CS 277 (W24): Control and Reinforcement Learning
Exercise 3

Due date: Tuesday, February 20, 2024 (Pacific Time)

Roy Fox
https://royf.org/crs/CS277/W24

Instructions: In theory questions, a formal proof is not needed (unless specified otherwise); instead, briefly explain informally the reasoning behind your answers. In practice questions, include a printout of your code as a page in your PDF, and a screenshot of TensorBoard learning curves (`episode_reward_mean`, unless specified otherwise) as another page.

Part 1 Properties of linear–Gaussian systems (20 points)

Question 1.1 (7 points) Consider a deterministic uncontrolled LTI system with dynamics $x_{t+1} = Ax_t$, where A is an $n \times n$ transition matrix, that is only observable through a noiseless observation $y_t = Cx_t$, where C is a $k \times n$ observation matrix. The *observability matrix* of the system (A, C) is

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}.$$

We say that a state $x \neq 0$ is *unobservable* if, after starting at $x_0 = x$, we have only zero observations, i.e. $y_t = 0$ for all $t \geq 0$. Show that there exists an unobservable state $x \neq 0$ if and only if the rank of $\mathcal{O}$ is less than n. Hint: by the rank–nullity theorem, the rank and the dimension of the kernel ($\ker \mathcal{O} = \{x \mid \mathcal{O}x = 0\}$) sum to the dimension of the domain, in this case n.

Question 1.2 (7 points) A system (A, C) as in the previous question whose observability matrix has full column rank (i.e. rank n) is called *fully observable*. Show that a system is fully observable if and only if we can uniquely find what x_0 was at time 0 after seeing enough observations $y_0, \ldots, y_{t-1}$. Guidance: in one direction, use the fact that any full column-rank matrix M has a left inverse $M^\dagger M = I$. In the other direction, show that if $x_0 = x$ and $x_0 = x'$ induce the same observation sequence, then there exists an unobservable state.

Question 1.3 (6 points) When A itself isn't full-rank, i.e. it maps some states to 0, some information about x_0 may be lost by the dynamics and never become observable. On the other hand, only the current state x_t matters for control and future costs, so we may not actually need that information anyway. Show that, if $\ker \mathcal{O} \subseteq \ker A^n$, then we can uniquely find x_n from the observations $y_0, \ldots, y_{n-1}$.[1]

[1] As an aside, the other direction is also true, but you don't need to show it.

Hint: show that, under the question's assumptions, if $x_0 = x$ and $x_0 = x'$ induce the same $y_0, \ldots, y_{n-1}$, then they also induce the same x_n.

Part 2 Actor–Critic Policy Gradient (40 points)

In this part you'll implement an Actor–Critic Policy-Gradient algorithm. Download and read the code at https://royf.org/crs/CS277/W24/CS277E3.zip. Each part asks you to complete a code placeholder in file `a2c.py`.

Question 2.1 (10 points) Complete the placeholders marked as `Part 2.1` by writing PyTorch code that calculates the actor loss. The actor loss is a policy-gradient loss with pre-computed advantage estimates (`advantages`) plus a negative-entropy loss on the actor policy, weighted by `entropy_loss_coeff` (i.e. a slight push to *maximize* entropy).
Hint: You might want to use Distribution.entropy to compute the entropy.

Question 2.2 (15 points) Complete the placeholders marked as `Part 2.2` by writing PyTorch code that calculates the critic loss. The critic loss is a temporal-difference loss, the square error between the pre-computed value targets and the critic values.
In the function `update`, `traj` is part of a single trajectory, but in this assignment we will **not** assume that it's the entire episode. The batch contains tuples $(s_t, a_t, r_t, s'_t, \text{done}_t, \log \pi(a_t|s_t))$ for some consecutive steps $t \in \{t_1, \ldots, t_2\}$ in a trajectory.
Useful: (a) `Actor.critic`, a function that gets an array of observations and returns a same-size tensor of value predictions; (b) `dones`, a boolean array indicating episode termination in each time step (think: why is this useful here?); and (c) make sure to use `detach()` on tensors that are supposed to be the target.

Question 2.3 (5 points) Complete the placeholders marked as `Part 2.3` by writing PyTorch code that calculates for each step the discounted one-step advantages for the actor's policy gradient.
Hint: advantage should `detach()`.

Question 2.4 (10 points) Run your code on the `CartPole-v1` environment for 1,000,000 time steps and report the results.

```
python run.py --training-steps 1000000\
              --env CartPole-v1
```

Part 3 Generalized Advantage Estimation (40 points)

Recall the definition of the GAE[2] as

$$A^\lambda(s_t, a_t) = \sum_{\Delta t} (\lambda\gamma)^{\Delta t} A(s_{t+\Delta t}, a_{t+\Delta t}).$$

[2] https://arxiv.org/abs/1506.02438

Question 3.1 (10 points) Write down a mathematical expression for the advantage estimate $A^\lambda(s_t, a_t)$ using the rewards $r_t, r_{t+1}, \ldots$ and the value estimates $V_\phi(s_t), V_\phi(s_{t+1}), \ldots$.

Question 3.2 (15 points) Complete the placeholders marked as `Part 3.2` by using A^λ as the advantage estimates.

Question 3.3 (7 points) Run your code on `CartPole-v1` with a variety of λ values. To train the agent with GAE use

```
python run.py --training-steps 1000000\
              --env CartPole-v1\
              --GAE\
              --_lambda <lambda>
```

Visualize the results in TensorBoard, and attach the resulting plots.

Question 3.4 (8 points) Briefly discuss the results, including:

- What was the best value of λ in your experiments?
- What happens as $\lambda \to 0$?
- What happens as $\lambda \to 1$ in theory? What happens in practice?