

UNIVERSITY OF PENNSYLVANIA
ESE 650: LEARNING IN ROBOTICS
SPRING 2021

[04/14] HOMEWORK 4

DUE: 04/29 THU 11.59 PM ET

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

- **04/15:** The damping coefficient of the pendulum in Problem 1 should be taken to be $b = 0.1$.

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 $\min(\text{your total points}, 100)$.

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

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

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

$$|u| \leq 1.$$

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

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

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

- 10 (a) **(10 points)** Discuss how the stochastic controller $u_\theta(\cdot | x)$ is implemented
 11 in the code and how we compute the log-likelihood $\log u_\theta(u|x)$. Discuss
 12 how the constraint $|u| \leq 1$ is imposed in the code.
- 13 (b) **(30 points)** Implement code to train the policy using policy gradients.
- 14 (c) **(10 points)** Report the cumulative reward over 1000 time-steps as a function
 15 of parameter updates to θ . Modify your code to change the mass to $m = 2$.
 16 Evaluate the trained policy on this new dynamics and report the changes in
 17 the cumulative reward.

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

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

```
32
33
34 class q_t(nn.Module):
35     def __init__(s, xdim, udim, hdim=16):
36         super().__init__()
37         s.xdim, s.udim = xdim, udim
38         s.m = nn.Sequential(
```

```

1         nn.Linear(xdim, hdim),
2         nn.ReLU(True),
3         nn.Linear(hdim, udim),
4     )
5
6     def forward(s, x):
7         return s.m(x)
8
9     def control(s, x, eps=0):
10        # 1. get q values for all controls
11        q = s.m(x)
12
13        # eps-greedy strategy to choose control input
14        # note that for eps=0
15        # you should return the correct control u
16        return u

```

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

```

19
20 def rollout(e, q, eps=0, T=1000):
21     traj = []
22
23     x = e.reset()
24     for t in range(T):
25         u = q.control(th.from_numpy(x).float().unsqueeze(0),
26                      eps=eps)
27         u = u.int().numpy().squeeze()
28
29         xp, r, d, info = e.step(u)
30         t = dict(x=x, xp=xp, r=r, u=u, d=d, info=info)
31         x = xp
32         traj.append(t)
33         if d:
34             break
35     return traj
36

```

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

```

41
42 import copy
43 qc = copy.deepcopy(q)

```

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

```

1
2 if __name__ == '__main__':
3     e = gym.make('CartPole-v0')
4
5     xdim, udim = e.observation_space.shape[0], \
6                 e.action_space.n
7
8     q = q_t(xdim, udim, 8)
9     optim = th.optim.Adam(q.parameters(), lr=1e-3,
10                           weight_decay=1e-4)
11
12     # this is the replay buffer
13     ds = []
14
15     # collect few random trajectories with
16     # eps=1
17     for i in range(1000):
18         ds.append(rollout(e, q, eps=1, T=200))
19
20     for i in range(1000):
21         q.train()
22         t = rollout(e, q)
23         ds.append(t)
24
25         # perform sgd updates on the q network
26         # need to call zero grad on q function
27         # to clear the gradient buffer
28         q.zero_grad()
29         f = loss(q, ds)
30         f.backward()
31         optim.step()
32     print('Log data to plot')
33

```

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

```

37
38 def evaluate(q):
39     # 1. create a new environment e
40     # 2. run the learned q network for 100 trajectories on
41     # this new environment and report the average discounted
42     # return of these 100 trajectories
43     return r
44

```

- 45 **(30 points)** Correctly code up all the methods above. Plot the following
- 46 1. **(10 points)** average return of the policy on the training environment across
 - 47 10 episodes every 1000 weight updates of the q -network.
 - 48 2. **(10 points)** average return of the policy on the evaluation environment every
 - 49 1000 weight updates of the q -network.

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