Homework 3

CMU 10-703: Deep Reinforcement Learning (Fall 2023)
OUT: Oct. 23, 2023
DUE: Nov. 6, 2023 by 11:59pm ET

Instructions: START HERE

- **Collaboration policy:** You may work in groups of up to three people for this assignment. It is also OK to get clarification (but not solutions) from books or online resources after you have thought about the problems on your own. You are expected to comply with the University Policy on Academic Integrity and Plagiarism[1].

- **Late Submission Policy:** You are allowed a total of 8 grace days for your homeworks. However, no more than 3 grace days may be applied to a single assignment. Any assignment submitted after 3 days will not receive any credit. Grace days do not need to be requested or mentioned in emails; we will automatically apply them to students who submit late. We will not give any further extensions so make sure you only use them when you are absolutely sure you need them. See the Assignments and Grading Policy here for more information about grace days and late submissions: `https://cmudeeprl.github.io/703website_f23/logistics/`

- **Submitting your work:**

  - **Gradescope:** Please write your answers and copy your plots into the provided LaTeX template, and upload a PDF to the GradeScope assignment titled "Homework 1." Additionally, export the code from your Colab notebook ([File → Export .py]) and upload it the GradeScope assignment titled "Homework 1: Code." Each team should only upload one copy of each part. Regrade requests can be made within one week of the assignment being graded.
  - **Autolab:** Autolab is not used for this assignment.

[1] `https://www.cmu.edu/policies/`

Problem 0: Collaborators

Please list your name and Andrew ID, as well as those of your collaborators.

Problem 1: CMA-ES (45 pts)

In this problem you will implement CMA-ES, a black-box optimization algorithm. To help you get started, we have provided some template code in this Notebook:

`https://colab.research.google.com/drive/1TsPRNQZSuK1uoKVfq1RB4XLNhxx9XcAF`

You are welcome to implement the assignment from scratch using whatever programming language you prefer.

1. [**20 pts**] Implement CMA-ES using the following update equations:

$$\mu_{t+1} \leftarrow \frac{1}{\text{elite size}} \sum_{i=1}^{\text{elite size}} \theta_t^{(i)}, \qquad \Sigma_t \leftarrow \text{Cov}\left(\theta_t^{(1)}, \cdots, \theta_t^{(\text{elite size})}\right) + \epsilon I,$$

where $\theta_t^{(i)}$ denotes the i-th best parameters from the previous iteration and $\epsilon \in \mathbb{R}$ is a small constant. We recommend the following hyperparameters:

- Initial μ: $\vec{0}$
- Initial covariance: $100I$
- Population size: 100
- Fraction of population to keep at each iteration: 10%
- Noise ϵ added to covariance at each step: $0.25I$

Run your implementation of CMA-ES to *maximize* on the following simple objective function:

$$f(x) = -\|x - x^*\|_2^2 \quad \text{where} \quad x^* = [65, 49].$$

This function is optimized when $x = x^*$. Run your implementation of CMA-ES on this function, confirming that you get the correct solution. Make a plot showing the values of μ from each iteration. Use $\mu[0]$ for the X axis and $\mu[1]$ for the Y axis. Please label the initial and final values of μ, as well as the global optimum. Remember to label your axes.

2. [**10 pts**] In the second part of this problem, you will use CMA-ES to solve a RL problem. You will use the `Cartpole-v0` environment from OpenAI gym. Our first task will be to make an objective function that takes as input the parameters of a policy and outputs the reward of that policy. We will parametrize the policy as

$$\pi(a = \text{LEFT} \mid s) = s \cdot w + b,$$

where $w \in \mathbb{R}^4$ and $b \in \mathbb{R}$ are parameters that you will optimize with CMA-ES. Define a function that takes as input a single vector $x = (w, b)$ and the environment and returns the total (undiscounted) reward from one episode. To check your implementation, evaluate the following policies 1000 times and report the average total reward (we've provided the answer for the first policy):

$x = (-1, -1, -1, -1, -1)$	$x = (1, 0, 1, 0, 1)$	$x = (0, 1, 2, 3, 4)$
15.6		

3. [**15 pts**] Run CMA-ES on the RL objective function. CMA-ES should be able to get an average population reward of at least 195 in 10 iterations. Include plot showing mean sample reward and best sample reward (Y axis) across iterations (X axis). Remember to label both lines and both axes.

Problem 2: BC and DAGGER (36 pt)

Behavior cloning and DAGGER are simple solutions to sequential decision making problems when provided with expert supervision. In this problem, you will implement them from an expert policy model. The template code is provided with the attached code package.

Background: Imitation Module

We have provided you with some function templates that you should implement. If you need to modify the function signatures, you may do so, but specify in your report what you changed and why.

Preliminaries (0 pt)

First, you will implement either a TensorFlow or Pytorch model in `model_tensorflow.py` or `model_pytorch.py`. You will use this model a number of times in the subsequent problems. To test that your model is implemented correctly, you will use it to solve a toy classification task.

Next, you will implement a function to collect data using a policy in an environment. This function will be used in subsequent problems. To test your BC or DAGGER implementation, you will run the `BCDAGGER.py`.

Behavior Cloning (16 pt)

Start by implementing the `train()` method of the `Imitation` class in `imitation.py`.

1. [**8 pt**] Run your behavior cloning implementation for 100 iterations by calling `plot_student_vs_expert()`. Every iteration use 100 expert episodes and run 1 seed. Plot the reward, training loss, and training accuracy throughout training, remembering to label the axes. In addition, plot the expert return with a horizontal line.

2. [**8 pt**] This question studies how the amount of expert data effects the performance. You will run the same experiment by calling `plot_compare_num_episodes()`, each time varying the number of expert episodes collected at each iteration. Use values of 1, 10,

50, and 100 in `keys`, and run 3 seeds for each value and then average over the seeds. As before, plot the reward, loss, and accuracy for each, remembering to label each line and the axes.

DAGGER (20 pt)

In the previous problem, you saw that when the cloned agent is in states far from normal expert demonstration states, it does a worse job of controlling the cart-pole than the expert. In this problem you will implement the DAGGER algorithm [2]. Implementing DAGGER is quite straightforward. First, implement the `generate_dagger_data()` method of the `Imitation` class. Second, set `mode = dagger`.

1. [**8 pt**] Run your DAGGER implementation for 100 iterations by calling `plot_student_vs_expert()`. Use 100 expert episodes and run 1 seed. Plot the reward, training loss, and training accuracy throughout training, remembering to label the axes. In addition, plot the expert return with a horizontal line.
2. [**8 pt**] This question studies how the amount of expert data effects performance. You will run the same experiment by calling `plot_compare_num_episodes()`, each time varying the number of expert episodes collected at each iteration. Use values of 1, 10, 50, and 100 in `keys`, and run 3 seeds for each value and then average over the seeds. As before, plot the reward, loss, and accuracy for each, remembering to label each line and the axes.
3. [**4 pt**] Does your DAGGER implementation outperform your BC implementation? Generate a hypothesis to explain this observation. What experiment could you run to test this hypothesis?

Problem 3: Goal-Conditioned BC (45 pt)

Goal-conditioned tasks aim to reach any desired state, i.e., *goal*. This is actually an instance of multi-task learning, where each task is to learn to reach a specified goal in the state space. In this problem, we focus on using imitation learning to solve goal-conditioned tasks from expert demonstrations. Interestingly, the expert is a classic search algorithm.

You will (1) build a goal-conditioned environment, (2) implement one shortest-path algorithm on this environment to obtain the expert data, (3) train a goal-conditioned policy from expert data using behavior cloning (GCBC), and (4) apply expert relabeling trick to improve the policy.

Build Goal-Conditioned Task (3 pt)

As the first step, you will help create the environment. Though environment dynamics are normally assumed to be unknown to RL/IL algorithms, sometimes we need to build toy environments for sanity check. We simplified the classic `FourRooms` environment for this problem. It has four fully-connected $l \times l$ rooms, where l is an odd number.

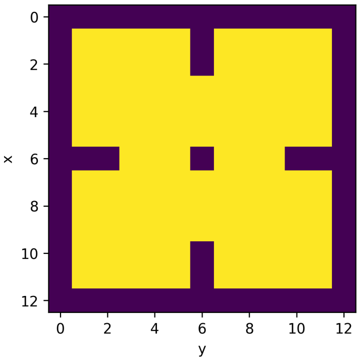

Figure 1: Our `FourRooms` Map, where $l = 5$.

Fig 1 is the map of our `FourRooms` environment, which is a binary matrix. Each pixel represents a grid (a 2D integer coordinate (x, y)), the bright pixel means a valid grid that can be passed, while the dark pixel means a grid that cannot be passed, i.e., wall.

Thus the state space $\mathcal{S}$ is composed of all the valid grids. The action space $\mathcal{A}$ is composed of moving right, up, left, and down, each of which can be expressed as a 2D *unit* vector. The transition is deterministic for $s \in \mathcal{S}, a \in \mathcal{A}$:

$$s' = f(s, a) = \begin{cases} s + a & \text{if } s + a \, \text{is valid} \\ s & \text{otherwise} \end{cases} \tag{1}$$

Formally, the goal-reaching task can be expressed by a tuple (s, g, T), where $s, g \in \mathcal{S}$ and $T \in \mathbb{N}$ is maximum horizon. The agent starts from state s, takes an action in each step, and tries to reach the goal g (the goal is fixed through the episode). The episode ends when it reaches the goal or time is over. The trajectory can be expressed as $(s_1, a_1, s_2, \ldots, s_{t-1}, a_{t-1}, s_t)$ of horizon $t \leq T$, where $s_1 = s$. If it reaches the goal, i.e., $s_t = g$, then we consider it has a successful trial, otherwise it fails.

[**3 pt**] Implement the deterministic dynamics f in Eq. 1 through the `step()` in `FourRooms` class. Plot one state trajectory of the uniformly random policy.

Run Search Algorithm as Expert (8 pt)

As you are aware, this maze-like problem can be easily solved by search algorithms for shortest-path problem.

[**8 pt**] Now please implement **any single-source shortest path algorithm** as expert on this environment given a goal-reaching task $(s, g, \cdot)$ and then collect expert data ($N = 1000$ trajectories) for GCBC. Plot a subset of expert state trajectories using provided code.

Train Goal-Conditioned Policy via BC (10 pt)

Now you have obtained the expert demonstrations $\mathcal{D} = \{s_1^j, a_1^j, \ldots, s_{T_j}^j\}_{j=1}^D$, where T_j is the horizon of the j-th trajectory, and the goal is the final state, i.e., $g^j = s_{T_j}^j$.

In this section, you will train a goal-conditioned policy $\pi_\theta(a|s, g)$ that is conditioned on both state and goal via BC:

$$\max_\theta \mathbb{E}_{(s_t^j, a_t^j, g^j)\sim\mathcal{D}} \left[\log \pi_\theta(a_t^j|s_t^j, g^j)\right] \tag{2}$$

[10 pt] Run your vanilla GCBC implementation for 150 iterations (in `vanilla` mode) with 5 seeds(Generate 5 different sets of trajectories, same setting for all the GCBC implementations below). Plot the success rate, training loss, and training accuracy throughout training across the seeds.

Expert Relabeling Trick (20 pt)

In goal-conditioned IL, we can relabel the expert goals based on the optimality assumption of expert trajectory. Since search algorithms naturally satisfy the geodesic property [1], in this section, you will implement the **expert relabeling trick**: relabel the goal g^j in (s_t^j, a_t^j, g^j) as any future state s_k^j, where $t < k \leq T_j$, in the expert demonstrations.

1. **[10 pt]** Run your expert-relabeling GCBC implementation for 150 iterations (in `relabel` mode) with 5 seeds. Plot the success rate, training loss, and training accuracy throughout training across the seeds (and label each axis).
   Does expert-relabeling GCBC perform better than vanilla GCBC? If yes, please write the possible reasons. If no, please write the possible issues.

2. **[10 pt]** Does expert relabeling trick also apply to **random policy data**?
   In this question, please apply the whole GCBC process to a random policy: generate $N = 1000$ trajectories from a uniformly random policy, and train vanilla GCBC with *original* goals (mostly not reached by the random policy though) and relabel GCBC with *relabeled* goals. Please Train 50 iterations for 5 seeds. Plot the success rate, training loss, and training accuracy throughout training across the seeds (and label each axis). Limit the Y-axis within [0,1] for the success rate plot.
   Compare the performance together with that of "real expert" data. Write 2-3 sentences to give the possible reasons.

Conceptual Questions (4 pt)

True or False with explanation for each question.

Assumption: Expert policy is optimal, and the dynamics can be any(Not limited to FourRooms).

1. **[2 pt]** Given the expert demonstrations, can GCBC *without* expert relabeling acquire optimal policy?

2. **[2 pt]** Given the expert demonstrations, can GCBC *with* expert relabeling acquire optimal policy?

Feedback

Feedback: You can help the course staff improve the course by providing feedback. What was the most confusing part of this homework, and what would have made it less confusing?

Time Spent: How many hours did you spend working on this assignment? Your answer will not affect your grade.

References

[1] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. In *Advances in Neural Information Processing Systems*, pages 15324–15335, 2019.

[2] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011.