Master IASD
Deep Reinforcement Learning
HW1: Imitation Learning

Triforce Neural:
Arij Boubaker, Linghao Zeng, Zhe Huang

1 Analysis

Question 1

First, we consider the given condition over the horizon T:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{p_{\pi^*}(s_t)}[\pi_\theta(a \neq \pi^*(s_t)|s_t)] \leq \epsilon$$

This expresses the expected rate at which the learned policy π_θ disagrees with the expert policy π^* is at most ϵ over the horizon T.

We aim to prove that for any time step t, the absolute difference in state distributions between π_θ and π^* is bounded by $2T\epsilon$:

$$|p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)| \leq 2T\epsilon$$

We apply the union bound, which states that for a set of events $E_1, E_2, \ldots, E_T$, the probability of at least one of these events occurring is less than or equal to the sum of the probabilities of each event:

$$\Pr\left[\bigcup_{t=1}^{T} E_t\right] \leq \sum_{t=1}^{T}\Pr[E_t]$$

We denote the total variation distance between the state distributions under π_θ and π^* at time t as $D_t = |p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)|$. The probability of disagreement at any state over the horizon T contributes to this variation distance.

Considering the absolute difference in state distributions and noting that the total variation distance at each time step contributes to the sum, we conclude that:

$$|p_{\pi_\theta}(s_t) - p_{\pi^*}(s_t)| \leq 2T\epsilon$$

This final inequality demonstrates that the deviation in state distributions between the learned policy π_θ and the expert policy π^* is bounded by $2T\epsilon$, under the assumption of bounded expected disagreement.

Question 2

Part (a)

We need to show that $J(\pi^*) - J(\pi_\theta) = O(T\epsilon)$ when $r(s_t) = 0$ for all $t < T$.

The expected return for a policy π is given by:

$$J(\pi) = \sum_{t=1}^{T} \mathbb{E}_{p_\pi(s_t)}[r(s_t)]$$

Since $r(s_t) = 0$ for all $t < T$, the expected return simplifies to:

$$J(\pi) = \mathbb{E}_{p_\pi(s_T)}[r(s_T)]$$

Thus, the difference in expected returns is:

$$J(\pi^*) - J(\pi_\theta) = \mathbb{E}_{p_{\pi^*}(s_T)}[r(s_T)] - \mathbb{E}_{p_{\pi_\theta}(s_T)}[r(s_T)]$$

Expanding this, we have:

$$\begin{aligned} J(\pi^*) - J(\pi_\theta) &= \sum_{s_T} p_{\pi^*}(s_T) r(s_T) - \sum_{s_T} p_{\pi_\theta}(s_T) r(s_T) \\ &= \sum_{s_T} (p_{\pi^*}(s_T) - p_{\pi_\theta}(s_T)) r(s_T) \end{aligned}$$

Using the bound $|p_{\pi_\theta}(s_T) - p_{\pi^*}(s_T)| \leq 2T\epsilon$ (as derived from previous results) and the fact that $|r(s_T)| \leq R_{\max}$, we get:

$$\begin{aligned} |J(\pi^*) - J(\pi_\theta)| &\leq \sum_{s_T} |p_{\pi^*}(s_T) - p_{\pi_\theta}(s_T)| R_{\max} \\ &\leq 2T\epsilon R_{\max} \end{aligned}$$

Part (b)

Now, we consider the case of an arbitrary reward function. The expected return difference is:

$$\begin{aligned} J(\pi^*) - J(\pi_\theta) &= \sum_{t=1}^{T} \mathbb{E}_{p_{\pi^*}(s_t)}[r(s_t)] - \sum_{t=1}^{T} \mathbb{E}_{p_{\pi_\theta}(s_t)}[r(s_t)] \\ &= \sum_{t=1}^{T} (\mathbb{E}_{p_{\pi^*}(s_t)}[r(s_t)] - \mathbb{E}_{p_{\pi_\theta}(s_t)}[r(s_t)]) \\ &\leq \sum_{t=1}^{T} R_{\max} \cdot |p_{\pi^*}(s_t) - p_{\pi_\theta}(s_t)| \\ &\leq \sum_{t=1}^{T} R_{\max} \cdot 2T\epsilon \\ &= R_{\max} \cdot 2T^2\epsilon \\ &= O(T^2\epsilon) \end{aligned}$$

This concludes the proof that $J(\pi^*) - J(\pi_\theta) = O(T^2\epsilon)$ for an arbitrary reward function.

2 Editing Code

In the implementation of behavioral cloning for MuJoCo tasks in OpenAI Gym, key components of the provided codebase were completed as per the instructions. The `MLP_policy.py` was enhanced to define the forward and update functions of the policy network, crucial for generating and refining actions based on observed data. In `utils.py`, the `sample_trajectory` function was developed to collect state-action sequences from policy execution. The central piece of the implementation, `run_training_loop` in `scripts/run_hw1.py`, orchestrated the overall behavioral cloning process. This involved initializing the environment and policy, leveraging expert data for initial training, and iteratively updating the policy through data collection, training, and evaluation.

3 Behavioral Cloning

Question 3.1

We conducted experiments on Behavioral Cloning across multiple environments to evaluate the performance of BC agents as shown in the following table:

Environment	Expert	Avg	Std	ep_len	Learning Rate	Train Steps	Trajectories
Ant-v4	4713	4740	21	1000	5e-3	1000	5
HalfCheetah-v4	4205	3998	94	1000	5e-3	1000	5
Hopper-v4	3772	982	102	288	5e-3	10000	17
Hopper-v4	3772	589	199	212	1e-3	10000	24
Walker2d-v4	5566	107	197	54	5e-3	10000	93
Walker2d-v4	5566	3363	2048	677	3e-3	10000	7

Table 1: Behavioral Cloning performance across different environments, evaluated with a fixed number of iterations (`n_iter`=1) due to BC's single-phase training nature. All models were trained using a Mean Squared Error (MSE) loss function and a neural network architecture of 2 hidden layers with 64 units each. The training batch size was set at 100 for all experiments. Evaluation was based on approximately `eval_batch_size` / `ep_len` rollouts with an evaluation batch size of 5000, allowing for the collection of multiple trajectories to calculate the mean and standard deviation of the policy's return.

Based on the data presented in the table, we can conduct a preliminary analysis of the BC agents' performance across various environments. The Ant-v4 environment showcases a BC agent that not only achieves but slightly surpasses the performance of the expert, with an average return of 4740 compared to the expert's 4713. This indicates a successful application of BC in this environment, where the agent manages to replicate the expert's behavior effectively.

In contrast, the Hopper-v4 and Walker2d-v4 environments present scenarios where BC agents fall short of reaching 30% of the expert's performance. For instance, in one of the Hopper-v4 experiments, the BC agent achieves an average return of 982, which is significantly lower than the expert's return of 3772, falling well below the 30% threshold. Similarly, in the Walker2d-v4 environment, the lowest average return observed is 107, a stark contrast to the expert's return of 5566, indicating a considerable gap in performance.

This analysis suggests that while BC can be effective in some contexts (e.g., Ant-v4), its performance can drastically vary in different environments and hyperparamters, particularly those that might present more complex or nuanced challenges for the agent to learn solely from expert demonstrations.

Question 3.2

In our experiment, we investigated the impact of neural network depth and learning rates on the performance of a Behavioral Cloning agent within a specified task. The chosen hyperparameters were manipulated to discern their influence on the agent's ability to imitate expert behavior accurately.

The graph displaying the BC agent's performance variation with respect to these hyperparameters reveals significant insights.

We selected the number of layers as a hyperparameter to understand how the model's capacity affects learning, hypothesizing that additional layers might capture more complex patterns. Conversely, we also anticipated that too many layers could lead to overfitting. The learning rate was chosen because it determines the step size in the optimization landscape, which is crucial for convergence and learning efficiency.

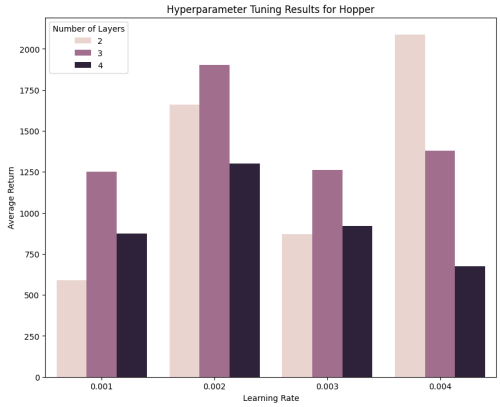

Figure 1: This graph illustrates the impact of network complexity and learning rates on the average return in the Hopper-v4 environment. The x-axis represents the learning rate applied during training, while the y-axis indicates the average return. Different colors represent the number of layers in the neural network architecture, highlighting the performance trends across the configurations.

Number of layers	Learning Rate	Average Return	Std Return
2	1e-5	589	199
2	2e-5	1661	298
2	3e-5	871	369
2	4e-5	2086	737
3	1e-5	1252	255
3	2e-5	1900	718
3	3e-5	1260	385
3	4e-5	1387	730
4	1e-5	874	5
4	2e-5	1302	265
4	3e-5	918	284
4	4e-5	676	340

Table 2: Performance metrics across different neural network depths and learning rates, showcasing the average and standard deviation of returns for the Hopper-v4 environment. This illustrates the impact of network complexity and learning rate adjustments on the reinforcement learning agent's ability to replicate expert behavior.

Note: We can also investigate the influence of the number of training steps per iteration on the performance of Behavioral Cloning in the Hopper-v4 environment. This parameter is pivotal because it directly affects the variety of experiences the agent encounters during training and its subsequent learning efficacy. Insufficient training steps may not provide ample diversity of expert data, potentially hindering the agent's ability to accurately emulate the expert's behavior. Conversely, an excessively high number of training steps per iteration could lead the agent to rote memorization of the expert's actions, potentially resulting in overfitting. Such overfitting is often characterized by a reduction in the average reward and its standard deviation during evaluation.

4 DAgger

Question 4.1

In our experiment with the DAgger algorithm on the Ant-v4 environment, we observed promising results in the following table:

Environment	Expert	Avg	Std	ep_len	Lr	Train steps	eval_batch_size
Ant	4597	4715	63	1000	5e-3	10 * 1000	5000

Table 3: Performance of the DAgger algorithm on the Ant-v4 environment. This table reflects the DAgger algorithm's capability to not only learn from but also improve upon the expert's policy in the Ant-v4 task. Achieving an average return of 4715, the DAgger agent outperformed the expert, whose return was 4597. The relatively low standard deviation of 63 suggests a consistent performance across different rollouts, further indicating the robustness of the learned policy.

Question 4.2

Environment	Expert	Avg return	Std return	Lr	eval_batch_size
Ant	4597	4715	63	5e-3	5000
HalfCheetah	4237	4050	79	5e-3	100000
Hopper	3714	3711	3	5e-3	100000
Hopper	3716	3715	3	2e-3	100000
Walker2d	5310	5272	439	5e-3	100000
Walker2d	5394	5346	317	1e-3	100000

Table 4: This table juxtaposes the average returns achieved by expert policies and the DAgger algorithm across different tasks within the reinforcement learning framework. The data reveals that while the DAgger algorithm demonstrates a strong alignment with expert performance in the Hopper environment, evidenced by a minimal standard deviation, there is considerable variability in the Walker2d environment, as reflected by a higher standard deviation. Each task's learning rate and evaluation batch size parameters are tailored, illustrating the necessity for specific hyperparameter optimization to enhance algorithmic performance.

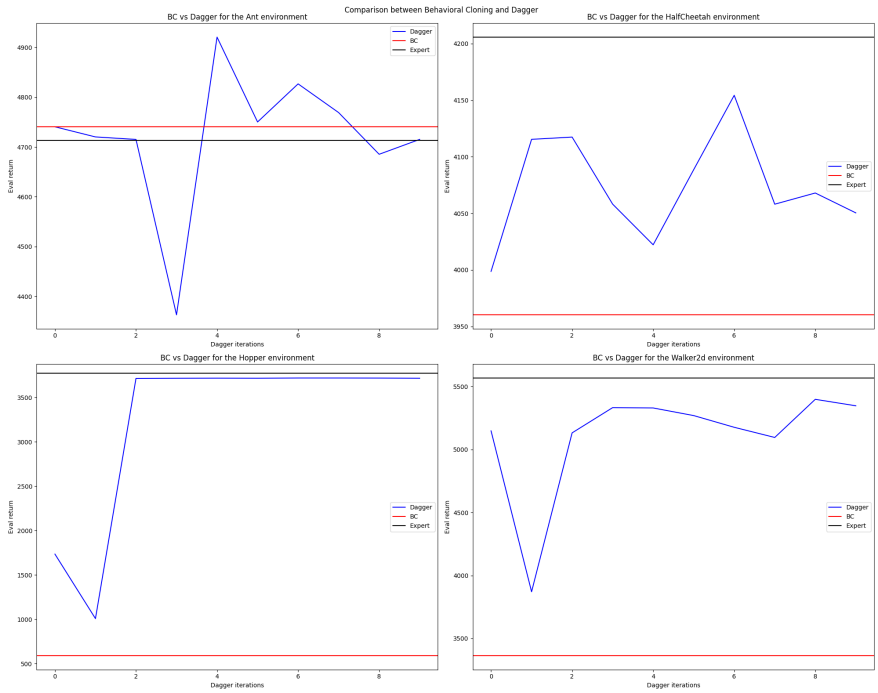

Figure 2: The set of line graphs illustrates the performance comparison between BC and the DAgger algorithm across four different environments: Ant, HalfCheetah, Hopper, and Walker2d. The blue lines represent the varying performance of DAgger across iterations, while the red horizontal lines indicate the consistent benchmark set by the expert's performance. Notably, DAgger's performance fluctuates in the Ant and HalfCheetah environments, suggesting a learning curve with occasional dips and peaks that do not consistently surpass the expert level. In the Hopper environment, DAgger shows a marked improvement after initial iterations, approaching the expert's performance. For the Walker2d environment, DAgger's performance exhibits high variability, with some iterations achieving superior returns compared to the expert. These observations underscore the dynamic learning process of DAgger, which iteratively refines its policy through interactions with the expert's policy.

5 Discussion

Question 5.1

We approached this assignment with a strategic time management plan, taking into account the varying complexities of each component. We dedicated four days to the intricate task of editing the code, ensuring precision and functionality. Following that, we invested about a week in developing the Behavioral Cloning, as it required extensive research and detailed analysis. The remaining time was allocated to redact the report, answering the theoretical questions and working on the DAgger part, balancing our efforts to maintain a high standard across all sections of the assignment.

Question 5.2

This assignment was both challenging and insightful part of the course, effectively blending theoretical knowledge with practical skills.
Experimenting with different hyperparameters to observe their impact on performance was particularly enlightening.
However, we faced challenges with environment setup and understanding code interdependencies.