Exercise 1

Gengcong Yan - 1009903
ELEC-E8125 - Reinforcement Learning

October 4, 2021

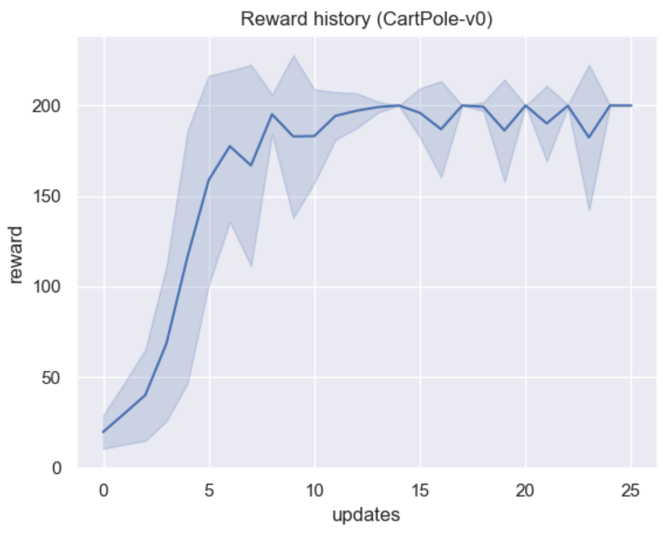


Figure 1: Reward History

1 Task 1

The reward history during training on a 200 timesteps is shown in Fg.1.

2 Question 1.1

After training with max 200 timesteps per episode, we test the model with max 200 and 500 timesteps, respectively. We know that the average reward in 200 timesteps is 200, which means the pole stay balanced to the last every time. the average reward in 500 timesteps is also 500 in this run test, making up 100% in maximum time limit.

From the result above, I think we can say that the model trained to balance for 200 timesteps can also mostly balance the pole for 500 timesteps. The task for balancing a pole is relatively simple compared to other complicated systems, experience gained from 200 timesteps training are sufficient to balance for a longer time.There is no new variable introduced into current system withe the time passing by.

3 Task 2

I repeat the experiment a few times, each time training the model from scratch with 200 timesteps and testing it for 500 timesteps. [**458.79,462.61,500.00, 497.21,477.85,499.88**] are different rewards I obtained. Although their performances are not exactly the same every time, they are relatively successful in completing the task.

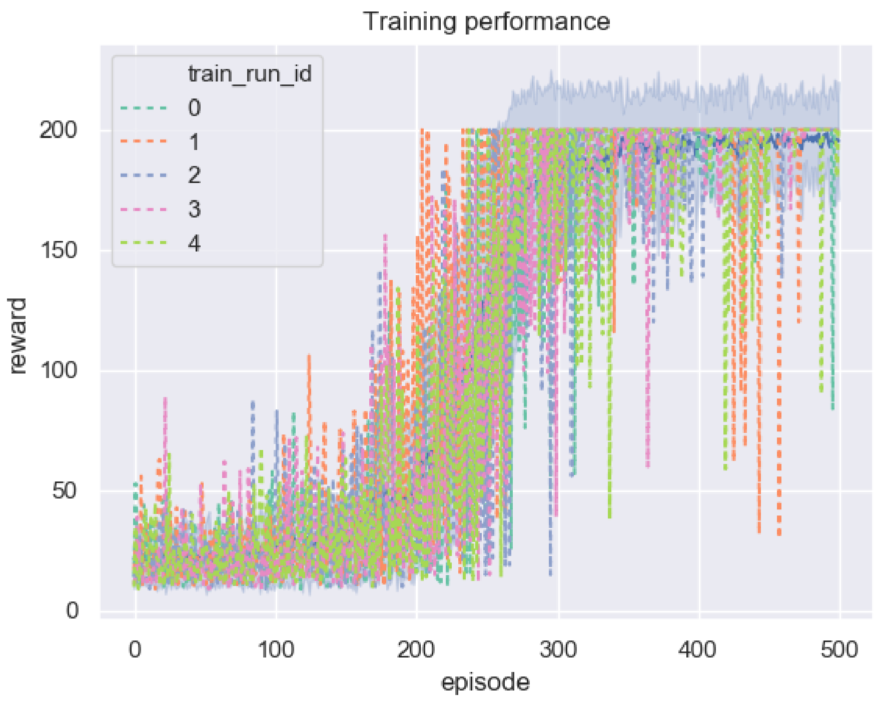


Figure 2: Reward History

4 Question 1.2

It depends on the situations. Because there is a lot of randomness in the entire training process of the model, the model will also modify its own policies according to the different situations encountered, so that it can achieve the best. When the training is insufficient, the performance of the models will be different. But when the training is long enough and the model has converged to the optimal policy, their performance should be very close to each other.

5 Question 2

There is a large variance between the runs of the script even after the same training process. I think the trade-off between exploration and exploitation causes such large variance.The stochasticity in each run makes the training process perform different decisions on how to balance exploration and exploitation.

This implies that comparison between different algorithms should be controlled carefully. We need to control irrelevant variables and reduce the impact of randomness on the results. we can train for a longer time in order to converge to optimal policy or consider to train repeatedly for average experiment results.

6 Task 3

6.1 Target point (1,1)

```python
# Get close to point(1,1)
def new_reward_point(state, action, next_state, env=None):
    target_x,target_y=1.0,1.0
    cpos=env.get_cartesian_pos(state)
    diff=np.sqrt((target_x-cpos[0])**2+(target_y-cpos[1])**2)

    # diff=np.abs(target_x-cpos[0])+np.abs(target_y-cpos[1])
    # cvel=env.get_ee_velocity(state, next_state)
    # print("_____")
    # print(diff)
    return min(4,1/diff)
```

In order to prevent the reward increase too large, I set a limit for maximum reward as 4. From the render test we can see, the end-effector will approach to target point(1,1),then stop at target or the area nearby. A small probability of instability occurs when rewards start to converge late in training. The reward will suddenly become very low, because the episode terminates when the agent reaches the target position. The reward for reaching the target will be somewhat lower than the reward for staying near the target. I will improve it afterwards

6.2 Maximum possible speed

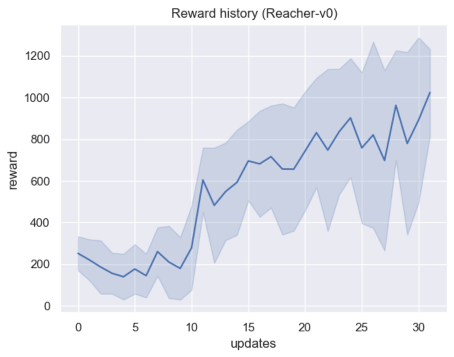
```python
# Achieve Maximum Velocity
def new_reward_velocity(state, action, next_state, env=None):

    cvel=env.get_ee_velocity(state, next_state)
    # print(cvel)
    # return 1+ (cvel[0])**2+(cvel[1])**2
    return (1+abs(cvel[0]))**2+(1+abs(cvel[1]))**2
```

when rendering the movement of manipulator under this reward function, I observed the joints are moving between 2 points with fast velocity. Maybe because the reward function is too simple, so it is not a fast circular motion around the center as I thought before.

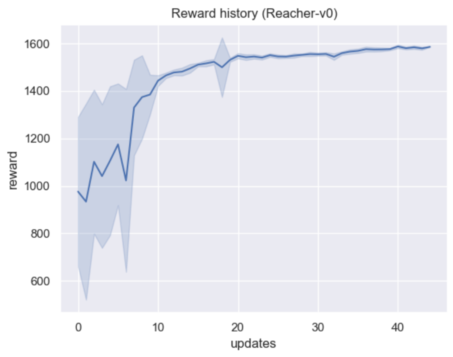
There are reward history of Reacher-v0 in Fig.3 including two reward functions

7 Task 4

We wrote the same reward function as in Task 3 in the *get_reward* function in *reacher.py*. Then we can get the reward history of Reacher-v0 in Fig.??, rewards and actions heatmap in Fig.4.



(a) Target(1,1)



(b) Max Velocity

Figure 3: Points in 2 datasets

```
def get_reward(self, prev_state, action, next_state):
        # Just hang around, living a peaceful life of a typical
            2D manipulator
        # TODO: Implement me! (either here or in train.py)

        target_x,target_y=1.0,1.0
        cpos=self.get_cartesian_pos(next_state)
        diff=np.sqrt((target_x-cpos[0])**2+(target_y-cpos[1])
            **2)

        return min(4,1/diff)
```

8 Question 3

From the plot in Fig4, we can analyze that, There are two spots in the reward heatamp,spots approximately at$(\pm 3.14, -1.88)$ are considered to the same one,because $sin(\pm\pi) = 0$, $cos(\pm\pi) = -1$, representing same angle in the graph. when Θ_0 and Θ_1 get closer to spots in heatmap, they can get more reward than other places. It's natural that the target point(1,1) is in upper-right of coordinate, so when the manipulator are around this area, it gets closer to target, resulting in big rewards. From the action heatmap, we can know the manipulator will take different action on joint 1 and 2 to make itself get closer to target point. I roughly estimated several scenarios:

1. Stop. When the end-effector is close to target, it stops around that area.
2. J1+. when the end-effector is far away from target, joint 1 rotate towards left to get closer to target.
3. J1-,J2+,J2-. when the end-effector gets relatively close to target, the joints will take different action to keep them stay in the area, avoid moving away from the target due to current speed.

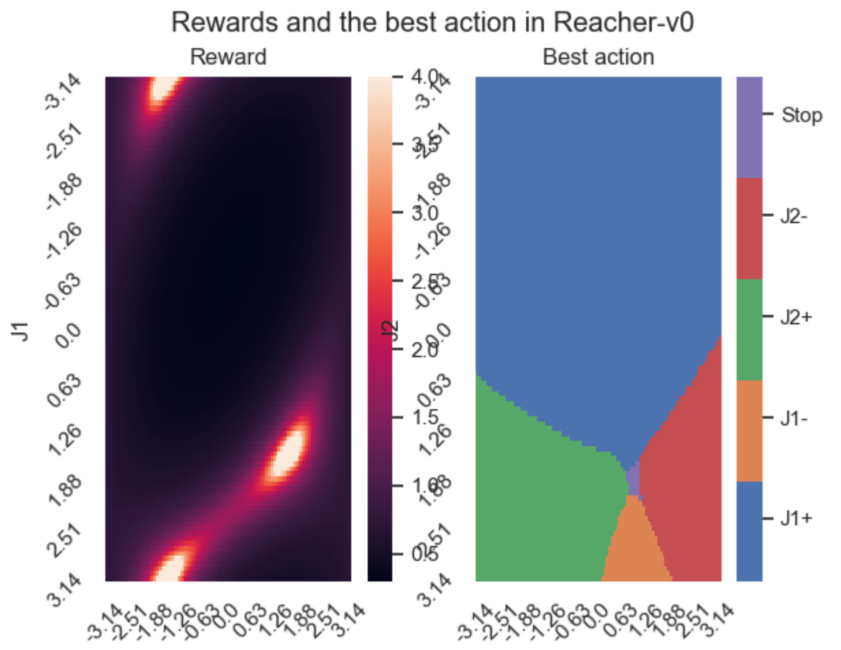


Figure 4: Reward and Best Action

There is a small problem with the current reward equation that can be seen in actions, because the episode terminates when the end-effector reaches the target position with less timesteps. so the total reward for reaching the target will be somewhat lower than the total reward for staying near the target until to the maximum timesteps. The reward for reaching the target point should be adjusted larger so that even if episode ends early, the reward is still greater than spending lot's of time around. I will revise latter.

8.1 Question 3.1

From the cartesian equation below we can know, with the angel of joints we can calculate the position of the end-effector of manipulator.

$$x = L_1 sin(\theta_0) + L_2 sin(\theta_0 + \theta_1) \tag{1}$$

$$y = -L_1 cos(\theta_0) - L_2 con(\theta_0 + \theta_1) \tag{2}$$

We can see the highest reward in heatmap is around $(1.5, 1.5)$ and$(\pm 3.14, -1.8)$,then we can know the position of end-effector is near target (1,1). And when the distance of target and end-effector increases, the rewards decrease, resulting in lowest reward area symmetrical in the third quadrant.

8.2 Question 3.2

I think the policy learn to reach the goal from every possible state. Regardless of which area the end-effector is in, there are corresponding actions that force it to approach the target point, and these actions do not contradict each other in adjacent areas, preventing it stuck in the middle of areas and unable to move.

9 Feedback

1. 15-20h
2. It took a lot more time than expected. Since I'm just learning about reinforcement learning and getting into code for the first time, some questions are a little bit confusing to me personally and I don't know how to start. I wish there were more examples to show us what to do during the learning process, and to give us a list of what code needs to be changed in which files for each question.
3. I learned a lot form this exercise.