535514 Spring 2025: Reinforcement Learning **(Due: 2025/05/23, 21:00)**

Homework 3: Soft Actor-Critic

Submission Guidelines: Your deliverables shall consist of 2 separate files – (i) A PDF file: Please compile all your write-ups into one .pdf file (photos/scanned copies are acceptable; please make sure that the electronic files are of good quality and reader-friendly); (ii) A zip file: Please compress all your source code (including **sac.py** and **sac_halfcheetah.py**) into one .zip file. Please submit your deliverables via E3.

Problem 1 (Soft Actor-Critic for Continuous Control) (20+15+15+25+25=100 points)

In this problem, we will take a deeper look at the actual implementation of Soft Actor-Critic (SAC) algorithm. Please first take a look at the attached file **sac.py** and answer the following questions and finish the implementation in **sac.py**. For ease of exposition, the pseudo code of SAC is provided as below.

(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:

- The output layer of the actor network produces the mean and the logarithm of the standard deviation (Lines 44-50 and Lines 57-73 in **sac.py**). Why is this needed?
- We mentioned the "reparamterization trick" in Lecture 11. Could you carefully explain how this trick is implemented in SAC? (Hint: Lines 66-67 in **sac.py**)
- In Lines 71-72 of **sac.py**, the calculation of log probability at the actor network enforces some additional manipulations. Why is this needed? (Hint: You may check Appendix C of the original SAC paper https://arxiv.org/abs/1801.01290)

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

- 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 "twin Q" manages to address? (Hint: This trick already appears in TD3)
- Based on the above, could you write down the exact mathematical expression of the loss function used for the update the two soft Q networks?

(c) (**Alpha Network of SAC**) The vanilla SAC presumes using a fixed temperature parameter α for the entropy bonus. Subsequently, the enhanced SAC in https://arxiv.org/abs/1812.05905 introduced an auto-tuning mechanism of α.

- This auto-tuning scheme is implemented in Line 255 of **sac.py**). Can you explain why this is implemented in this way? (Hint: You can refer to Section 5 of https://arxiv.org/abs/1812.05905)

Algorithm 1 Soft Actor-Critic Algorithm

Initialize:
Policy network $\pi_\phi(a|s)$
Q-networks $Q_{\theta_1}(s,a)$, $Q_{\theta_2}(s,a)$
Value network $V_\psi(s)$ and target value network $V_{\bar{\psi}}(s)$ with $\bar{\psi} \leftarrow \psi$
Replay buffer $\mathcal{D}$
for each iteration **do**
  for each environment step **do**
    Sample action $a_t \sim \pi_\phi(\cdot|s_t)$
    Execute a_t, observe reward r_t, and next state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
  end for
  for each gradient step **do**
    Sample minibatch $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ from $\mathcal{D}$
    Update Q-networks:
    Compute target value:

$$y_i = r_i + \gamma V_{\bar{\psi}}(s'_i)$$

    Minimize loss:

$$\mathcal{L}_{Q_j} = \frac{1}{N}\sum_i \left(Q_{\theta_j}(s_i, a_i) - y_i\right)^2, \quad j = 1, 2$$

    Update value network:
    Sample $a_i \sim \pi_\phi(\cdot|s_i)$, compute $\log \pi_\phi(a_i|s_i)$
    Compute soft Q estimate:

$$\hat{Q}_i = \min(Q_{\theta_1}(s_i, a_i), Q_{\theta_2}(s_i, a_i))$$

    Minimize value loss:

$$\mathcal{L}_V = \frac{1}{N}\sum_i \left(V_\psi(s_i) - (\hat{Q}_i - \alpha \log \pi_\phi(a_i|s_i))\right)^2$$

    Update policy network:

$$\mathcal{L}_\pi = \frac{1}{N}\sum_i \left(\alpha \log \pi_\phi(a_i|s_i) - Q_{\theta_1}(s_i, a_i)\right)$$

    Soft update for target value network:

$$\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$$

  end for
end for

(d) (**SAC on Pendulum-v1**) Next, let's finish the implementation of **sac.py**. Specifically, you need to implement the following three things in PyTorch:

- Computing the Q loss in the **update_model** function (about 5-10 lines).
- Computing the V loss in the **update_model** function (no more than 5 lines).
- Computing the actor loss in the **update_model** function (no more than 5 lines).

We start by solving the simple "Pendulum-v1" problem (https://www.gymlibrary.dev/environments/classic_control/pendulum/) using the SAC algorithm. Please briefly summarize your results (including the snapshots of Weight & Bias record) in the report and document all the hyperparameters (e.g. learning rates, NN architecture, and batch size) of your experiments. (Note: Pendulum-v1 is a rather basic environment mostly for the purpose of sanity check. As a result, typically it would take no more than 300 episodes to reach a well-performing policy of score above -130)

(e) (**SAC on HalfCheetah-v5**) Based on your implementation for (d), please adapt your SAC algorithm to solve the "HalfCheetah-v5" locomotion task in MuJoCo (https://gymnasium.farama.org/environments/mujoco/half_cheetah/). Save your code in another python file named **sac_halfcheetah.py**. Please add comments to your code whenever needed for better readability.

Train your Halfcheetah under SAC for 1 million environment steps and reach a score of at least 7000 within 1 million steps. Again, briefly summarize your results in the report and document all the hyperparameters of your experiments. (Note: As HalfCheetah is a more challenging environment than Pendulum, it would take a bit more training time to reach a well-performing policy, and it might require some efforts to tune the hyperparameters, e.g., learning rates and the batch sizes. That being said, under SAC, it is typically expected that Halfcheetah can achieve a score of 7000-12000 in 1 million training steps. You may use the hyperparameters suggested by https://arxiv.org/pdf/1812.05905 as a starting point)