UNIVERSITY OF PENNSYLVANIA

ESE 650: LEARNING IN ROBOTICS

SPRING 2023

[04/16] HOMEWORK 4

DUE: 05/1 MON 11.59 PM ET

Changelog: This space will be used to note down updates/errata to the homework problems.

Instructions. Read the following instructions carefully before beginning to work on the homework.

- You will submit solutions typeset in LaTeX on Gradescope (strongly encouraged). You can use hw_template.tex on Canvas in the "Homeworks" folder to do so. If your handwriting is *unambiguously legible*, you can submit PDF scans/tablet-created PDFs.
- Please start a new problem on a fresh page and mark all the pages corresponding to each problem. Failure to do so may result in your work not graded completely.
- Clearly indicate the name and Penn email ID of all your collaborators on your submitted solutions.
- **For each problem in the homework, you should mention the total amount of time you spent on it. This helps us gauge the perceived difficulty of the problems.**
- You can be informal while typesetting the solutions, e.g., if you want to draw a picture feel free to draw it on paper clearly, click a picture and include it in your solution. Do not spend undue time on typesetting solutions.
- You will see an entry of the form "HW 4 PDF" where you will upload the PDF of your solutions. You will also see entries like "HW 4 Problem 1 Code" where you will upload your solution for the respective problems. **For each programming problem, you should create a fresh Python file**. This file should contain all the code to reproduce the results of the problem and you will upload the .py file to Gradescope. If we have installed Autograder for a particular problem, you will use the Autograder.
- **You should include all the relevant plots in the PDF, without doing so you will not get full credit.** You can, for instance, export your Jupyter

notebook as a PDF (you can also use text cells to write your solutions) and export the same notebook as a Python file to upload your code.

- **Your PDF solutions should be completely self-contained. We will run the Python file to check if your solution reproduces the results in the PDF.**

Credit. The points for the problems add up to 100. You only need to solve for 100 points to get full credit, i.e., your final score will be $\min(\text{your total points}, 100)$.

Problem 1 (Policy Gradient, 50 points). You will write the code for computing the optimal controller for taking a damped pendulum with dynamics

$$ml^2\ddot{x} + b\dot{x} + mgl\sin(x) = u$$

with $g = 9.8$, $m = 1$, $l = 1$ and $b = 0.1$ from its initial position at $x(0) = \dot{x}(0) = 0$ to the upright position $x = \pi, \dot{x} = 0$. The torque u has a constraint that only allows

$$|u| \leq 1.$$

Implement this controller using policy-gradients and a neural network. The reward at a state x and control u is

$$r(x, \dot{x}, u) = -\frac{1}{2}\left[(\pi - x)^2 + \dot{x}^2 + \frac{1}{100}u^2\right].$$

We have provided you some example code on Canvas (p1.py); feel free to modify this code as you wish. But read the comments inside the code carefully before beginning to write your solution.

(a) **(10 points)** Discuss how the stochastic controller $u_\theta(\cdot \mid x)$ is implemented in the code and how we compute the log-likelihood $\log u_\theta(u|x)$. Discuss how the constraint $|u| \leq 1$ is imposed in the code.

(b) **(30 points)** Implement code to train the policy using policy gradients.

(c) **(10 points)** Report the cumulative reward over 1000 time-steps as a function of parameter updates to θ. Modify your code to change the mass to $m = 2$. Evaluate the trained policy on this new dynamics and report the changes in the cumulative reward.

Problem 2 (Q-Learning, 50 points). You will write code for Q-learning with the DDQN trick in this problem for a simple environment called the CartPole https://stanford.edu/ jeffjar/cartpole.html. We have provided some example code that uses PyTorch for Q-learning. You need to fill in the functions for epsilon-greedy exploration and the optimization objective. The code is given at p2.py on Canvas; feel free to modify this code as you wish.

We will be using OpenAI Gym's version of CartPole, read the code at https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py to understand the environment. The class for the q-function in the example code looks as follows. The neural network s.m is a two-layer neural network with ReLU non-linearity. Notice that we have structured the network not as $q(x, u) : X \times U \to \mathbb{R}$ but instead as $q(x) : X \to U$. This way the neural network returns the q-value for all controls u with a single call to q_t.forward. You should implement the epsilon-greedy strategy to pick a control in the function q_t.control.

```
class q_t(nn.Module):
    def __init__(s, xdim, udim, hdim=16):
        super().__init__()
        s.xdim, s.udim = xdim, udim
        s.m = nn.Sequential(
```

```
                            nn.Linear(xdim, hdim),
                            nn.ReLU(True),
                            nn.Linear(hdim, udim),
                            )
    def forward(s, x):
        return s.m(x)

    def control(s, x, eps=0):
        # 1. get q values for all controls
        q = s.m(x)

        # eps-greedy strategy to choose control input
        # note that for eps=0
        # you should return the correct control u
        return u
```

Read the rollout function carefully. It takes a q-network and runs it for T timesteps to return a trajectory. You should add this trajectory to the replay buffer.

```
def rollout(e, q, eps=0, T=1000):
    traj = []

    x = e.reset()
    for t in range(T):
        u = q.control(th.from_numpy(x).float().unsqueeze(0),
                      eps=eps)
        u = u.int().numpy().squeeze()

        xp,r,d,info = e.step(u)
        t = dict(x=x,xp=xp,r=r,u=u,d=d,info=info)
        x = xp
        traj.append(t)
        if d:
            break
    return traj
```

You will code up the Bellman error minimization objective with the Double-Q network trick. Hint: You can use the following code to create a copy of the q-network. You can also modify the class q_t to create a copy of s.m inside it directly.

```
import copy
qc = copy.deepcopy(q)
```

Read the main function. We will create an environment e using the OpenAI Gym library, then initialize the q-network and create an optimizer (in this case Adam) to update the parameters of the q-network. The power of PyTorch lies in being able to call f.backward() to compute the gradient of whatever objective that depends on the parameters of the q-function. The call optim.step() udpates the parameters of the value-function.

```
if __name__=='__main__':
```

```python
    e = gym.make('CartPole-v0')

    xdim, udim =    e.observation_space.shape[0], \
                    e.action_space.n

    q = q_t(xdim, udim, 8)
    optim = th.optim.Adam(q.parameters(), lr=1e-3,
                          weight_decay=1e-4)

    # this is the replay buffer
    ds = []

    # collect few random trajectories with
    # eps=1
    for i in range(1000):
        ds.append(rollout(e, q, eps=1, T=200))

    for i in range(1000):
        q.train()
        t = rollout(e, q)
        ds.append(t)

        # perform sgd updates on the q network
        # need to call zero grad on q function
        # to clear the gradient buffer
        q.zero_grad()
        f = loss(q, ds)
        f.backward()
        optim.step()
        print('Log data to plot')
```

During training you should keep track of the average return of the network. You should also fill in this function that evaluates the learnt q-function on a new environment.

```python
def evaluate(q):
    # 1. create a new environment e
    # 2. run the learned q network for 100 trajectories on
    # this new environment and report the average discounted
    # return of these 100 trajectories
    return r
```

(30 points) Correctly code up all the methods above. Plot the following

1. **(10 points)** average return of the policy on the training environment across 10 episodes every 1000 weight updates of the q-network.
2. **(10 points)** average return of the policy on the evaluation environment every 1000 weight updates of the q-network.

The maximum achievable average reward for this environment is 200. You can call the render() function of the environment to see your trained policy on the Cartpole in action.