Assignment 1

Authors: Jainta Paul [u1471999]
Yeaseen Arafat [u1464680]

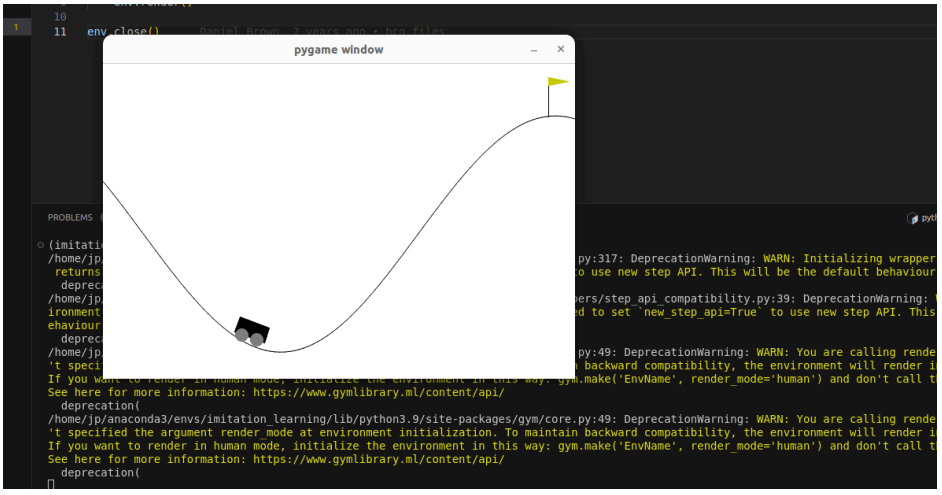
Part 1

We notice that the car(**MountainCar-v0**) oscillates back and forth in a valley, trying to reach the flag at the top of the right hill. The code uses random actions (**env.action_space.sample()**), explaining the back-and-forth movement without apparent progress. The car rarely (if ever) reaches the goal, as random actions are not effective for such a goal-directed task.

MountainCar is challenging for reinforcement learning (RL) for several key reasons:

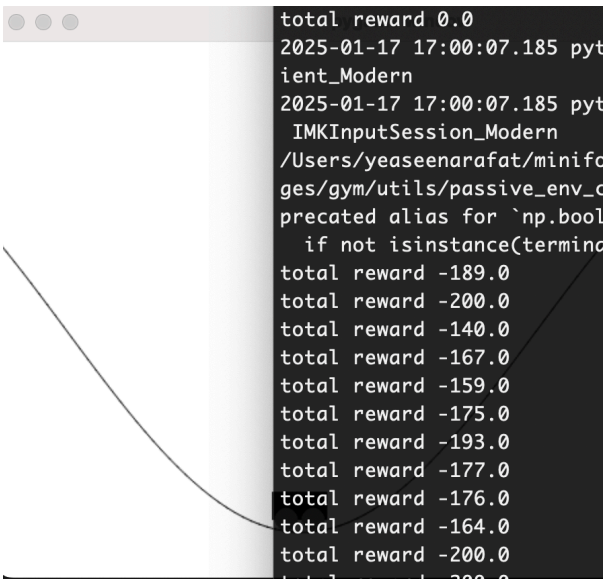
Sparse Rewards: The agent only receives a positive reward when it reaches the goal (flag) at the top of the hill. During all other times, it receives a small negative reward (-1 per timestep in the standard implementation). This creates a "sparse reward" problem where the agent gets no immediate helpful feedback, be it positive or negative, about whether its actions are helping it make progress toward the goal. Without sufficient exploration, the agent cannot find the sequence of actions required to escape the valley.

Local Optima Problem: In the MountainCar environment, a naive trial-and-error approach often gets stuck in a local optimum, where the agent learns to stay near the valley bottom to minimize short-term penalties (-1 per step) instead of building momentum to reach the goal. Building momentum requires temporarily moving away from the goal, which increases immediate penalties, but a random or greedy strategy fails to recognize this as necessary for long-term success. Without intermediate rewards or effective exploration strategies, the agent cannot escape the valley, leading to stagnation and suboptimal performance.



Part 2

We liked the strategy: **Left, right, left, right up the hill**. It gave us the best score: **-140**.



Part 3

It doesn't do a good job of imitating. It gets stuck often and gives us a -200 score each time.

Average Policy Return: -200
Max Policy Return: -200
Min Policy Return: -200

We tried all combinations of the following:

- learning_rate: 1e-2, 1e-3
- num_iterations: 100, 150, 200
- hidden_sizes: 64, 128

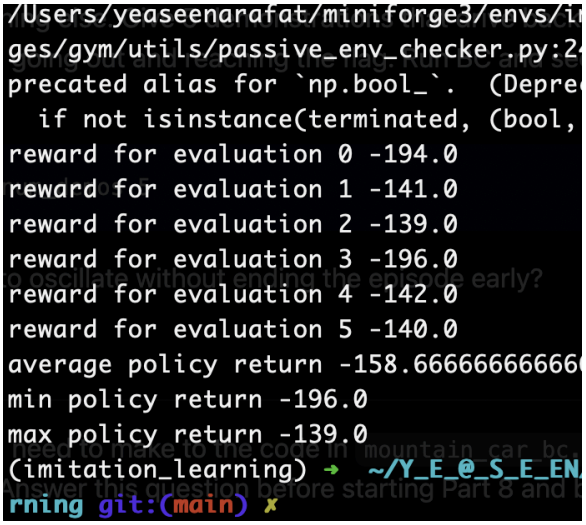
Part 4

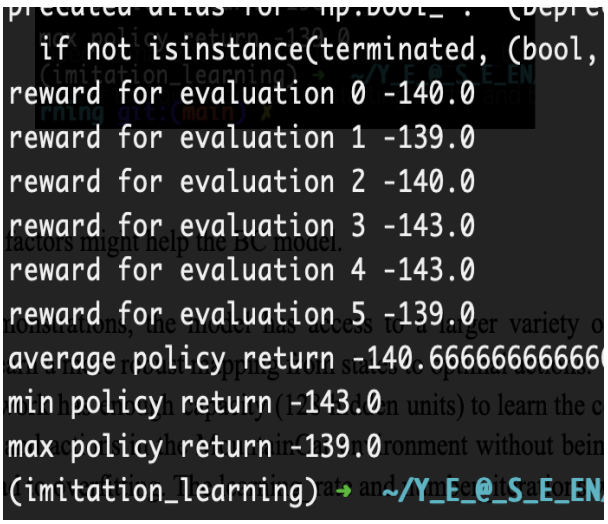
Wow! With the number of demos 5, the BC works better and does **ACTUALLY** escape the valley! This is quite interesting at this point.

We had the following parameters:

learning_rate: 1e-3
num_iterations: 100
hidden_sizes: 128

Here, we are providing two runs:





We think the following factors might help the BC model.

1. With more demonstrations, the model has access to a larger variety of state-action pairs, enabling it to learn a more robust mapping from states to optimal actions.
2. The neural network has enough capacity (128 hidden units) to learn the complex relationship between states and actions in the MountainCar environment without being excessively large, which could lead to overfitting. The learning_rate and number_iterations were standard here.

→ Does the agent copy the strategy you used?

Yes. The agen copied the strategy we demonstrated in the human demo part. We provided the strategy in all 5 demos: **Left, Right, Left, Right!**

Here, we have one thought on the BC's performance: all 5 demos had the same pattern which might have helped the BC model to imitate the human demo strategy.

Part 5

The policy is learning to mimic the average behavior across both demonstrations - that's why we see faster oscillations (combining the bad demo's behavior with the good demo's behavior) even though it's not achieving good performance. Bad demonstrations are problematic because behavioral cloning treats all demonstration data equally - it doesn't distinguish between "good" and "bad" behaviors. It tries to find a policy to explain all the observed state-action pairs, essentially averaging across them. This leads to compromised behavior that may capture surface features (like oscillation frequency) without learning the strategic elements (like building up momentum in the right way) needed for success.

One potential solution would be to weight demonstrations based on their "quality" during training. This would make the policy learn more strongly from successful demonstrations while reducing the influence of poor ones.

Part 6

No. We were not able to teach the BC model to imitate the human demos. This time the agent didn't follow the 5 human demonstration we put for oscillation. Maybe, the BC model didn't get understand the reward system because in the human demos we never put a scenario where rewards was good enough.

Part 7

Based on the BCO paper, here are the key changes needed to implement behavioral cloning from observation compared to standard behavioral cloning:

- Create a pre-demonstration phase where the agent explores randomly to collect state transitions and actions.
- Build a dataset of (st, at, st+1) tuples from this random exploration
- Train an inverse dynamics model to predict actions from state transitions
- Use a trained inverse dynamics model to infer the missing actions for each state transition
- Use inverse model to infer actions from demonstrated state transitions

Part 8

Initially, the BC model's performance dropped significantly with the inclusion of the inverse dynamics model. We kept the same parameters for the neural network of the indervse dynamics model.

Later, we changed the number of interaction from 10 to 100 and number of demos 5 to 12, then it really performed well, but only once. With the same setup, we weren't able to reproduce the same kind of result. The other results were worse. We think it was once successful because of the inverse dynamics model's training dataset's bias.