CS224R Spring 2023 Homework 2
Online Reinforcement Learning

Due 5/3/2023

SUNet ID: ishancs (06681175)
Name: Ishan Sabane
Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1: Impact of Reward Functions

1. Design a reward function for the Quadruped task such that the agent walks in a clockwise circle (watch `mujoco_mpc/videos/part1.avi` for an example of the desired behavior). The structure of the reward function for the Quadruped task is:

$$r_t = -w_0 \cdot r_{t,\text{height}} - w_1 \cdot r_{t,\text{pos}}(w_2, w_3) + c$$

   where

   - $r_{t,\text{height}}$ is the absolute difference between the agent's torso height over its feet and the target height of 1,
   - $r_{t,\text{pos}}(w_2, w_3)$ is the ℓ^2 norm of the difference between the agent's torso position and the goal position, which moves at each time-step according to the desired *walk speed* w_2 and *walk direction* w_3, and
   - c consists of other reward terms for balance, effort, and posture.

   You will design the reward function by choosing values for w_0, w_1, w_2, and w_3. Here is how you can run the Quadruped task with $w_0 = w_1 = w_2 = w_3 = 0$:

```
./build/bin/mjpc --task="Quadruped Flat" --steps=100 \
    --horizon=0.35 --w0=0.0 --w1=0.0 --w2=0.0 --w3=0.0
```

   The program will run the simulator for the specified number of time-steps and save a video in the `mujoco_mpc/videos/` directory.

   Fill in Table 1 with the parameters of your reward function.

Parameter	Value
w_0	**1.00**
w_1	**1.00**
w_2	**0.20**
w_3	**-0.50**

Table 1: Parameters of your reward function.

2. In this next part, you will see how different reward functions impact the agent's behavior in the In-Hand Manipulation task. The structure of the reward function for the Hand task is:

$$r_t = -w_0 \cdot r_{t,\text{cube pos}} - w_1 \cdot r_{t,\text{cube ori}} - w_2 \cdot r_{t,\text{cube vel}} - w_3 \cdot r_{t,\text{actuator}}$$

   where

- $r_{t,\text{cube pos}}$ is the ℓ^2 norm of the difference between the cube's position and the hand palm position,
- $r_{t,\text{cube ori}}$ is the ℓ^2 norm of the difference between the cube's orientation and the goal orientation,
- $r_{t,\text{cube vel}}$ is the ℓ^2 norm of the cube's linear velocity, and
- $r_{t,\text{actuator}}$ is the ℓ^2 norm of the control inputs.

For each part below, you will watch the videos located in `mujoco_mpc/videos`. Then, in 1-2 sentences, describe the agent's behavior and why the reward function leads to this behavior.

(a) Watch `mujoco_mpc/videos/part2a.avi`, which was generated with the parameters $w_0 = 20$, $w_1 = 3$, $w_2 = 10$, and $w_3 = 0.1$:

```
./build/bin/mjpc --task="Hand" --steps=100 \
    --horizon=2.5 --w0=20.0 --w1=3.0 --w2=10.0 --w3=0.1
```

Describe the agent's behavior and why the reward function leads to this behavior in 1-2 sentences.

In this demo, the hand is able to move it's fingers to turn the cube in some form to achieve the correct orientation of the cube without the cube falling from the hand. This is because in the reward function; w0 makes sure that the cube does not slip from the hand, w1 makes sure that the cube turns to achieve the correct orientation, w2 controls the speed at which the cube is moved and finally w3 makes sure that the control inputs are small in magnitude to achieve smooth transitions.

(b) Watch `mujoco_mpc/videos/part2b.avi`, which was generated with the parameters $w_0 = 20$, $w_1 = 3$, $w_2 = 10$, and $w_3 = 1$:

```
./build/bin/mjpc --task="Hand" --steps=100 \
    --horizon=0.25 --w0=20.0 --w1=3.0 --w2=10.0 --w3=1.0
```

Describe the agent's behavior and why the reward function leads to this behavior in 1-2 sentences.

In this demo, the hand does not move at all. This is due to the fact that the weight of the reward for control inputs is larger. Hence the control inputs tend to become zero to maximise the term - w3*(l2 norm of control inputs) to zero which maximizes the rewards!

(c) Watch `mujoco_mpc/videos/part2c.avi`, which was generated with the parameters $w_0 = 0$, $w_1 = 0$, $w_2 = 0$, and $w_3 = 1$:

```
./build/bin/mjpc --task="Hand" --steps=100 \
    --horizon=2.5 --w0=0.0 --w1=0.0 --w2=0.0 --w3=1.0
```

Describe the agent's behavior and why the reward function leads to this behavior in 1-2 sentences.

In this demo, the cube simply falls from the palm of the hand as the hand does not even try to keep the cube on the hand. This is because the weights for the positon, orientation and velocity of the cube are zero. Hence, it does not matter to do anything to the cube for the hand as it does not get any reward. Moreover, as we have a unit weight to the control inputs, the reward is maximized as the control inputs become zero.

Problem 2:

1. Optimizing behaviour in environments with sparse reward functions is difficult due to limited reward supervision. To alleviate that, we have provide the agent with 5 successful demonstrations, which we will use to pre-train with behaviour cloning. In **ac.py** complete the **pretrain** function of the **PixelACAgent** class to train **both** the policy and encoder using the supervised behaviour cloning loss. That is, given state-action pairs o_t, a_t optimize the loss

$$\mathcal{L}_{\pi_\theta, f_\theta}(o_t, a_t) = -\log \pi_\theta(a_t | f_\theta(\text{aug}(o_t)))$$

   with respect to both π_θ and f_θ.

2. In the second part, we will try to improve the performance of the policy with additional fine-tuning with reinforcement learning. Your implementation will be in the **update** method of the **PixelACAgent** class.

   - We begin by implementing the critic update using the standard Bellman objective. Consider transitions $(o_t, a_t, r_t, o_{t+1}, \gamma)$ and implement the following steps:
     (a) Process the observations o_t, o_{t+1} through the augmentation and encoder networks to obtain features $f_\theta(\text{aug}(o_t))$ and $f_\theta(\text{aug}(o_{t+1}))$.
     (b) Sample next state actions from the policy $a'_{t+1} \sim \pi_\theta(f_\theta(\text{aug}(o_{t+1})))$.
     (c) Compute the Bellman targets

$$y = r_t + \gamma \min\{\bar{Q}_{\theta^i}(f_\theta(\text{aug}(o_{t+1})), a'_{t+1}), \bar{Q}_{\theta^j}(f_\theta(\text{aug}(o_{t+1})), a'_{t+1})\}$$

     where $\bar{Q}_{\theta^i}$ and $\bar{Q}_{\theta^j}$ are two randomly sampled critics.
     (d) Compute the loss:

$$\mathcal{L}_{Q_\theta, f_\theta} = \sum_{i=1}^{N} (Q_{\theta^i}(f_\theta(\text{aug}(o_t)), a_t) - \textbf{sg}(y))^2$$

     where sg stands for the stop gradient operator.
     (e) Take a gradient step with respect to **both** the encoder and critic parameters.
     (f) Update the target critic parameters using exponential moving average

$$\bar{Q}_{\theta^i} = (1 - \tau)\bar{Q}_{\theta^i} + \tau Q_{\theta^i}$$

   - In the final part, we will improve the policy. First sample an action from the actor $a'_t \sim \pi_\theta(\textbf{sg}(f_\theta(\text{aug}(o_t))))$ and compute the objective:

$$\mathcal{L}_{\pi_\theta} = -\frac{1}{N} \sum_{i=1}^{N} Q_{\theta^i}(\textbf{sg}(f_\theta(\text{aug}(o_t))), a'_t)$$

     Take a gradient step on this objective with respect to the policy only.

- Once you are done, run the RL fine-tuning with

```
python train.py agent.num_critics=2 utd=1
```

Attach evaluation results from Tensorboard.

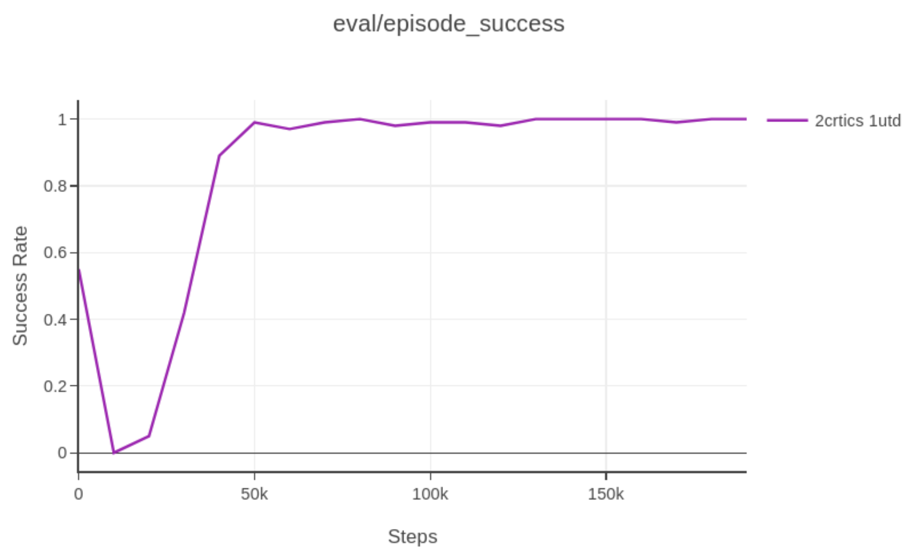

Figure 1: Successful Task Completion Rate as a function of Steps: 10 Critics and 5 Update-to-data

- In the final part, we will explore some optimization parameter choices. The update-to-data (UTD) ratio stands for the number of gradient steps we do, with respect to each environment step. So far we have used only 2 critics and UTD of 1. Repeat the previous part with:

```
python train.py agent.num_critics=10 utd=5
```

Attach your results and compare them to the previous question. Provide an explanation of why we observe these effects.

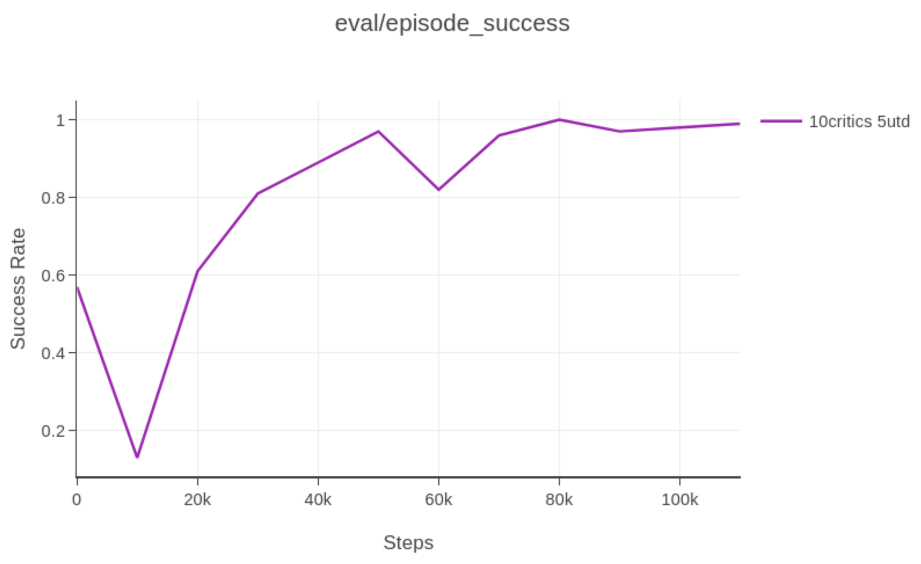

Figure 2: Successful Task Completion Rate as a function of Steps: 10 Critics and 5 Update-to-data

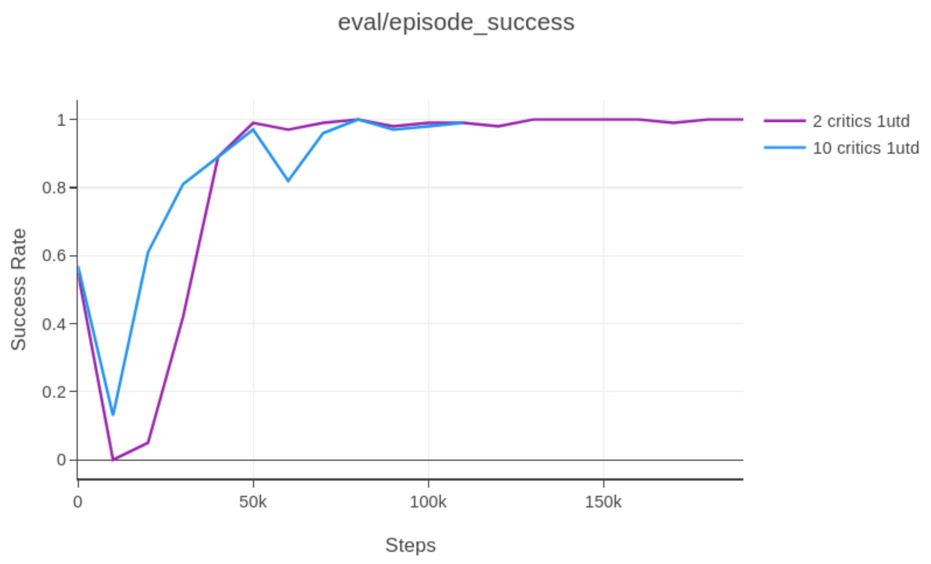

Figure 3: Comparison of the two models for the evaluation success rate.

Observations

From the above two plots, we see that the model with 10 critics and 5 update to data is able to achieve early successes compared to the model with just 2 critics and 1 update to data. This is due to the fact that higher number of critics prevent overestimation of the Q values and the more frequent data updates allow the model to learn faster from the data samples.

For the convergence, both the models converge to almost 100% success rate in almost the same number of iterations. This is because of the pretraining using the behaviour cloning agent which is able to optimize the model for the given five successful trajectories.