Exercise 1

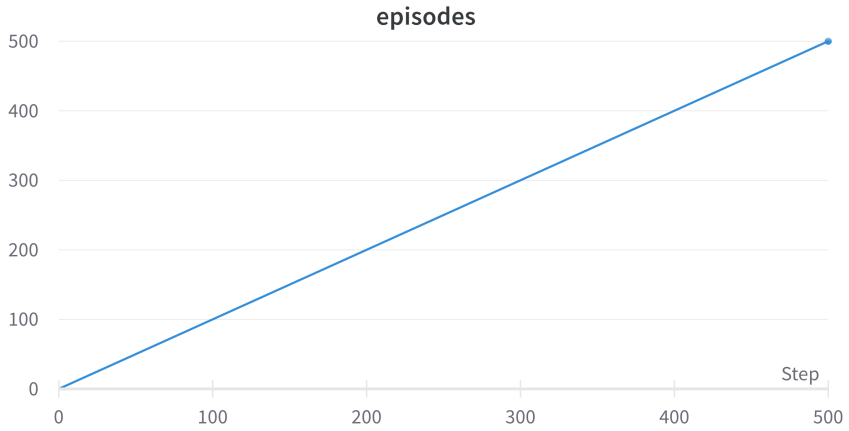
Marco Di Francesco - 100632815
ELEC-E8125 - Reinforcement Learning

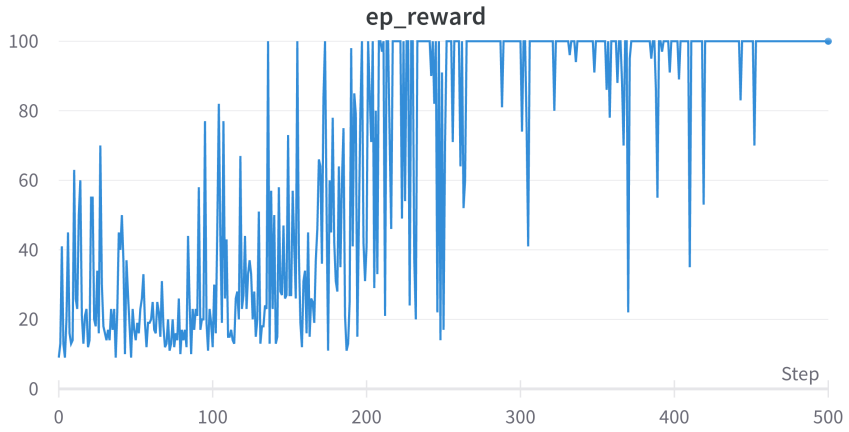
September 18, 2022

WARNING: in this assignment I may have used "we" when talking about making stuff. The assignment was completly done individually, I'm just used to write papers this way.

1 Task 1

- Average test reward: 927.8
- Average episode length: 927.8





1.1 Question 1.1

Tested with seeds:

- seed 10: 6/10 episodes balanced correclly for 1000 timesteps
- seed 20: 5/10 episodes balanced correctly for 1000 timesteps
- seed 30: 6/10 episodes balanced correctly for 1000 timesteps
- seed 40: 8/10 episodes balanced correctly for 1000 timesteps
- seed 50: 5/10 episodes balanced correctly for 1000 timesteps

Did the same model, trained to balance for 100 timesteps, learn to always balance the pole for 1000 timesteps? Why/why not?

The model is stocastical and does not produce always the same results. In our case the model was not able to always balance the pole correctly mainly because was trained to perform correctly only the first 100 iterations, while in test we are making 10 times the amount of iterations.

2 Task 2

Trained with 5 different seeds, tested each train with seed=1.

Average test reward:

- Run seed 2: Average test reward: 1000.0 episode length: 1000.0
- Run seed 4: Average test reward: 397.7 episode length: 397.7
- Run seed 6: Average test reward: 566.3 episode length: 566.3
- Run seed 8: Average test reward: 447.8 episode length: 447.8
- Run seed 10: Average test reward: 300.4 episode length: 300.4

2.1 Question 2.1

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

The model is stocastical, for this reason the trains are do not always have the same performances.

The results of the model for the test runs depended on how well the training policy was performing after the last policy update during the training phase. Practically speaking, the performance of the models depended on the strategy the model learned, for instance in the run with seed 2 we can see that the model learned to keep the pole as central as possible during both training and test phase, while all the other runs shifted slowly in the right or left directions, having perfect performances in the short term (100 timesteps) but low performance in the long term (1000 timesteps).

2.2 Question 2.2

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

Stochasticity plays a role in the exploration during the training phase allowing the model to explore making different choices when encontering the same state. As said in the previous question, this lead to learn different strategies such that we can have some models that perform very well in some tasks, like learning 1000 timesteps instead of only 100 timesteps.

3 Task 3

3.1 Task 3.1

The code measures the angle in radiants from the initial position and does not reset it when making a complete rotation, for this reason we can simply make the difference between previous and next state for θ_0 such that we always try to decrease this number.

```
def get_reward(self, prev_state, action, next_state):
    return prev_state[0] - next_state[0]
```

3.2 Task 3.2

The goal in this case is to reduce the distance from the red dot and the position (1, 1). In this case we used euclidean distance to measure this distance, and because we want to maximize the reward by being closest the point, we are making the negation of the distance.

$$-\sqrt{(1-x)^2 + (1-y)^2}$$

```
def get_reward(self, prev_state, action, next_state):
    x, y = self.get_cartesian_pos(next_state)
    distance =  np.sqrt((1-x)**2 + (1-y)**2)
    return -distance
```

4 Task 4

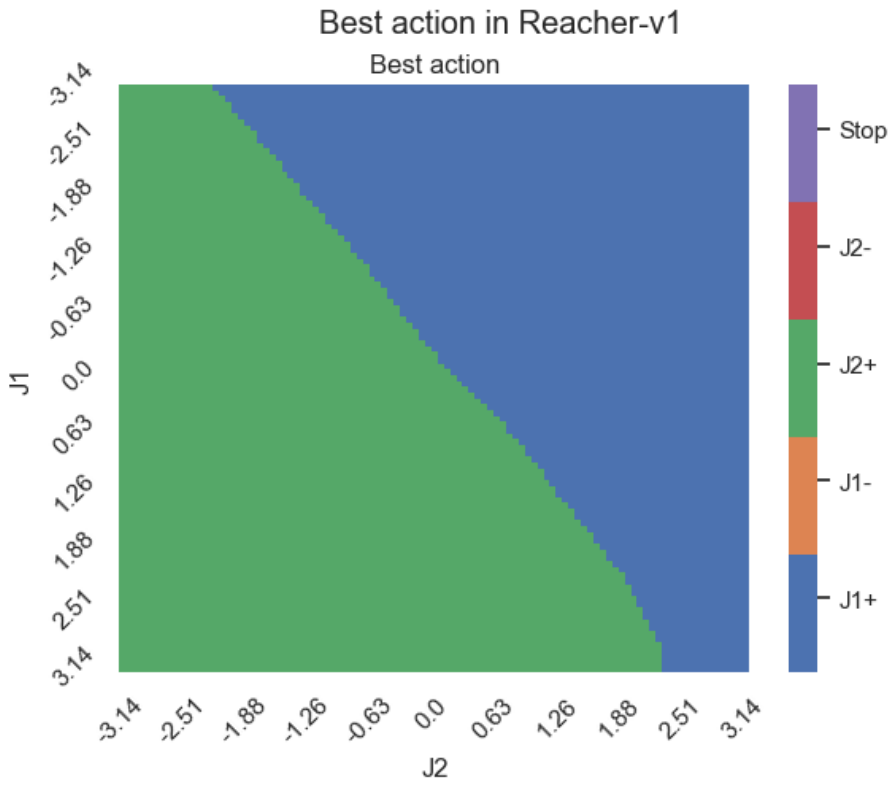
Trained for 1000 episodes.

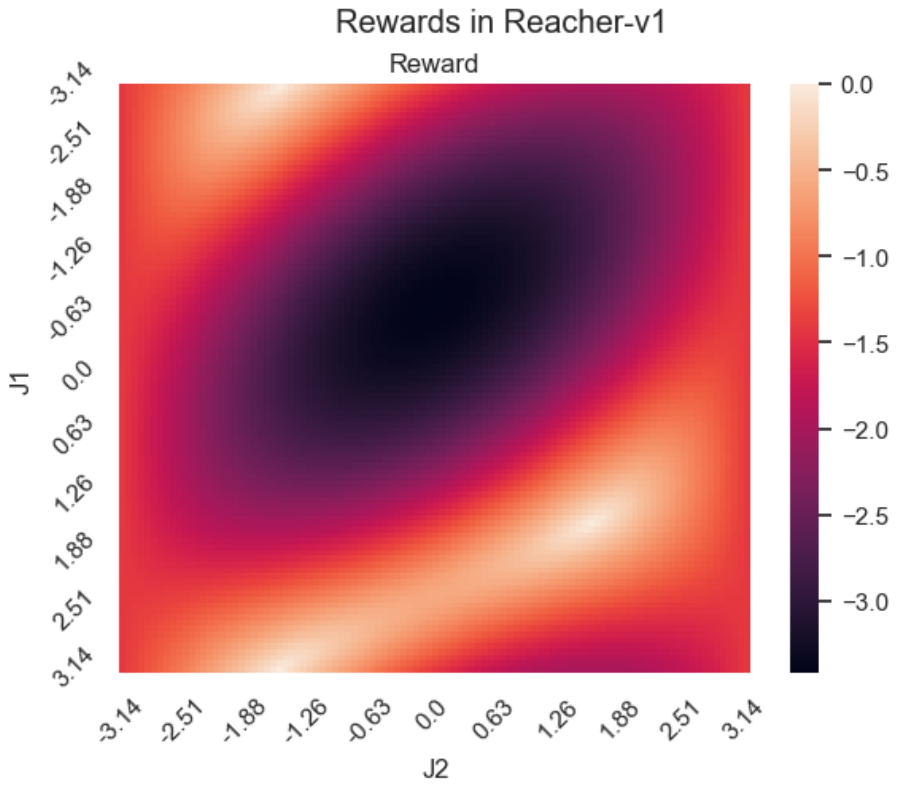
In order not to overcoplicate code writing code from scratch we looked at the implementation of the function get_action in agent.py, and we can see that the argmax function is used, so the same is done in the notebook.

```
actions[i, j] = action_probs.argmax()
```

To get the rewards we took the reward function from task 3.2.

```
x, y = reacher.ReacherEnv().get_cartesian_pos(state)
distance = np.sqrt((1 - x) ** 2 + (1 - y) ** 2)
rewards[i, j] = -distance
```





4.1 Question 4.1

Where are the highest and lowest reward achieved?

```
1 print("Min Reward:", rewards.min())
2 print("Max Reward:", rewards.max())
```

- Min Reward: -3.4138
- Max Reward: $-2.482 * 10^{-16}$ (very close to 0)

4.2 Question 4.2

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?

The policy was not able to learn to reach the optimal state in the optimal way for each possible state. Looking at the best action plot we can see that we can see that the best action is always to increase either θ_0 or θ_1 that translates to go counter clockwise, and this is obviously not the best action to take. For instance if we start at position [0, 1] instead of going right to [1, 1], it will go left.

5 Feedback

5.1 Question 1

How much time did you spend solving this exercise?

7 hours.

5.2 Question 2

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

No.