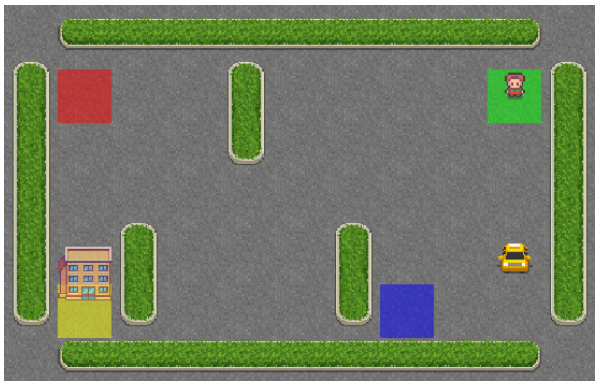

Figure 1: Taxi-v3

The Taxi-v3 environment is a grid-based problem with the goal of moving a taxi to pick up and drop off a passenger at designated locations. There are four designated locations: Red, Green, Yellow, and Blue, in a 5x5 grid world. The taxi starts at a random square, and the passenger starts at one of the designated locations. The goal is to move the taxi to the passenger's location, pick up the passenger, move to the passenger's desired destination, and drop off the passenger. Once the passenger is dropped off, the episode ends. For a more detailed description for action space, observation space and rewards, please refer to the official documentation of **Taxi-v3**.

Please implement the **DQN** algorithm on this environment to perform iterative training, record the process, plot the curve of the reward value against the number of episodes and save the training model. If the training fails, provide an explanation for the failure.

The algorithm contains at least the following core structure, for example:

```
class ReplayBuffer:
    def __init__(self, capacity)
        ...

    def add(self, state, action, reward, next_state, done):
        ...

    def sample(self, batch_size):
        ...

class QNet:
    def __init__(self ,state_dim, action_dim)
        ...

    def forward(self, x):
        ...

class DQN:
    def __init__(self, state_dim, action_dim):
        ...

    def take_action(self, state):
        ...

    def update(self, ...):
        ...
```

Problem 2: REINFORCE

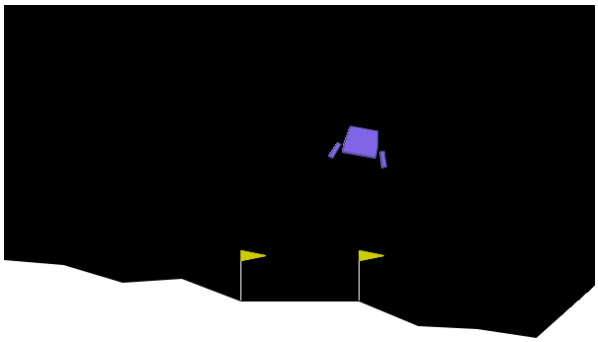

Figure 2: LunarLander-v3

The **LunarLander-v3** environment is a game where the goal is to land a spacecraft safely on a lunar surface. The agent controls a lander with three thrusters and must navigate the lander to a designated landing pad. The environment consists of a 2D world, with a lander that has to be maneuvered through gravitational forces and various terrain conditions. The agent receives continuous state observations that include the lander's position, velocity, angle, and angular velocity. The objective is to minimize the fuel used while ensuring a safe landing. A successful landing is achieved when the lander touches down on the landing pad without tipping over or crashing.

Please implement the **REINFORCE** algorithm on this environment to perform iterative training, record the process, plot the curve of the reward value against the number of episodes and save the training model. If the training fails, provide an explanation for the failure.

The algorithm contains at least the following core structure, for example:

```
class PolicyNet:
    def __init__(self ,state_dim, action_dim)
        ...

    def forward(self, x):
        ...

class REINFORCE:
    def __init__(self, state_dim, action_dim):
        ...

    def take_action(self, state):
        ...

    def update(self, ...):
        ...
```

Problem 3: Actor-Critic

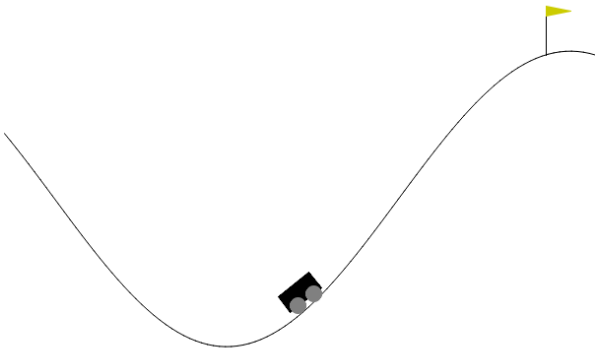

Figure 3: MountainCarContinuous-v0

The **MountainCarContinuous-v0** environment is a continuous control problem where the goal is to drive a car up a steep mountain to reach the flag at the top. The car starts at the bottom of the valley, between two hills, and the objective is to use the car's continuous acceleration to climb the mountain. The agent can apply a continuous force in either direction (accelerating left or right) to change the car's velocity and attempt to reach the goal. However, due to the steep slopes, the car cannot directly drive up the hill from the starting point and must first build momentum by driving back and forth between the hills. The episode ends when the car reaches the flag at the top of the mountain.

Please implement the **Actor-Critic** algorithm on this environment to perform iterative training, record the process, plot the curve of the reward value against the number of episodes and save the training model. If the training fails, provide an explanation for the failure.

The algorithm contains at least the following core structure, for example:

```
class PolicyNet:
    def __init__(self ,state_dim, action_dim)
        ...

    def forward(self, x):
        ...

class ValueNet:
    def __init__(self ,state_dim, action_dim)
        ...

    def forward(self, x):
        ...

class ActorCritic:
    def __init__(self, state_dim, action_dim):
        ...

    def take_action(self, state):
        ...

    def update(self, ...):
        ...
```

Hints

Record the Process

For better observing the training process, it is recommended to use **tqdm** library.

```
Iteration 0: 100%|██████████| 50/50 [01:04<00:00,  1.30s/it, episode=50, return=-169.732]
Iteration 1: 100%|██████████| 50/50 [02:57<00:00,  3.54s/it, episode=100, return=-89.984]
Iteration 2: 100%|██████████| 50/50 [02:42<00:00,  3.25s/it, episode=150, return=-133.620]
Iteration 3: 100%|██████████| 50/50 [02:16<00:00,  2.74s/it, episode=200, return=-80.833]
Iteration 4: 100%|██████████| 50/50 [01:44<00:00,  2.09s/it, episode=250, return=-114.135]
Iteration 5: 100%|██████████| 50/50 [02:55<00:00,  3.51s/it, episode=300, return=-75.560]
Iteration 6: 100%|██████████| 50/50 [02:09<00:00,  2.59s/it, episode=350, return=-171.444]
Iteration 7: 100%|██████████| 50/50 [02:02<00:00,  2.44s/it, episode=400, return=-10.854]
Iteration 8: 100%|██████████| 50/50 [01:29<00:00,  1.80s/it, episode=450, return=-55.924]
Iteration 9: 100%|██████████| 50/50 [02:02<00:00,  2.44s/it, episode=500, return=-14.461]
```

Figure 4: Example Training Process

Plot the Curve

For creating high-quality plots, it is recommended to use **matplotlib** library.

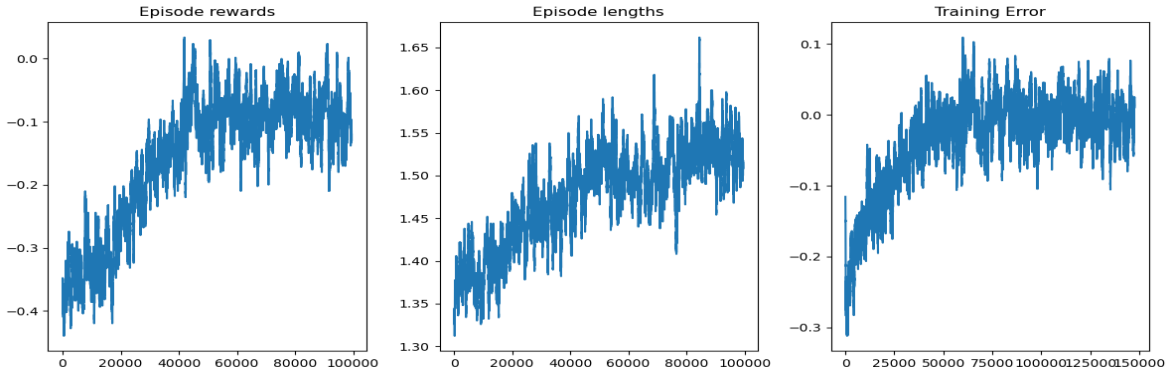

Figure 5: Example Curves

Save the Model

You should save the trained model locally using a method such as **torch.save()**. The folder contains a model trained with DQN algorithm after 500 episodes called **"dqn_lunalander.pth"**. You can just change the **"env_name"** and **"model_path"** in **"evaluate.py"** to observe and evaluate the model's performance. Model's name should follow: "[algorithm]_[environment].pth".

The final assignment grade will be based on both the code you have written (convert your .ipynb files into PDF files, ensuring that the code execution process and outputs are preserved) and your model's performance (average reward for 10 random games) in the three environments.