535515 Spring 2023: Reinforcement Learning **(Practice Problems)**

Homework 3: Soft Actor Critic

Submission Guidelines: Your deliverable shall be a PDF file, which could be either handwritten or typeset in either LaTeX or other word processors (e.g., Google Documents or MS Words). Please submit your deliverable via E3.

Problem 1 (Soft Actor Critic for Continuous Control) (100 points)

In this problem, we will take a deeper look at the actual implementation of Soft Actor Critic (SAC) algorithm. Specifically, we use the minimal implementation of SAC from the CleanRL framework (https://github.com/vwxyzjn/cleanrl) to further investigate the various important aspects of SAC. Please first take a look at the attached file **sac_continuous_action.py** and answer the following questions. For ease of exposition, the pseudo code of SAC is provided as below.

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
    Execute a in the environment
    Observe next state s', reward r, and done signal d to indicate whether s' is terminal
    Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
    If s' is terminal, reset environment state.
    if it's time to update **then**
        for j in range(however many updates) **do**
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

            Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2 \qquad \text{for } i = 1, 2$$

            Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)\middle| s\right)\right),$$

            where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.
            Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$$

        end for
    end if
until convergence

Figure 1: The pseudo code of the SAC algorithm.

(a) (Actor Network of SAC) As shown by the source code, the first major class is **Actor**. There are three salient designs that are not completely addressed in the pseudo code:

- (1) The output layer the actor network produces the mean and the logarithm of the standard deviation (cf. Lines 112-113 and Lines 122-144 in **sac_continuous_action.py**). Why is this needed?

- (2) We mentioned the "reparamterization trick" in Lecture 25. Could you carefully explain how this trick is implemented in SAC? (Hint: Lines 132-144 in **sac_continuous_action.py**)
- (3) In Lines 141-143 of **sac_continuous_action.py**, the actor network enforces some additional manipulations on the actions. Why is this needed? Is there any potential issue resulting from these manipulations?
- (4) Could you write down the actual loss function used for the update the actor network in this implementation? (Hint: Lines 264-274)

(b) (Soft Q Network of SAC) Another important class is **SoftQNetwork**. There are also several important tricks integrated with SAC.

- (5) It appears that SAC uses two soft Q networks for the critic (in both the pseudo code and the python code). Why is this needed? Could you explain what issue this "double Q" manages to address?
- (6) Based on (4), could you write down the loss function used for the update the two soft Q networks? (Hint: Lines 243-260)

(c) (Main Training Procedure of SAC) Next, let's take a closer look at the training part of SAC.

- (7) It is known that SAC is inherently an "off-policy" RL algorithm. Could you point out which part of the code demonstrates that SAC learns in an off-policy manner? What is the effective "behavior policy" in SAC?
- (8) In this implementation, there is an "auto-tuning" scheme for the temperature parameter α. Could you explain how this auto-tuning scheme works? (Hint: Please see Lines 276-284. You may also refer to the extended version of the SAC paper at https://arxiv.org/pdf/1812.05905.pdf)