Homework 4

6501 Reinforcement Learning (Spring 2024)

Submission deadline: 11:59pm, May 12

Rules

- Collaboration is allowed but must be stated in precision per sub-problem. For example, comment in the sub-problem which idea is a result of a discussion with whom.
- Leveraging large language models is allowed but must be stated. Describe how you use them or provide precise prompts that you use.
- Please put all proofs (typed in latex) and figures for experiments in a single pdf file. Submit the pdf to the Question 1 in Gradescope. Put all codes in a single .py file, and submit it to Question 2 in Gradescope.

1 Monotonic Policy Improvement

In this problem, we will prove that policy gradient or actor-critic can produce monotonic policy improvement when the value estimation is accurate.

As mentioned in the class, policy gradient (PG), natural policy gradient (NPG), and actor-critic are all based on the following update:

$$\theta_{k+1} \leftarrow \operatorname*{argmax}_{\theta} \left(V^{\pi_\theta}(\rho) - V^{\pi_{\theta_k}}(\rho) - \frac{1}{\eta} D(\theta, \theta_k) \right) \tag{1}$$

for some distance function D that measures the distance between θ and θ_k, where ρ is the initial state distribution.

(a) (5%) Show that if Eq. (1) can be performed exactly, then $V^{\pi_{\theta_{k+1}}}(\rho) \geq V^{\pi_{\theta_k}}(\rho)$. (Hint: a distance function should satisfy $D(x, y) \geq 0$ for all x, y, and $D(x, x) = 0$ for all x).

In *on-policy* PG and NPG and actor-critic, we use either unbiased estimators or biased but low-variance estimators to approximate the objective in Eq. (1). This allows us to approximately perform Eq. (1). In *off-policy* actor-critic (Page 14 here), however, we mentioned that our objective becomes quite different from Eq. (1) due to the state distribution mismatch between the current policy π_{θ_k} and behavior policy $\hat{\pi}$. Furthermore, this state distribution mismatch is uneasy to be corrected through importance weight because we do not know the ratio $d_\rho^{\pi_{\theta_k}}(s)/d_\rho^{\hat{\pi}}(s)$. This makes the monotonic improvement property shown in (a) no longer holds in general.

In the tabular case, however, we can still show monotonic improvement. To get an intuition why this holds, recall that in the full-information "policy iteration" algorithm (Page 44 here), we simultaneously perform policy updates on all states, which resembles having a behavior policy that produces a uniform distribution over states, i.e., $d_\rho^{\hat{\pi}} = \text{Uniform}(\mathcal{S})$. This $d_\rho^{\hat{\pi}}$ is also not equal to the occupancy measure of the current policy $d_\rho^{\pi_k}$, but we can still show that π_k converges to the optimal policy. This demonstrates that state distribution mismatch in policy update is not an issue in the tabular case (it just affects the convergence rate).

Below, we formalize this. By Page 14 here, off-policy actor-critic would approximate the following update in the tabular case:

$$\pi_{k+1} \leftarrow \operatorname*{argmax}_{\pi} \left(\sum_s d_\rho^{\hat{\pi}}(s) \left(\sum_a \left(\pi(a|s) - \pi_k(a|s)\right) Q^{\pi_k}(s, a) - \frac{1}{\eta} D(\pi(\cdot|s), \pi_k(\cdot|s)) \right) \right) \tag{2}$$

for some behavior policy $\hat{\pi}$ and some distance function D.

(b) (5%) Argue that Eq. (2) is equivalent to

$$\pi_{k+1}(\cdot|s) \leftarrow \operatorname*{argmax}_{\pi(\cdot|s)} \left(\sum_a \left(\pi(a|s) - \pi_k(a|s)\right) Q^{\pi_k}(s,a) - \frac{1}{\eta} D(\pi(\cdot|s), \pi_k(\cdot|s)) \right) \tag{3}$$

as long as $d_\rho^{\hat{\pi}}(s) > 0$ for all s.

(c) (5%) Show that Eq. (3) implies $V^{\pi_{k+1}}(\rho) \geq V^{\pi_k}(\rho)$ for any initial distribution ρ.

Combining (b) and (c), we formally showed that off-policy actor-critic is still performing reasonable update, justifying that we can eliminate the state distribution ratio in Page 14 here for the tabular special case.

2 Implementing DDPG / TD3

In this problem, we will implement actor-critic algorithms that work for continuous action sets. Specifically, we focus on deep deterministic policy gradient (DDPG) and twin-delayed DDPG (TD3). Download the starter code here.

2.1 Environment

We focus on the Hopper environment, which is part of the Mujoco environment. The hopper is a two-dimensional one-legged figure that consist of four main body parts – the torso at the top, the thigh in the middle, the leg in the bottom, and a single foot on which the entire body rests (Figure 1). The goal is to make hops that move in the forward (right) direction by applying torques on the three hinges connecting the four body parts. More information about the environment can be found in `https://gymnasium.farama.org/environments/mujoco/hopper/`. You can search for some videos on the internet to see what the expected movement of Hopper looks like.

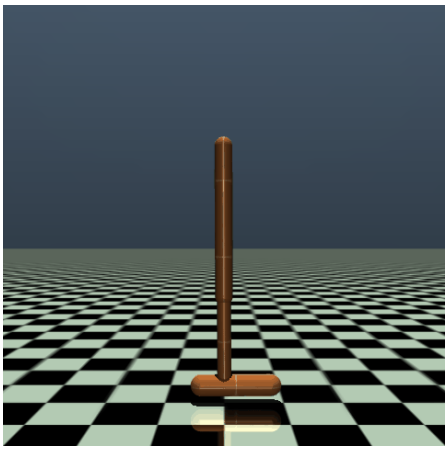

Figure 1: Hopper

Observation / state space: The observation / state space includes 11 features that describe the height of the hopper, angles of the joints, velocities, and angular velocities. The range of each feature is $(-\infty, \infty)$.

Action space: The action space is 3 dimensional. Each dimension represents the amount of torque applied to a joint. The range of each dimension is $[-1, 1]$.

Rewards: The reward is a sum of three parts. The first part is a constant for each step before termination. The second part measures how fast the hopper is moving in the positive x direction. The third part is a cost for taking large actions, i.e., the larger the magnitude of the action is, the larger the cost.

Episode termination / truncation: An episode terminates if the observation / state is outside a pre-defined range (usually means the hopper is fallen). When the number of steps reaches 1000, the episode will be truncated.

The Hopper environment requires you to install the Mujoco package, which might take some efforts. See the installation guide here. If issues persist in the installation, you can try to work on alternative environments that also have continuous actions (for example, the continuous version of Lunar Lander).

2.2 DDPG

DDPG is an off-policy actor-critic algorithm for continuous action space. It can also be viewed as an adaptation of Q-learning for continuous actions. The pseudocode of DDPG is given in Algorithm 1. Also see actor-critic slides Page 25 or OpenAI spinning up.

Algorithm 1 DDPG

Hyper-Parameters: $\sigma, \eta, \lambda, \tau, N, M$.
Initialize a replay buffer D with capacity N.
Initialize two actor networks (mappings from the state space to the action space) with weights θ and θ_{tar}.
Initialize two critic networks (mappings from the state-action space to a scalar value) with weights ϕ and ϕ_{tar}.
Initialize the environment with state s_1.
for $t = 1, \ldots$ **do**
Select action

$$a_t = \Pi_{\mathcal{A}} \left(\mu(s_t; \theta_{\text{tar}}) + \mathcal{N}(0, \sigma^2 I) \right)$$

where $\Pi_{\mathcal{A}}$ is the projection operator to the action space (in Hopper, simply clip each coordinate to $[-1, 1]$). Execute action a_t, observe reward r_t, and observe the next state s_{t+1} if the episode hasn't terminate.
Let s'_t be the terminal state if the episode terminates after t; otherwise, let $s'_t = s_{t+1}$.
Store transition (s_t, a_t, r_t, s'_t) in replay buffer D.
Sample a random minibatch of transitions $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{M}$ with size M from D.
For each j, define

$$a'_j = \mu(s'_j; \theta_{\text{tar}}).$$

Set

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is the terminal state} \\ r_j + \gamma Q(s'_j, a'_j; \phi_{\text{tar}}) & \text{otherwise} \end{cases} \tag{4}$$

Critic update: Perform a gradient descent step on MSE loss:

$$\phi \leftarrow \phi - \eta \nabla_{\phi} \frac{1}{M} \sum_{j=1}^{M} \left(Q(s_j, a_j; \phi) - y_j \right)^2.$$

Actor update: Perform a gradient ascent step:

$$\theta \leftarrow \theta + \lambda \nabla_{\theta} \frac{1}{M} \sum_{j=1}^{M} Q(s_j, \mu(s_j; \theta); \phi).$$

Target network update:

$$\phi_{\text{tar}} \leftarrow \tau \phi + (1 - \tau) \phi_{\text{tar}},$$
$$\theta_{\text{tar}} \leftarrow \tau \theta + (1 - \tau) \theta_{\text{tar}}.$$

If the episode is terminated, let s_{t+1} be the initial state of the next episode.

Tasks (40%) Implement DDPG (Algorithm 1) to play Hopper. Plot the evolution of the episodic return over time (x-axis: number of episodes, y-axis: episodic return).

2.3 TD3

TD3 is a direct successor of DDPG, which mainly include three additional elements. See actor-critic slides Page 26-29 or OpenAI spinning up. These three elements are:

- **Double-Q** Use two pairs of critic networks (so there are four critic networks in total: $\phi_1, \phi_{\text{tar}1}, \phi_2, \phi_{\text{tar}2}$), and always use the minimum of the target networks $\min_{\ell=1,2} Q(s, a; \phi_{\text{tar}\ell})$ as the regression target.
- **Delayed actor Update** Update actor less frequently than critic.
- **Target policy smoothing** Instead of using $Q(s', \mu(s'; \theta_{\text{tar}}); \phi_{\text{tar}})$ as the regression target, use $Q(s', a'; \phi_{\text{tar}})$ where a' is $\mu(s'; \theta_{\text{tar}})$ plus noise.

The psuedocode of TD3 is given in Algorithm 2, with the differences with DDPG highlighted in this color.

Algorithm 2 TD3

Hyper-Parameters: $\sigma, \eta, \lambda, \tau, N, M, c, \sigma', n$.
Initialize a replay buffer D with capacity N.
Initialize two actor networks (mappings from the state space to the action space) with weights θ and θ_{tar}.
Initialize four critic networks (mappings from the state-action space to a scalar value) with weights ϕ_1 and $\phi_{\text{tar}1}$.
Initialize the environment with state s_1.
for $t = 1, \ldots$ **do**
Select action

$$a_t = \Pi_{\mathcal{A}} \left(\mu_{\theta_{\text{tar}}}(s_t) + \mathcal{N}(0, \sigma^2 I) \right)$$

where $\Pi_{\mathcal{A}}$ is the projection operator to the action space (in Hopper, simply clip each coordinate to $[-1, 1]$). Execute action a_t, observe reward r_t, and observe the next state s_{t+1} if the episode hasn't terminate.
Let s'_t be the terminal state if the episode terminates after t; otherwise, let $s'_t = s_{t+1}$.
Store transition (s_t, a_t, r_t, s'_t) in replay buffer D.
Sample a random minibatch of transitions $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{M}$ with size M from D.
For each j, draw

$$a'_j = \Pi_{\mathcal{A}} \left(\mu(s'_j; \theta_{\text{tar}}) + \epsilon_j \right)$$

where $\epsilon_j \sim \Pi_{[-c,c]}(\mathcal{N}(0, \sigma'^2 I))$ is a truncated Gaussian noise.
Set

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is the terminal state} \\ r_j + \gamma \min_{\ell \in \{1,2\}} Q(s'_j, a'_j; \phi_{\text{tar}\ell}) & \text{otherwise} \end{cases} \tag{5}$$

Critic update: Perform a gradient descent step on MSE loss:

$$\forall \ell \in \{1, 2\} \qquad \phi_\ell \leftarrow \phi_\ell - \eta \nabla_{\phi_\ell} \frac{1}{M} \sum_{j=1}^{M} \left(Q(s_j, a_j; \phi_\ell) - y_j \right)^2 .$$

if $t \ mod \ n = 0$ **then**
Actor update: Perform a gradient ascent step:

$$\theta \leftarrow \theta + \lambda \nabla_\theta \frac{1}{M} \sum_{j=1}^{M} Q(s_j, \mu(s_j; \theta); \phi_1). \tag{6}$$

Target network update:

$$\forall \ell \in \{1, 2\} \qquad \phi_{\text{tar}\ell} \leftarrow \tau \phi_\ell + (1 - \tau) \phi_{\text{tar}\ell},$$
$$\theta_{\text{tar}} \leftarrow \tau \theta + (1 - \tau) \theta_{\text{tar}}.$$

If the episode is terminated, let s_{t+1} be the initial state of the next episode.

Task (35%) Implement TD3 (Algorithm 2) to play Hopper. Plot the episodic return over time (x-axis: number of episodes, y-axis: episodic return). Feel free to not entirely follow the original implementation of TD3, but just incorporate similar ideas. For example, instead of only updating the actor every n steps, you might simply create a separation between the learning rates of actor and critic, which could produce a similar effect.

Also, it will be interesting to render the environment and see how your Hopper is performing (we do not have this part of starter code right now, but will try to include it afterwards).

You will get full score if the final episodic return of your Hopper can reach $\gtrsim 1500$ or higher. Please refer to [1] to see the performance of the original implementation of TD3.

3 Survey (20%)

Thank you for staying in the class till the end. Please let us know any suggestions for the course so we can improve it in the future.

References

[1] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.