884 Pset1

Francis McCann

March 2020

1 Problem 1

1.1 Reward Function

The reward function I designed for this task was very simple:

$$r(t) = -1((goal_x - x)^2 + (goal_y - y)^2) + solvedbonus$$

As you can see the reward function consists of two terms. The first term serves to avoid sparse reward by providing a penalty for each state that is directly proportional to the euclidean distance of the robot arm from the goal state. The fact that it is squared also more heavily penalizes being far away vs close (not a linear relationship) In addition we have a term to add a large bonus once the goal state is achieved in order to stop training.

1.2 Video

Here is a link to the video for the first part: `https://youtu.be/khI_VPAY7JI`

2 Problem 2

2.1 First Question

The reward function I designed in Problem 1 does end up succeeding for this problem. However it does not succeed in the best way. In the beginning, it struggles to get around the wall and crashes into it. In order to overcome this we would need to encourage are agent to explore a little more and not just take the most direct path to the goal state. In order to do this we could either penalize proximity to the wall or encourage exploration of less valuable states.

2.2 Second Question

In order to outperform the previous reward function I designed the following one:

$$r(t) = r(t) = -1((goal_x - x)^2 + (goal_y - y)^2) + solvedbonus + wallPenalty$$

where the wallPenalty is given by: -1 if the euclidean distance is greater than 5 cm. If it is not then the penalty is 0. This makes it so that our reward function provides an incentive for the agent to avoid the wall without having to calculate that the area near the wall is low value using a lot of experience or pretty deeply discounted states.

While this fix does provide a nontrivial improvement it does have some disadvantages. We won't always have advantage of every obstacle in our environment so this wouldn't generalize well to walls that were not in the exact same position. Our old function in a scenario such as this one would still explore that are if there was no longer a wall whereas this approach wouldn't. This could be problematic if the position of the wall and the goal state were flipped for example.

2.3 Videos

As you can see this reward function still finds a solution, however it stays around the edges of the box. It would have probably converged if trained longer but I trained it the same amount as the new reward function and since it took longer to get around the wall it wasn't able to learn the fine details in that timeframe.

The second video Reaches the goal more smoothly and stays there.

2.4 plots

If you look at the plots you will see that the rewards keep growing for the new reward function. That is because the goal state provides a pretty large reward and I couldn't figure out a safe way to kill training based on an immediate reward so I was looking at average reward and the new reward function solves the problem much quicker. Thus it just stays in the reward state and racks up a ton of rewards. If I let it train long enough it would reach a plateau of speed and level off, however for the sake of this pset I did not think that was necessary.

3 Problem 3

The reward function I used for this problem was:

$$r(t) = -(0.5) * proximitypenalty - objectgoalpenalty + goalstate$$

Where the terms are as follows:

$$objectgoalpenalty = ((goal_x - obj_x)^2 + (goal_y - obj_y)^2)$$
$$proximitypenalty = (arm_x - obj_x)^2 + (arm_y - obj_y)^2$$

$goalstate = 10$ if object goal penalty $<= 0.025^2$ else 0

Object goal penalty and proximity penalty are both euclidean distance penalties similar to the ones specified above. Since the terms are square being farther away is penalized much more heavily. In addition I weighted the terms to place a larger emphasis on moving the object towards the goal instead of just having the robot arm close to the object.