Exercise 1

Nguyen Xuan Binh - 887799
ELEC-E8125 - Reinforcement Learning

July 10, 2023

Task 1 - 10 points

Train a model with 100 timesteps per episode. Use the command python train.py seed=1 max_episode_steps=100. Then test the model for 1000 timesteps with command: python train.py testing=true seed=1 max_episode_steps=1000 use_wandb=false – this will evaluate the trained model in 10 episodes and report the average reward (and episode length) for these 10 episodes. Export the training plot episodes-ep_reward from wandb (see README.md) and attach the plot in your report. Report also the average reward after testing the model.

This is the plot exported from the wandb website

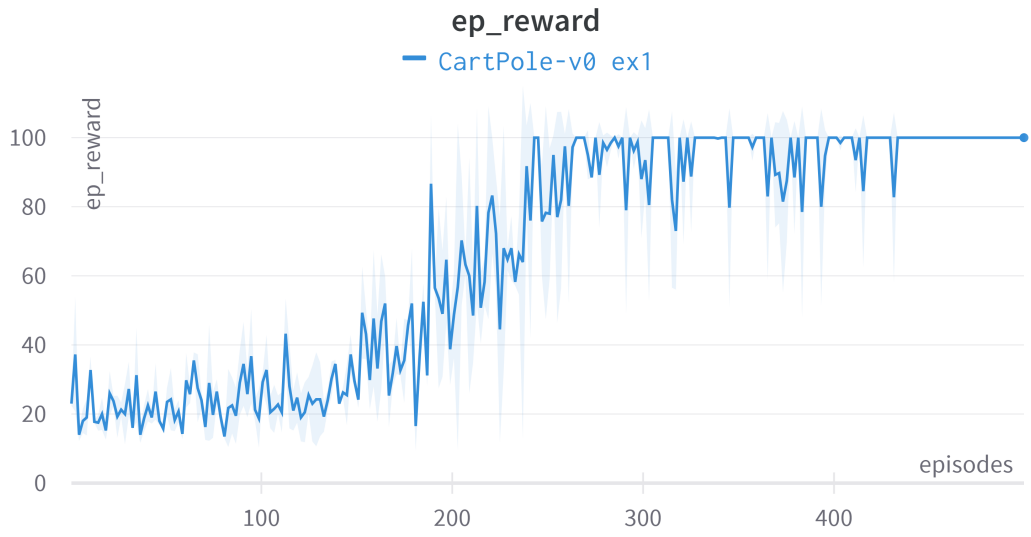


Figure 1: The episode rewards plot for the trained model

The average reward after testing the model is 849.6

Question 1.1 - 10 points

Test trained model from Task 1 five times with different random seeds. Did the same model, trained to balance for 100 timesteps, learn to always balance the pole for 1000 timesteps? Why/why not?

I run the testing for seed values of 100, 200, 300, 400, 500 and these are the results:

- Seed 100: Average test reward: 785.0
- Seed 200: Average test reward: 820.4
- Seed 300: Average test reward: 819.9
- Seed 400: Average test reward: 761.7
- Seed 500: Average test reward: 762.5

Out of the 10 episodes, the testing reward using falls in the range of 750-850. Therefore, my answer is that the pole is highly unlikely to be balanced for 1000 timesteps given any seed values, even though success do sometimes occur. The reason for the failures is because the pole is only trained for 100 timesteps, which we cannot guarantee its balance in the step 10 times more than the training timesteps. The cause of failure is most often related to the cartpole moving outside of the screen.

Task 2 - 10 points

Repeat the experiment in Task 1 five times, each time training the model from scratch with 100 timesteps and testing it for 1000 timesteps (in this Task it is enough to test with one seed). Use a different seed number for each training/testing cycle. Report the average test reward for each repeat.

I run the training and testing for different seed values of 100, 200, 300, 400, 500 and these are the average test rewards:

- Seed 100: Average test reward: 395.7
- Seed 200: Average test reward: 363.1
- Seed 300: Average test reward: 492.6
- Seed 400: Average test reward: 599.3
- Seed 500: Average test reward: 278.9

Question 2.1 - 15 points

Are the behavior and performance of the trained models the same every time? Why/why not? Analyze the causes briefly.

As we can see from above, each time I run with a different seed number, the average test reward will change by standard deviation of 50-100. Therefore, the behavior and performance of the trained models will differ every time a different seed is used. This is because the training environment is stochastic by nature. The environment initial state is different each time and the transition action for each time step is also probabilistic for different seed number, which explains the underlying stochasticity of the model.

Question 2.2 - 10 points

What are the implications of this stochasticity, when it comes to comparing reinforcement learning algorithms to each other? Please explain.

The stochasticity will first cause the variance in performance of the RL models, which we can see from above that each seed number results in a different average testing reward. Secondly, this randomness also causes the initial state to vary significantly, which many RL models are highly sensitive to. Additionally, the randomness may make the performance of RL models harder to quantify, as it is often unsure whether good performance is due to luck or due to the underlying algorithm. Therefore, when we compare RL algorithms, we should run multiple trials with different seed number and record the average performance plus its standard deviation (like Task 1). Additionally, we should use the same seed for comparing RL models to reproduce the same randomness and make them start from the same initial states so their performance is easier to compare than starting from different initial states.

Task 3 - 20 points

Edit the function get_reward in reacher.py, and write a reward function to incentivize the agent to learn the following behaviors:

1. Keep the manipulator rotating clockwise continuously (wrt angle θ_0). You can use a lower number of training episodes for this, e.g. train.py env=reacher_v1 train_episodes=200

2. Reach the goal point located in x = [1.0,1.0] (marked in red). Use at least 500 training episodes. Train one model for each behavior and include the model files in your submission. Also include the reward function implementations in your report (write them out or attach as screenshots). Hint: Use the observation vector to get the quantities required to compute the new reward (such as the position of the manipulator). You can get the Cartesian position of the end-effector with env.get_cartesian_pos(state).

This is the code of my get_reward function for two different goals

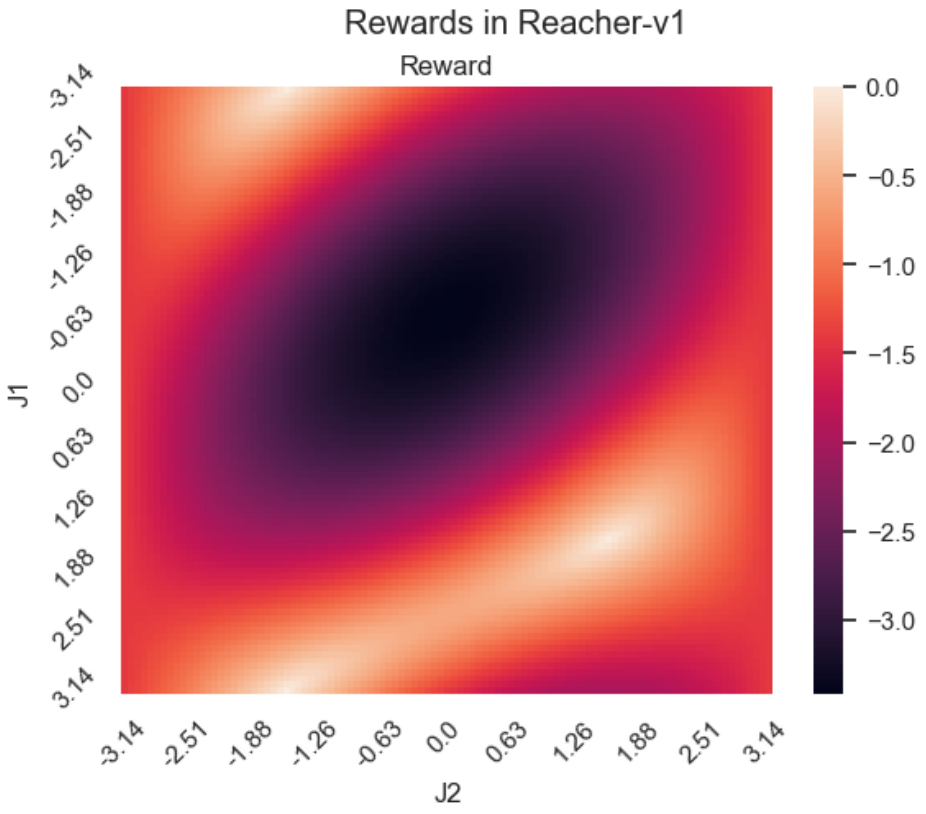
```
# Keep the manipulator rotating clockwise continuously (wrt
    angle $\theta_0$)
def get_reward(self, prev_state, action, next_state):
    if action == 1:
        return 1
    else:
        return 0

# Reach the goal point located in x = [1.0,1.0] (marked in red)
def get_reward(self, prev_state, action, next_state):
    next_position = self.get_cartesian_pos(next_state)
    euclidean_distance = np.sqrt(np.sum((next_position - self.
       goal)**2))
    # Since the algorithm tries to maximize while this distance
        needs to be minimized,
    # a negative sign is needed for the distance
    return -euclidean_distance
```

You can find two trained models attached in my submission as "Task3_rotate_params.pt" and "Task3_reach_goal_params.pt"

Task 4 - 10 points

Now, let us visualize the reward function for the second behavior (reaching the goal [1,1]). Plot the values of the second reward function from Task 3 and the learned best action as a function of the state (the joint positions). Use the plot_rew.ipynb script as a starting point. After plotting, answer the questions below. Include the two figures produced by plot_rew.ipynb in your report.

The two figures produced from the plot_rew.ipynb script are


Rewards in Reacher-v1
Reward
J1
J2
-3.14
-2.51
-1.88
-1.26
-0.63
0.0
0.63
1.26
1.88
2.51
3.14
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0


Best action in Reacher-v1
Best action
J1
J2
-3.14
-2.51
-1.88
-1.26
-0.63
0.0
0.63
1.26
1.88
2.51
3.14
Stop
J2-
J2+
J1-
J1+

Question 4.1 - 5 points

Where are the highest and lowest reward achieved?

The highest reward is achieved at the region with darkest color and the lowest reward is achieved at the region with lightest color in the rewards in Reacher-v1 graph. By some python code, I can determine that the state with highest reward is [-8.1681409e-01 4.4408921e-16] and the state with lowest reward is [-3.14159265 -1.57079633]

```
# Find highest and lowest reward state
# min and max are reversed because the reward is negative
max_idx = np.unravel_index(np.argmin(rewards), rewards.shape)
min_idx = np.unravel_index(np.argmax(rewards), rewards.shape)

max_state = np.array([state_range[max_idx[0]], state_range[
    max_idx[1]]])
min_state = np.array([state_range[min_idx[0]], state_range[
    min_idx[1]]])
```

Question 4.2 - 10 points

Did the policy learn to reach the goal from every possible state (manipulator configuration) in an optimal way (i.e. with lowest possible number of steps)? Why/why not?

In the best action in Reacher-v1 plot, we can see that the model only recognizes two actions, J1+ and J2+, which correspond to increasing values of two rotating joint angular parameters θ_0 and θ_1. This means that the model is always rotating anticlockwise. This behavior is optimal when the initial state is close to the goal in anticlockwise direction. However, this policy will not reach the goal in an optimal way if the initial state has the goal in the clockwise direction from the reacher. This is because the reacher will instead rotate anticlockwise in the long way to reach the goal instead of choosing the shorter way by rotating clockwise. This is because the actions of J1- and J2- are never recognized in this policy.

Feedback

Question 1: How much time did you spend solving this exercise?

I spend around 2 days or 16-20 hours, mainly for environment setup of the project code and reading the documentations on how to train, test the RL models and export wandb figures

Question 2: Did you find any of the particular tasks or questions difficult to solve?

None of the questions is particularly hard. The only quite unpleasant thing is I cannot find out a way of training multiple settings without wandb writing to the same params.pt file