535514 Spring 2025: Reinforcement Learning **(Due: 2025/03/26, Wednesday, 21:00)**

Homework 1: Fundamentals of MDPs and Policy Gradient

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 (Q-Value Iteration) (12+12=24 points)

(a) Recall that in Lecture 2, we define $V^*(s) := \max_{\pi\in\Pi} V^\pi(s)$ and $Q^*(s,a) := \max_{\pi\in\Pi} Q^\pi(s,a)$. Suppose $\gamma \in (0,1)$. Prove the following Bellman optimality equations:

$$V^*(s) = \max_{a\in\mathcal{A}} Q^*(s,a) \tag{1}$$

$$Q^*(s,a) = R_{s,a} + \gamma \sum_{s'\in\mathcal{S}} P^a_{ss'} V^*(s'). \tag{2}$$

Please carefully justify every step of your proof. (Hint: For (1), you may first prove that $V^*(s) \le \max_a Q^*(s,a)$ and then show $V^*(s) < \max_a Q^*(s,a)$ cannot happen by contradiction. On the other hand, (2) can be shown by using the similar argument or by leveraging the fact that $Q^\pi(s,a) = R_{s,a} + \gamma \sum_{s'} P^a_{ss'} V^\pi(s')$)

(b) Based on (a), we thereby have the recursive Bellman optimality equation for the optimal action-value function Q_* as:

$$Q^*(s,a) = R_{s,a} + \gamma \sum_{s'} P^a_{ss'} \big(\max_{a'} Q^*(s',a')\big) \tag{3}$$

Similar to the standard Value Iteration, we can study the *Q-Value Iteration* by defining the Bellman optimality operator $T^* : \mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \to \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ for the action-value function: For every state-action pair (s,a), define

$$[T^*(Q)](s,a) := R_{s,a} + \gamma \sum_{s'} P^a_{ss'} \max_{a'} Q(s',a') \tag{4}$$

- Show that the operator T^* is a γ-contraction operator in terms of ℓ_∞-norm. Please carefully justify every step of your proof. (Hint: For any two action-value functions Q, Q', we have $\|T^*(Q) - T^*(Q')\|_\infty = \max_{(s,a)} \big|[T^*(Q)](s,a) - [T^*(Q')](s,a)\big|$)
- Does the operator T^* still have the γ-contraction property under ℓ_1-norm?

Problem 2 (A Super Useful Property Used in Policy Gradient) (10 points)

Please prove the following useful property discussed in Lecture 4: For any function $f : \mathcal{S}\times\mathcal{A} \to \mathbb{R}$,

$$\mathbb{E}_{\tau\sim P^{\pi_\theta}_\mu}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t,a_t)\right] = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi_\theta}_\mu}\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\big[f(s,a)\big] \tag{5}$$

(Hint: It might be slightly easier to go from the RHS to LHS. Specifically, you may first expand the RHS of (5) into a sum of $f(s,a)$ over s and a and then apply the definition of $d^{\pi_\theta}_\mu$, which involves a sum of probability over t. Next, try to reorganize the triple summation into the form of the LHS of (5))

Problem 3 (Baseline for Variance Reduction) (7+7+7=21 points)

Consider an example similar to that in the slides of Lecture 4 for explaining the baseline. Suppose that there are only 1 non-terminal starting state (denoted by s) and 3 actions (denoted by a, b, c) in the MDP of interest. After applying any of the actions in the starting state s, the MDP would evolve from s to the terminal state, with probability 1. Moreover, consider the following setting:

- The rewards are deterministic and the reward function is defined as $r(s,a) = 100$, $r(s,b) = 98$, and $r(s,c) = 95$. Moreover, there is no terminal reward.
- We consider a softmax policy with parameters $\theta_a, \theta_b, \theta_c$ such that $\pi_\theta(\cdot|s) = \exp(\theta_\cdot)/(\exp(\theta_a) + \exp(\theta_b) + \exp(\theta_c))$. Moreover, the parameters are currently $\theta_a = 0$, $\theta_b = \ln 5$, $\theta_c = \ln 4$.
- We would like to combine PG with SGD. At each policy update, we would construct an unbiased estimate $\widehat{\nabla} V$ of the true policy gradient $\nabla_\theta V^{\pi_\theta}$ by sampling one trajectory (Note: $\widehat{\nabla} V$ is a random vector. In this example, each trajectory has only one time step, and $s_0 = s$, a_0 is either a, b, or c, and s_1 is the terminal state).

(a) What are the mean vector of $\widehat{\nabla} V$ (denoted by $\mathbb{E}[\widehat{\nabla} V]$) and the covariance matrix of $\widehat{\nabla} V$ (i.e., $\mathbb{E}[(\widehat{\nabla} V - \mathbb{E}[\widehat{\nabla} V])(\widehat{\nabla} V - \mathbb{E}[\widehat{\nabla} V])^\intercal]$)?

(b) Suppose we leverage the value function $V^{\pi_\theta}(s)$ as the baseline and denote by $\tilde{\nabla} V$ the corresponding estimated policy gradient. Then, what are the mean vector and the covariance matrix of $\tilde{\nabla} V$? (Note: $\tilde{\nabla} V$ is also a random vector)

(c) Let $B(s)$ denote a baseline function and ∇V_B be the corresponding estimated policy gradient (∇V_B is again a random vector). Suppose we say that a baseline function $B(s)$ is *optimal* if it attains the minimum trace of the corresponding covariance matrix of ∇V_B among all possible state-dependent baselines. Please try to find one such optimal $B(s)$.

Problem 4 (Policy Gradient Algorithms With Function Approximation) (15+15+15=45 points)

In this problem, we will implement three policy gradient algorithms with the help of neural function approximators: (1) Vanilla REINFORCE, (2) REINFORCE with value function as the baseline, and (3) REINFORCE with Generalized Advantage Estimation (GAE). For the pseudo code of the algorithms, please see the slides of Lecture 4. You may write your code in either PyTorch or TensorFlow (though the sample code presumes PyTorch framework). Moreover, you are recommended to use either Tensorboard or Weight and Biases 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/
- Tensorboard: https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html
- Tensorflow: https://www.tensorflow.org/tutorials
- Tensorboard: https://www.tensorflow.org/tensorboard/get_started
- Weight and Biases: https://docs.wandb.ai/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 (REINFORCE with a baseline, without a baseline, and with GAE) saved in either .pth files or .ckpt files

(a) We start by solving the simple "CartPole-v0" problem (https://gym.openai.com/envs/CartPole-v0/) using the vanilla REINFORCE algorithm. Read through **reinforce.py** and then implement the member functions of the class **Policy** and the function **train**. Please briefly summarize your results in the report (including the snapshot of the Tensorboard record) and document all the hyperparameters (e.g. learning rates and NN architecture) of your experiments.

(b) Based on the code for (a), implement the REINFORCE algorithm with a baseline and solve the "LunarLander-v2" problem, which has slightly higher state and action space dimensionality (https://gym.openai.com/envs/LunarLander-v2/). Note that you are allowed to **design a baseline function on your own** (e.g., the baseline can either be a handcrafted state-dependent function, the value function, or some function learned directly from the trajectories). Please clearly explain your choice of the baseline function in the report. Save your code in another file named **reinforce_baseline .py**. Please add comments to your code whenever needed for better readability. Again, please briefly summarize your results in the report (including the snapshot of the Tensorboard record) and document all the hyperparameters of your experiments. (Note: As LunarLander is a slightly more challenging environment than CartPole, it might require some efforts to tune the hyperparameters, e.g., learning rates or learning rate scheduler. You could either do grid search or even use some more advanced techniques such as the evolutionary methods. Typically, the LunarLander problem is considered solved if the testing episodic return is above 200)

(c) Based on the code for (a), implement the REINFORCE algorithm with the Generalized Advantage Estimation (GAE) as your value prediction. Please run the code on "LunarLander-v2". Note that there is a hyperparameter λ in the GAE algorithm, **please try three different values of** λ and summarize your results of these three choices clearly in the report. For more details about GAE, please refer to the slides of Lectures 9-10 and the original paper (https://arxiv.org/abs/1506.02438). Please briefly explain your implementation of GAE in your report. Also, please save your code in another file named **reinforce_gae .py**. Last but not least, please add comments to your code whenever needed for better readability.

Remark: Regarding the snapshot of the Tensorboard and Weight and Biases records, please feel free to record any value that you aim to know. Here is an example:

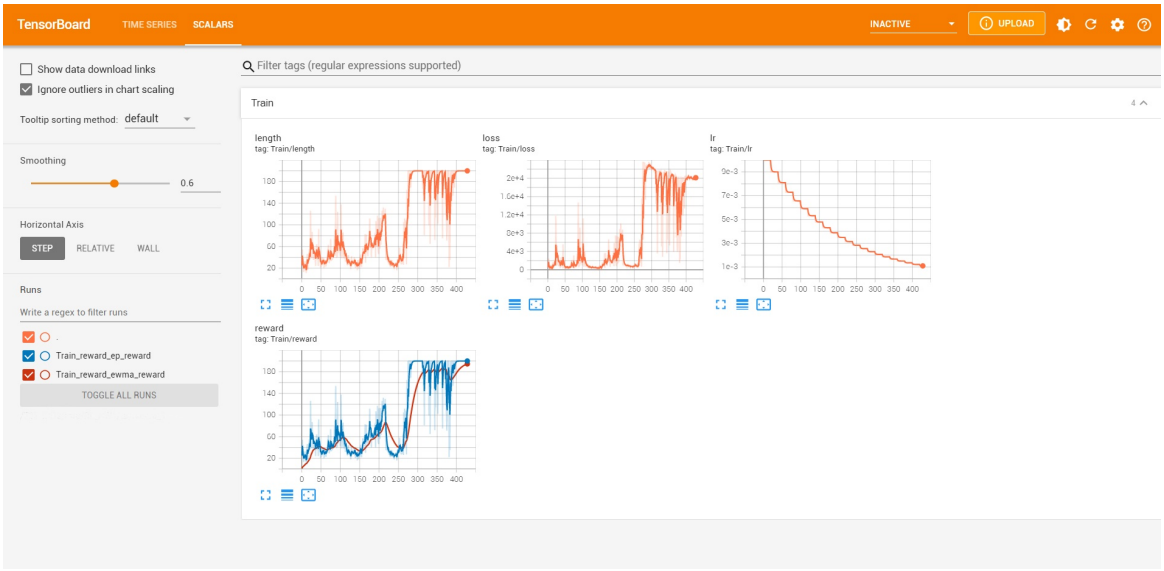

Figure 1: An example of the Tensorboard record.

Problem 5 (Installing and Getting Familiar With MuJoCo and D4RL) (10 points)

In this problem, you will be asked to install and investigate the D4RL Dataset, which is currently the standard dataset for Offline RL. We will continue to use the MuJoC tasks and D4RL dataset for the subsequent homeworks. Therefore, it would be very helpful to now set up the environment that you could easily reuse subsequently. To accomplish this task, you shall take the following steps:

- Please first take a careful look at the GitHub repo of D4RL https://github.com/Farama-Foundation/D4RL and the paper https://arxiv.org/abs/2004.07219. Then, install **d4rl** by following the installation

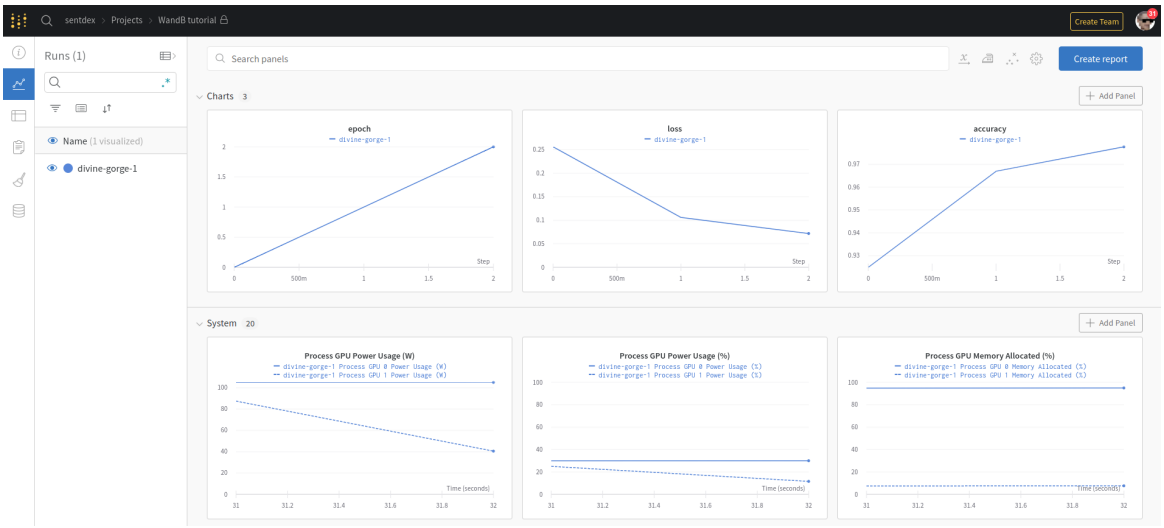

Figure 2: An example of the Weight and Biases record.

guide on the GitHub repo (you need to install several other packages, such as Gymnasium and MuJoCo).

- Read and run the provided **d4rl_sanity_check.py** and finish the following tasks:
  - (1) Generate an offline dataset by directly running **d4rl_sanity_check.py**.
  - (2) Describe the format of the dataset that you just obtained.
  - (3) Modify the **d4rl_sanity_check.py** and generate another offline dataset of one of the MuJoCo tasks (e.g., Hopper, Walker, or Halfcheetah). Similarly, describe the format of the MuJoCo dataset that you just obtained.