Homework 7

Release: 05/17/2022 Due: Wed. 05/25/2022, 11:59 PM

- You are **allowed** to consult any external resources, but you must cite them. You are also **allowed** to discuss with each other, but you need to acknowledge them. However, your submission must be your own work; specifically, you must **not** share your code or proof.
- Your submission includes a PDF file containing the solutions and results; and a zip file containing your source code.
- This homework is worth 17/100 of your final grade.
- This homework itself contains 38 points.
- It is highly recommended that you begin working on this assignment early.

Problem 1. GAE. (4pts)

1. Define $\delta_{t,h} = e_{t,h} - v_\omega(s_t) = \sum_{i=t}^{t+h} \gamma^{i-t} \cdot r_i + \gamma^{h+1} \cdot v_\omega(s_{t+h+1}) - v_\omega(s_t)$. The General Advantage Estimate (GAE) estimates $A^{\pi_\theta,\gamma}(s_t, a_t)$ by:

$$\hat{A}^{\pi_\theta,\gamma}_{\mathrm{GAE}(\lambda)}(s_t, a_t) = (1-\lambda) \sum_{h=0}^{\infty} \lambda^h \delta_{t,h}$$

   Please prove that $\hat{A}^{\pi_\theta,\gamma}_{\mathrm{GAE}(\lambda)}(s_t, a_t)$ also equals to $\sum_{h=0}^{\infty} (\gamma\lambda)^h \delta_{t+h,0}$, which is the way we calculate in program. (3pts)

2. We have $\hat{A}^{\pi_\theta,\gamma}_{\mathrm{GAE}(0)}(s_t, a_t) = r_t + \gamma v_\omega(s_{t+1}) - v_\omega(s_t)$ and $\hat{A}^{\pi_\theta,\gamma}_{\mathrm{GAE}(1)}(s_t, a_t) = \sum_{h=0}^{\infty} \gamma^h r_{t+h} - v_\omega(s_t)$. How to interpret the role of λ? How to interpret the role of γ? (1pt)

Problem 2. PPO & RND (34 pts).

In this problem, we will play with Proximal Policy Optimization (PPO).

1. We have attached an implementation of PPO from OpenAI Spinning Up. Please try to read and play with the implementation, and then answer the following questions (14pts):

   - In *core.py*, we have the implementation of both `actor` and `critic`. What are the roles of `actor` and `critic` respectively? What are their input and output? Please describe the function of `MLPActorCritic::step()` (2pts).
   - There are two types of `actor` for both continuous and discrete actions. How are they implemented respectively? What is the difference in network design? (1pt)
   - In `MLPGaussianActor`, we don't use the network to predict the std of a normal distribution. How do we determine the std? Is there anything to take care of when tuning our network? (1pt)
   - What's the role of GAE-Lambda? How do we implement it and tune the parameter? (1pt)
   - When collecting experiences, how do we handle the reward/value of the last step of a trajectory? There are two cases. (1pt)

- For a collected trajectory, how to calculate the return? How to calculate the advantage? Please note that the implementation uses an advantage normalization trick. What's the potential benefit of this trick? (2pts)
- How do we update the `critic`? What's the loss function? (1pt)
- How do we update the `actor`? What's the loss function? (1pt)
- How to understand clipping ratio for the advantage? What is its motivation? (1pt)
- How to understand the KL-based early stopping trick? What's the potential benefit? (1pt)
- Please note that PPO is an on-policy algorithm, and we only use the latest collected trajectory to update the model. Why can't we use all the history to update like in DQN? (1pt)
- The implementation uses MPI to parallel the program. What's the main part we want to accelerate? Why is it more important in PPO than in DQN? (1pt)

2. In this part, you need to solve the PointMaze task with PPO. As shown in Fig. 1, a blue point, with an arrow indicating its direction, is moving in a "CSE291E maze", where the goal is the origin ($x = 0, y = 0$). In each episode, the point starts from a random location and has up to 1,000 steps to reach the goal. If the length of an episode is l, you will get a score of $1000 - l$ for this episode.

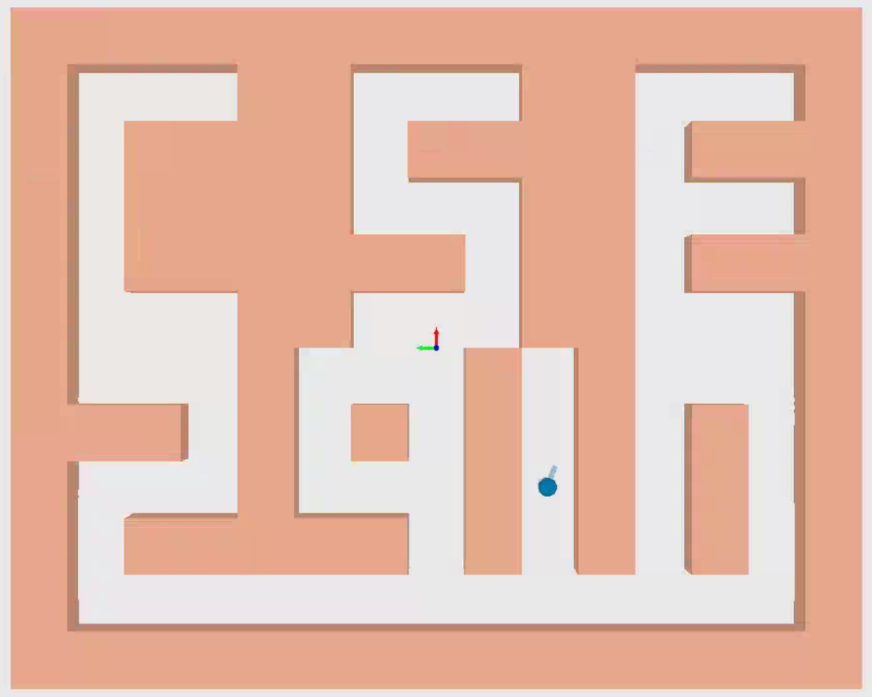

Figure 1: PointMaze.

Here are some of the basic information about the task:

- **observation**: The point has 3 DoFs: 2D position (x, y) and rotation θ. The observation is the state vector $[x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]$.
- **action**: The action is a 2D vector, and each element belongs to $[-1, 1]$. The first element controls the movement along the arrow direction, while the second element controls the rotation of the arrow. For the details of the transition, please check the function `PointMazeEnv::step()`.

- **reward**: You need to design a reward to solve the task. You are allowed to use heuristics about the map to design your reward.
- **done**: If $x^2 + y^2 \leq 0.5^2$ or 1000 steps are used up, the episode will terminate.

Based on the provided implementation, you need to design your reward function and tune your network. We have pointed out some parts that you may want to change in `point_maze.py`, `ppo.py`, and `core.py` (marked as "TODO"). You are also allowed to add more techniques that may lead to better performance. **However, you are NOT allowed to:**

- Directly control the point. The action should come from the output of your PPO policy network.
- Load the test cases during training. We may have additional test cases after your submission.
- Call `setqpos()`, `setqvel()`, `setqacc()`, `setqf()`, and etc.
- Modify the evaluate function and the environment except for the reward part.

After training your network, please run the evaluate function `evaluate()` in `ppo.py`. It will test your implementation for 200 cases and return a total score. You may want to use only a single thread to run the evaluation.

The credits for the PointMaze task are divided into three parts:

- (6pts) Your credits for this part is based on your total score:
  - 1pt: above 20,000.
  - 2pts: above 40,000.
  - 3pts: above 60,000.
  - 4pts: above 80,000.
  - 5pts: above 100,000.
  - 6pts: above 120,000.
- (8pts) Please submit your score to our leaderboard (https://forms.gle/MnUnYjP98Th4LF4aA) before the deadline. Your credit for this part is based on your rank on the leaderboard. Specifically, rank 1 gets 8pts, rank 2 gets 7.7pts, rank 3 gets 7.4pts.....
- (3pts) Please submit a detailed report with your modification, training curve, final score, evaluation screenshot, and how to run your code. You also need to submit your code and the trained model.

Please make sure your code is easy to run and your report is clear. Otherwise, you may lose points.

Hints: 1. You may want to start from some simple cases to check the correctness. 2. You may want to use `env.render()` to visualize.

3. In deep reinforcement learning, we typically add some intrinsic rewards to encourage exploration. Please implement random network distillation (RND) as an exploration bonus in your PointMaze task. Please compare and analyze the results w/ and w/o RND. In your report, please include your implementation and discussion. (3pts)

   Paper reference: https://arxiv.org/pdf/1810.12894.pdf