535514 Spring 2025: Reinforcement Learning **(Due: 2025/04/23, 21:00)**

Homework 2: DDPG, TRPO, and PPO

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 into one .zip file. Please submit your deliverables via E3.

Problem 1 (Surrogate Function in TRPO) (10 points)

In this problem, we will prove some key properties of the surrogate function used in TRPO. Recall from the slides of Lecture 8: Assume that both the state space and the action space are finite. The surrogate function $L_{\pi_{\theta_1}}(\pi_\theta)$ is defined as

$$L_{\pi_{\theta_1}}(\pi_\theta) := \eta(\pi_{\theta_1}) + \sum_{s\in\mathcal{S}} d_\mu^{\pi_{\theta_1}}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) A^{\pi_{\theta_1}}(s,a). \tag{1}$$

Show that $L_{\pi_{\theta_1}}(\pi_\theta)$ satisfies the two properties: (i) $L_{\pi_{\theta_1}}(\pi_{\theta_1}) = \eta(\pi_{\theta_1})$ and (ii) $\nabla_\theta L_{\pi_{\theta_1}}(\pi_\theta)|_{\theta=\theta_1} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_1}$.

Problem 2 (Solving TRPO Under Approximation Using Duality) (15+10=25 points)

Recall from the slides of Lecture 8: We would like to solve the following optimization problem (denoted by (OPT)):

$$\text{Minimize}_{\theta\in\mathbb{R}^d} \quad -(\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})^\top(\theta-\theta_k) \tag{2}$$

$$\text{subject to} \quad \frac{1}{2}(\theta-\theta_k)^\top H_{\theta_k}(\theta-\theta_k) - \delta \le 0. \tag{3}$$

We use θ^* to denote an optimizer of the above *primal* optimization problem (2)-(3). Note that in the above we write "Minimize" instead of "Maximize" simply to follow the conventions of the literature of optimization theory. Here we focus on the case where H is a *positive definite* matrix to avoid the technicalities (while H is only *non-negative definite* in general).

Based on the optimization theory, (OPT) is a convex optimization problem as both the objective and the constraints are convex functions. In this case, one standard way is to convert the constrained (OPT) into an unconstrained *dual* problem by defining the *Lagrangian* $\mathcal{L}(\theta,\lambda)$ and the *dual function* $D(\lambda)$ as:

$$\mathcal{L}(\theta,\lambda) := -(\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})^\top(\theta-\theta_k) + \lambda\Big(\frac{1}{2}(\theta-\theta_k)^\top H_{\theta_k}(\theta-\theta_k) - \delta\Big) \tag{4}$$

$$D(\lambda) := \min_{\theta\in\mathbb{R}^d} \mathcal{L}(\theta,\lambda), \tag{5}$$

where λ is called the *Lagrange multiplier*. Moreover, we call the following the *dual problem* of (OPT):

$$\max_{\lambda\ge 0} D(\lambda). \tag{6}$$

For ease of notation, we define $\lambda^* := \arg\max_{\lambda\ge 0} D(\lambda)$ as the optimizer of the dual problem. By the standard theory of convex optimization, if there exists one strictly feasible point in (OPT), then the optimal value of the dual problem is equal to the original problem (usually called "strong duality"). Moreover, if strong duality holds and a dual optimal solution λ^* exists, then any optimizer of the primal problem is also a minimizer of $\mathcal{L}(\theta,\lambda^*)$, i.e., $\theta^* = \arg\min_\theta \mathcal{L}(\theta,\lambda^*)$. For more details on duality, please refer to Chapter 5 of https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

(a) In this problem, please show that the dual function $D(\lambda)$ of (OPT) can be written as:

$$D(\lambda) = \frac{-1}{2\lambda}\big((\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})^\top H_{\theta_k}^{-1}(\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})\big) - \lambda\delta. \tag{7}$$

Accordingly, please find out λ^* based on (7).

(b) By the λ^* found in (a) and the property $\theta^* = \arg\min_\theta \mathcal{L}(\theta, \lambda^*)$, show that $\theta^* = \theta_k + \alpha H_{\theta_k}^{-1}\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k}$. What is the step size α?

Problem 3 (PPO With a Clipped Objective) (10 points)

Recall from Lecture 8 that the PPO-Clip updates the policy iteratively by maximizing the following objective function:

$$L^{\text{clip}}(\theta;\theta_k) := \mathbb{E}_{s\sim d_\mu^{\pi_{\theta_k}}, a\sim\pi_{\theta_k}(\cdot|s)}\big[L_{s,a}^{\text{clip}}\big], \tag{8}$$

$$\text{where}\quad L_{s,a}^{\text{clip}}(\theta;\theta_k) := \min\left\{\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\pi_{\theta_k}}(s,a), \text{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\pi_{\theta_k}}(s,a)\right\}. \tag{9}$$

As mentioned in class, another possibility is to consider the following variant:

$$\widetilde{L}_{s,a}^{\text{clip}}(\theta;\theta_k) := \text{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}, 1-\varepsilon, 1+\varepsilon\right)\cdot A^{\theta_k}(s,a). \tag{10}$$

Could you explain the behavioral differences between the two objective functions $L_{s,a}^{\text{clip}}(\theta;\theta_k)$ and $\widetilde{L}_{s,a}^{\text{clip}}(\theta;\theta_k)$? (Hint: For a clear comparison, it could be better to make a table and illustrative plots similar to Figure 1 below)

	$p_t(\theta) > 0$	A_t	**Return Value of** min	**Objective is Clipped**	**Sign of Objective**	**Gradient**
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
3	$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
4	$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
5	$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
6	$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

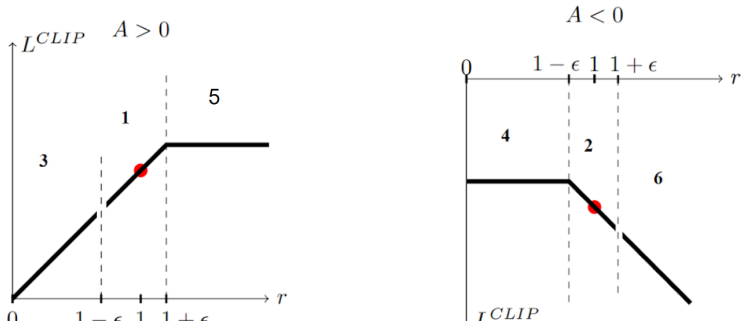

Figure 1: Behavior of the original PPO-clip objective.

Problem 4 (Deep Deterministic Policy Gradient for Continuous Control) (25+20+15=60 points)

In this problem, we will implement the deep determinisitc policy gradient (DDPG) algorithm with the help of neural function approximators, as discussed in Lecture 7. You may write your code in either PyTorch or TensorFlow (though the sample code presumes PyTorch framework). Moreover, you are recommended to use Weight & Bias (preferred) or Tensorboard to keep track of the loss terms and other related quantities of your implementation. If you are a beginner in learning the deep learning framework, please refer to the following

tutorials:

- PyTorch: https://pytorch.org/tutorials/
- Tensorflow: https://www.tensorflow.org/tutorials

For the deliverables, please submit the following:

- Technical report: Please summarize all your experimental results in 1 single report (and please be brief)
- All your source code
- Your well-trained models (both the actor and critic networks) saved in either .pth files or .ckpt files

(a) We start by solving the simple "Pendulum-v0" problem (https://gym.openai.com/envs/Pendulum-v0/) using the DDPG algorithm. Read through **ddpg.py** and then implement the member functions of the classes **Actor, Critic, and DDPG** as well as the function **train**. Please briefly summarize your results (including the snapshots of the Tensorboard or 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 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)

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

Recall from HW1 that you have already installed the MuJoCo package and will be able to directly play with the Halfcheetah locomotion task. Train your Halfcheetah for 500k environment steps and reach a score of at least 5000 within 500k 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, it is typically expected that Halfcheetah can achieve a score of 6000-10000 within 500k training steps)

(c) Based on your code for (b), further enhance your DDPG algorithm with the technique of "Clipped Double Q" (CDQ) for the critic update as in TD3, i.e., the loss of each critic network Q_{w_1}, Q_{w_2} with the clipped double Q shall be

$$L(w_i;\mathcal{B}) := \frac{1}{|\mathcal{B}|}\sum_{(s,a,r,s')\in\mathcal{B}}\Big(r+\gamma\min_{i'=1,2}Q_{w_i'}(s',\pi_\theta(s')+\varepsilon)-Q_{w_i}(s,a)\Big)^2,$$
$$\text{where } \varepsilon\sim\text{clip}\big(\mathcal{N}(0,\sigma^2),-\varepsilon_{\max},\varepsilon_{\max}\big)$$

to solve the "HalfCheetah" task in MuJoCo. Save your code in another file named **ddpg_cdq_cheetah.py**. Please add comments to your code whenever needed for better readability.

Again, train your Halfcheetah for 500k environment steps. What observations can you make from the added CDQ technique, compared to the vanilla DDPG in subproblem (b)? Please briefly summarize your results in the report and document all the hyperparameters of your experiments.