# CIS 419/519: Homework 6

Jiatong Sun

04/11/2020

Although the solutions are entirely my own, I consulted with the following people and sources while working on this homework: $YuchenSun$
https://pytorch.org/docs/stable/

## 1 Reinforcement Learning I

The reward function is fine because for a situation to have a robot run a maze can break down naturally into episodes and thus, simply using the sum of rewards is reasonable.

However, if the map is large and the episode length is not long enough, the agent will probably fail to discover the exit for many episodes. After running the agent for a while, the agent may never know there can be a total reward better than 0. Besides, since every policy it has already tried gives the same total reward 0, the agent will have no clue how to choose its policy for the following episode. Therefore, the policy is not improved.

There are two easy ways to solve the problem: increase the episode length and/or give non-goal state a value -1. This means the states that have been visited a lot will get worse and worse values so the agent will want to avoid that state. In this way, the agent can eventually reach the goal.

## 2 Reinforcement Learning II

According to equation for the expected discount return and value of a state:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad V^{\pi}(s) = \mathbb{E}[R_t | s_t = s]$$

If we add a constant $C$ to all the rewards, the total reward changes into

$$\begin{aligned} R'_t &= \sum_{k=0}^{\infty} \gamma^k (r_{t+k+1} + C) \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} + \sum_{k=0}^{\infty} \gamma^k C \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} + \frac{C}{1-\gamma} \\ &= R_t + \frac{C}{1-\gamma} \end{aligned} \tag{1}$$

The state value changes into

$$\begin{aligned} V^{\pi'}(s) &= \mathbb{E}[R'_t|s_t = s] \\ &= \mathbb{E}[R_t + \frac{C}{1-\gamma}|s_t = s] \\ &= \mathbb{E}[R_t|s_t = s] + \frac{C}{1-\gamma} \\ &= V^{\pi}(s) + \frac{C}{1-\gamma} \end{aligned} \tag{2}$$

Define $K \triangleq \sum_{k=0}^{\infty} \gamma^k C = \frac{C}{1-\gamma}$, then $K$ is a constant.

(a) From equation 1 and equation 2, we know that adding a constant to all the rewards only has the effect of shifting the total reward and state value, so the relative reward and relative state value remain the same.

Because we choose the policy that maximize the total reward, we still get the same best policy as before. So **only the interval between those rewards matters**.

The signs of those rewards are not important. For example, if we change {+1 for a goal, -1 for a collision} to {-1 for a goal, -3 for a collision}, the best policy gets worse rewards, but it will still be the best policy among all. Changing every reward to negative value essentially means finding a relative better policy among a group of bad policies.

(b) See the proof above.

(c) $K = \sum_{k=0}^{\infty} \gamma^k C = \frac{C}{1-\gamma},\ 0 < \gamma < 1$

# 3 Random policy for the MountainCar gym environment

(i) action space:

| Num | Action |
|---|---|
| 0 | push left |
| 1 | no push |
| 2 | push right |

observation space:

| Num | Observation | Min | Max |
|---|---|---|---|
| 0 | position | -1.2 | 0.6 |
| 1 | velocity | -0.07 | 0.07 |

state dimensionality:

| Num | State | Reward |
|---|---|---|
| 0 | non-goal | -1 |
| 1 | goal | 1 |

(ii) The mean reward obtained by the random policy over 10 episodes is -200.

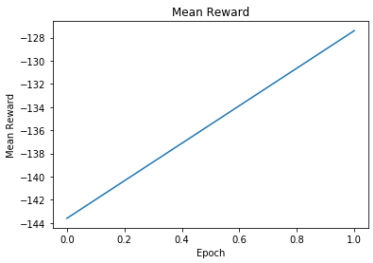
# 4 Train a Q-learner and generate expert trajectories

- The function `discretize()` attempts to change the continuous observation space into a discrete state space.

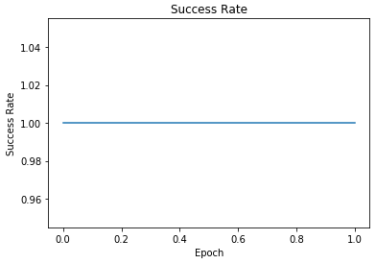
  We need this function because when we update the Q matrix, we need the index to be integers, so we cannot use the observation (which has float elements) directly. So we need the `discretize()` function to discretize the state first. It also makes sense in that we don't want the number of states to be infinite, and this function helps reduce the number of states to finite.

- If we increase any element of the `discretization` argument, it will divide the corresponding observation into more intervals and thus can make the Q-learning model more accurate, but meanwhile the model size will increase as well.

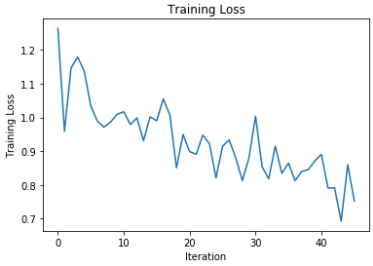
# 5 Train an imitation policy & Implement DAgger



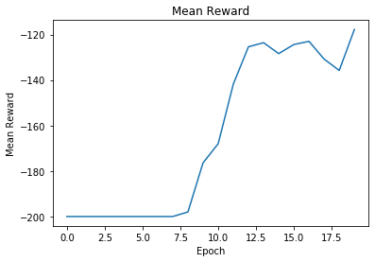

(a) $\pi_0$ mean reward



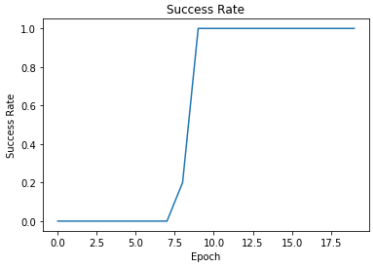

(b) $\pi_0$ success rate



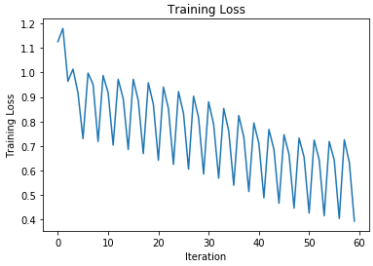

(c) $\pi_0$ training loss



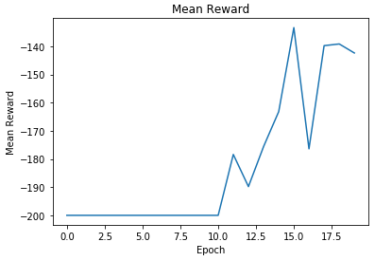

(d) $\pi_1$ mean reward



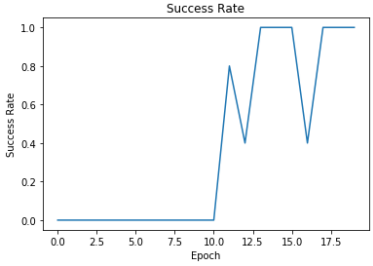

(e) $\pi_1$ success rate



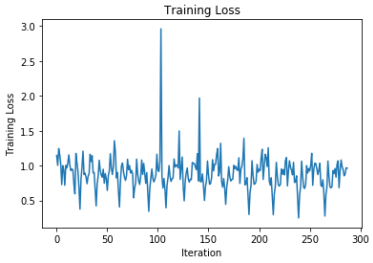

(f) $\pi_1$ training loss




(g) $\pi_2$ mean reward




(h) $\pi_2$ success rate




(i) $\pi_2$ training loss

Analysis:

1. Mean Reward: For $\pi_0$, the mean reward is a straight line because there are only two epochs so the tests can only be run for two times and the mean reward rises up immediately. In comparison, the mean rewards of $\pi_1$ and $\pi_2$ increase gradually, while the latter one has a larger oscillation. This is because the dataset is changing within every epoch for $\pi_2$.

2. Success Rate: The success rate curve for $\pi_0$ remains a line and the rate keeps 100% after the first epoch. In comparison, the success rates of $\pi_1$ and $\pi_2$ increase to a high level after several epochs, with $\pi_1$ keeping steady and $\pi_2$ occasionally oscillating.

3. Training Loss: The training losses of $\pi_0$ and $\pi_1$ goes down gradually, but the second one is oscillating much stronger because only two expert episodes data are feeded into $\pi_1$. On the contrary, the training loss for $\pi_2$ keeps the same level all the time. There are several spikes appearing periodically, indicating that the training loss is decreasing within every epoch, but since the dataset is aggregated in every dagger iteraion, the training loss becomes large again when a new epoch begins.