CS 6955 HW1: Behavior Cloning in PyTorch Written Report

Kutay Eken
u1322888

January 2025

1 Part 1:

I noticed that the car is trying to explore and can't achieve the goal in the given time/steps. I think that trial and error method like RL would struggle with MountainCar problem since it uses reward as a signal for what is good and bad. The reason for that is because the agent receives -1 reward for each step taken but, the agent does not get any feedback on whether the action is good or bad in the long run. For example, the agent initially has to move away from the flag to gain speed and climb up the hill, but when the agent does that, it gets a -1 reward, making the agent think that it was a bad action at that time which is not the case. The only time agent receives a positive reward is when the agent reaches the flag which might not be common during the trial and error (exploration) process. Therefore, the reward signal does not really help the agent during exploration, which may lead to a longer time for the agent to discover and learn the correct actions.

2 Part 2:

From the different strategies that I have tried, I liked the "go right, go left, go all the way up the hill" strategy most. My best score among the all trials was -84.

3 Part 3:

My agent did not do a good job imitating at first but after I have changed the default number of iterations to 5000, it performed really well and never got stuck. The car was able to climb the hill successfully in all evaluations. Below are the results I got:

- **Average Policy Return: -129.0**
- **Min. Policy Return: -144.0**

- **Max. Policy Return: -113.0**

4 Part 4:

When 5 demonstrations were given, the agent performed better during evaluations. Throughout the 5 demonstrations, I kept the strategy same ("go right", "go left", "go right all the way up the hill") and the agent was able to follow the same strategy during evaluations. I think that the reason agent performed better is because it was able to observe more so there were more data to learn from. Below are the results I got:

- **Average Policy Return: -117.5**
- **Min. Policy Return: -121.0**
- **Max. Policy Return: -113.0**

5 Part 5:

I believe an agent's performance can be worse when exposed to a poor demonstration because the agent attempts to mimic what it observes. After providing one good and one bad demonstration, I noticed that the agent did not converge on a single consistent policy. Instead, it learned a policy that was a mix of both "good" and "bad" behaviors, resulting in inconsistency and failures during evaluations. Specifically, the agent mimicked the bad demonstration in the scenarios it failed and followed the good demonstration in the scenarios where it successfully reached the goal. This suggests that exposing the agent to bad demonstrations may confuse it, as it treats all demonstrations equally, causing it to learn bad actions from poor examples.

One potential approach to make behavior cloning (BC) more robust against bad demonstrations is to inform the agent about the quality of each demonstration. This could be achieved by assigning weights to the demonstrations, where good demonstrations are given higher weights and bad ones lower weights. By doing so, the agent can focus more on learning from good demonstrations while still considering the bad ones to some extent, allowing for a more balanced and effective learning process.

6 Part 6:

Yes, I was able to teach the agent to oscillate without ending the episode early. It oscillated between two hills and failed all evaluations. Below are the results I got:

- **Average Policy Return: -200.0**
- **Min. Policy Return: -200.0**
- **Max. Policy Return: -200.0**

7 Part 7:

There are some parts that needs to be added and there are some parts that needs to be changed in 'mountain_car_bc.py' to implement BCO. The first part that needs to be added is the part where agent self-explores the task itself and collects (s,a,s') data. The next thing that needs to be added is the training process of the "Inverse Dynamic Model". The last part that needs to be modified comes in when we collect the human demonstration data. The reason for that is because, this time, we need the state transitions to predict the actions unlike the 'mountain_car_bc.py'. The rest of the code should not change too much (collecting human demos, evaluating and training policy).

8 Part 8:

When 5 demonstrations were given to the agent, these were the results I got:

- **Average Policy Return: -114.0**
- **Min. Policy Return: -120.0**
- **Max. Policy Return: -110.0**

The results clearly show that this was the best performance I could achieve without me directly playing it. It outperformed the results from 'mountain_car_bc.py', even though that was not the case all the time but it might be because of my demonstrations as well. To achieve this, I increased the number of interactions to 35 to ensure the agent could explore most of the state space effectively. Additionally, I set the default number of iterations to 5000. Lastly, I designed the inverse dynamics network as a 3-layer neural network since I thought that this was a more complex problem compared to Part 3.