Stanford **Spring 2023**

AA 203: Optimal and Learning-based Control
Homework #4
Due June 5 by 11:59 pm

Problem 1: Gain experience with data-driven learning in a "real world" setting.

Problem 2: Analyze the strengths and weaknesses of some fundamental reinforcement learning algorithms.

Problem 3: Reason through some concepts involved in implementing a neural-network-based actor-critic algorithm, and gain insight into modern deep reinforcement learning practice.

4.1 Widget sales. You have just purchased Widget Co., a small shop selling widgets. Congratulations! Widget Co. is in the business of buying widgets wholesale, and selling them to consumers at a markup. While the previous owners were losing money, you suspect that you can apply your knowledge of optimal control and reinforcement learning to turn the business around.

The shop is able to store between 0 and 5 widgets at a time. We write the number of widgets held in the shop on day t as s_t. Every day, you choose how many widgets to order from your supplier. You can order either zero widgets, a "half order" of 2 widgets, or a "full order" of 4 widgets. We write the number of widgets ordered to arrive on day t as a_t. A random number of customers (following an unknown distribution, though this distribution may be assumed to be consistent across all days) come to Widget Co. every day; each customer buys a widget if there are any available. We write the demand on day t as d_t, and assume $d_t \leq 5$. At the end of each day, you record a net profit r_t for that day.

(a) The previous owners were nice enough to give you the details of their last three years of operation. In particular, they have provided a dataset $\mathcal{D} = \{(s_t, a_t, r_t)\}_{t=1}^{T}$ containing records for each day t of the last three years. In `starter_widget_sales.py`, implement a Q-learning algorithm to learn tabulated Q-values from this dataset. Provide only the code you add to the provided starter file.

While you were busy with your model-free learning, your modelling-inclined intern noticed that the dynamics of the number of widgets in the shop each day are described by

$$s_{t+1} = f(s_t, a_t, d_t) \coloneqq \begin{cases} 0, & s_t + a_t - d_t < 0 \\ 5, & s_t + a_t - d_t > 5 \\ s_t + a_t - d_t, & \text{otherwise} \end{cases},$$

and the daily net profit is

$$r(s_t, a_t, d_t) = c_{\text{sell}} \min(s_t + a_t, d_t) - c_{\text{rent}} - c_{\text{storage}} s_t - g_{\text{order}}(a_t),$$

where $c_{\text{sell}} = 1.2$ is the price you set for each widget, $c_{\text{rent}} = 1$ is the fixed rent on your shop, $c_{\text{storage}} = 0.05$ is the cost for storing each widget overnight, and $g_{\text{order}}(a_t) = \sqrt{a_t}$ is the cost of ordering widgets from your supplier. The quantity $\min(s_t + a_t, d_t)$ is the "satisfied demand" on day t.

Moreover, after a few weeks of sales, your intern determined that the daily demand distribution for your widgets seems to be

$$d_t = \begin{cases} 0, & \text{with probability } 0.1 \\ 1, & \text{with probability } 0.3 \\ 2, & \text{with probability } 0.3 \\ 3, & \text{with probability } 0.2 \\ 4, & \text{with probability } 0.1 \end{cases}.$$

(b) In `starter_widget_sales.py`, implement value iteration to learn tabulated Q-values from the model your intern has provided. Submit only the code you add to the provided starter file. Also submit the plot generated by `starter_widget_sales.py` of Q-values from Q-learning compared to those from value iteration. What do you notice about the learned Q-values compared to those from value iteration? Why do you think this occurs?

(c) In `starter_widget_sales.py`, compute an optimal policy $\pi^*_{\mathrm{QL}}(s_t)$ based on your Q-learning work, and another optimal policy $\pi^*_{\mathrm{VI}}(s_t)$ based on your value iteration work. Report each optimal policy, simulate each one over five years, and compute the cumulative profit $\sum_{k=0}^{t} r_k$ for each day t and for each optimal policy. Submit only the code you add to the provided starter file. Also submit the plot generated by `starter_widget_sales.py` comparing the cumulative profits over time. What do you notice about the difference between the two cumulative profit trends? Why do you think this occurs?

4.2 Learning LQR. In this problem, you will study methods for learning an optimal policy for an unknown discrete-time dynamical system with an unknown cost function. All of the methods will use some form of episodic online reinforcement learning. That is, each method requires us to interact with the system, observe state transitions and incurred costs, and update a parametric policy with each observation. Specifically, at each time step t of episode i, we use the current policy to apply a control input $u_t^{(i)}$ that transitions the state from $x_t^{(i)}$ to $x_{t+1}^{(i)}$, and we incur an observed cost of $c_t^{(i)}$. Depending on the method, the policy is updated at each time step with the data $(x_t^{(i)}, u_t^{(i)}, c_t^{(i)}, x_{t+1}^{(i)})$, or at the end of the episode with all of the associated data $\{(x_t^{(i)}, u_t^{(i)}, c_t^{(i)}, x_{t+1}^{(i)})\}_{t=0}^{T-1}$, where T is the number of time steps in each episode.

Our goal is to learn a policy that minimizes the expected cost $\mathbb{E}[\sum_{t=0}^{T-1} \gamma^t c_t]$ with discount factor $\gamma \in [0, 1)$, where the expectation is taken over the initial system state, policy, and state transitions.

For this particular problem, the true underlying dynamics are linear time-invariant with the form

$$x_{t+1} = Ax_t + Bu_t,$$

with state $x_t \in \mathbb{R}^n$ and control input $u_t \in \mathbb{R}^m$. Moreover, $\mathbb{E}[x_0] = 0$ and $\mathrm{Var}[x_0] = \Sigma_x$. The true underlying stage cost function is

$$c(x, u) = x^\mathsf{T} Q x + u^\mathsf{T} R u,$$

with cost matrices $Q \succ 0$ and $R \succ 0$. To the learning agent, the matrices A, B, Q, and R are all *a priori* unknown.

We will investigate three approaches to learning an optimal policy: a model-based method, a model-free value-based method, and a model-free policy-based method. We provide high-level descriptions for each method, some references, and results below.

Model-based recursive least-squares: The first approach is a natural formulation of model identification adaptive control (MIAC). It consists of three steps performed at each time step to iteratively update estimates $(\hat{A}, \hat{B}, \hat{Q}, \hat{R})$ and compute a corresponding linear policy $u = Kx$ with parameter $K \in \mathbb{R}^{m \times n}$. Given the current state $x_t^{(i)}$, we:

1. Apply the control input $u_t^{(i)} = Kx_t^{(i)}$ and observe the incurred cost $c_t^{(i)}$ and next state $x_{t+1}^{(i)}$.
2. Fit $(\hat{A}, \hat{B})$ using recursive least-squares such that

$$(\hat{A}, \hat{B}) \leftarrow \operatorname*{arg\,min}_{A,B} \sum_{j=1}^{i-1} \sum_{\tau=0}^{T-1} \|Ax_\tau^{(j)} + Bu_\tau^{(j)} - x_{\tau+1}^{(j)}\|_2^2 + \sum_{\tau=0}^{t} \|Ax_\tau^{(i)} + Bu_\tau^{(i)} - x_{\tau+1}^{(i)}\|_2^2 + \eta(\|A\|_{\mathsf{F}}^2 + \|B\|_{\mathsf{F}}^2),$$

   where $\eta > 0$ is a regularization constant and $\|\cdot\|_{\mathsf{F}}$ denotes the Frobenius norm.
3. Fit $(\hat{Q}, \hat{R})$ using recursive least-squares such that

$$(\hat{Q}, \hat{R}) \leftarrow \operatorname*{arg\,min}_{Q,R} \left(\sum_{j=1}^{i-1} \sum_{\tau=0}^{T-1} \|(x_\tau^{(j)})^{\mathsf{T}} Q x_\tau^{(j)} + (u_\tau^{(j)})^{\mathsf{T}} R u_\tau^{(j)} - c_\tau^{(j)}\|_2^2 + \sum_{\tau=0}^{t} \|(x_\tau^{(i)})^{\mathsf{T}} Q x_\tau^{(i)} + (u_\tau^{(i)})^{\mathsf{T}} R u_\tau^{(i)} - c_\tau^{(i)}\|_2^2 + \eta(\|Q\|_{\mathsf{F}}^2 + \|R\|_{\mathsf{F}}^2) \right).$$

4. Solve the discounted discrete algebraic Ricatti equation

$$P = \hat{Q} + \gamma \hat{A}^{\mathsf{T}} P \hat{A} - \gamma^2 \hat{A}^{\mathsf{T}} P \hat{B} (\hat{R} + \gamma \hat{B}^{\mathsf{T}} P \hat{B})^{-1} \hat{B}^{\mathsf{T}} P \hat{A}$$

   for P, and update the policy $K \leftarrow -\gamma(\hat{R} + \gamma \hat{B}^{\mathsf{T}} P \hat{B})^{-1} \hat{B}^{\mathsf{T}} P \hat{A}$.

Model-free policy iteration (1): The second approach relies on the fact that for a linear time-invariant system with a quadratic cost function, we can write the Q function for a given policy $u = Kx$ in the form

$$Q_K(x, u) = \begin{pmatrix} x \\ u \end{pmatrix}^{\mathsf{T}} H_K \begin{pmatrix} x \\ u \end{pmatrix} = \begin{pmatrix} x \\ u \end{pmatrix}^{\mathsf{T}} \begin{bmatrix} H_{K,11} & H_{K,12} \\ H_{K,12}^{\mathsf{T}} & H_{K,22} \end{bmatrix} \begin{pmatrix} x \\ u \end{pmatrix},$$

where the Hessian $H_K \succ 0$ is an implicit function of the current gain K. This approach proceeds by alternating two steps of generalized policy iteration:

1. *Policy evaluation:* During episode i, apply the policy $u_t^{(i)} = Kx_t^{(i)} + \varepsilon_t^{(i)}$, where $\varepsilon_t^{(i)}$ is some zero-mean i.i.d. noise meant to excite the system, and observe $(c_t^{(i)}, x_{t+1}^{(i)})$. At the end of each time step t, fit $\hat{H}_K$ for the current policy K using recursive least-squares to minimize the temporal difference error such that

$$\hat{H}_K \leftarrow \operatorname*{arg\,min}_{H_K} \sum_{\tau=0}^{t} \|Q_K(x_\tau^{(i)}, u_\tau^{(i)}) - c_\tau^{(i)} - \gamma Q_K(x_{\tau+1}^{(i)}, Kx_{\tau+1}^{(i)})\|_2^2 + \eta \|H_K\|_{\mathsf{F}}^2$$

2. *Policy improvement:* At the end of episode i, improve the policy via the update

$$K \leftarrow \operatorname*{arg\,min}_{K} Q_K(x, Kx) = -H_{K,22}^{-1} H_{K,12}^{\mathsf{T}}.$$

Model-free policy gradient (2, §13.3): The third approach uses Monte Carlo policy gradients via the REINFORCE algorithm to train a Gaussian policy distribution $\pi_K(u \mid x) \coloneqq \mathcal{N}(Kx, \Sigma)$, where $K \in \mathbb{R}^{m \times n}$ is the gain parameter and $\Sigma \succ 0$ is fixed. This approach proceeds in two steps:

1. *Policy rollout:* During episode i, apply the stochastic policy $\pi_K(u \mid x)$ and observe the data $\{(x_t^{(i)}, u_t^{(i)}, c_t^{(i)}, x_{t+1}^{(i)})\}_{t=0}^{T-1}$.
2. *Policy gradient update:* At the end of episode i, compute the empirical mean and standard deviation of *all* of the costs observed so far, i.e.,

$$\mu^{(i)} := \frac{1}{iT} \sum_{j=1}^{i} \sum_{\tau=0}^{T-1} c_\tau^{(j)}, \quad \sigma^{(i)} := \frac{1}{iT} \sum_{j=1}^{i} \sum_{\tau=0}^{T-1} \left(c_\tau^{(j)} - \mu^{(i)} \right).$$

Then compute the policy update

$$K \leftarrow K - \alpha \sum_{t=0}^{T-1} \gamma^t v_t^{(i)} \nabla_K \pi_K(u_t^{(i)} \mid x_t^{(i)}).$$

where $\alpha > 0$ is the learning rate, and the observed costs in computing the tail returns

$$v_t^{(i)} := \sum_{\tau=t}^{T-1} \gamma^{\tau-t} \left(\frac{c_\tau^{(i)} - \mu^{(i)}}{\sigma^{(i)}} \right)$$

are standardized using $\mu^{(i)}$ and $\sigma^{(i)}$.

We have implemented all three algorithms and presented some results in Figure 1. It shows the difference $\|K^{(i)} - K^*\|_{\mathsf{F}}$ between the learned policy parameter $K^{(i)}$ and the optimal policy K^* at the end of each episode i. Figure 1 also displays the discounted cost $\sum_{t=0}^{T-1} \gamma^t c_t^{(i)}$ at the end of each episode i alongside the expected optimal cost-to-go $\mathbb{E}_{x_0}[V^*(x_0)]$ for the true dynamics and cost function. Figure 2 shows a zoomed-in version of these plots.

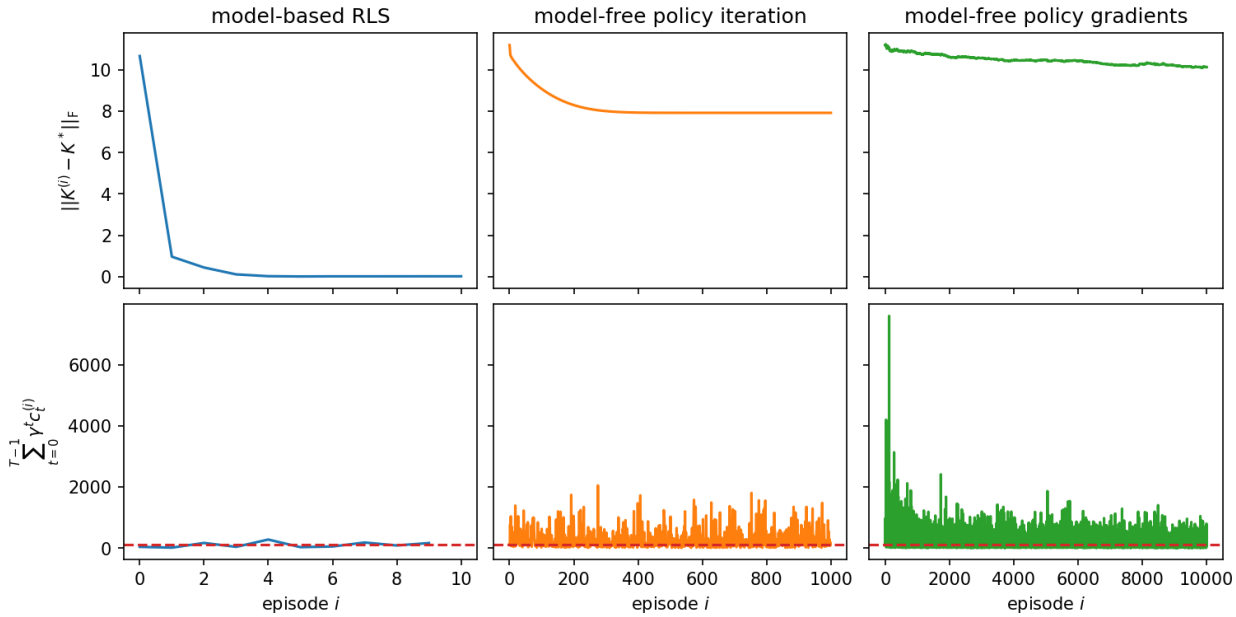

Figure 1: Gain errors and discounted cost sums for learning LQR.

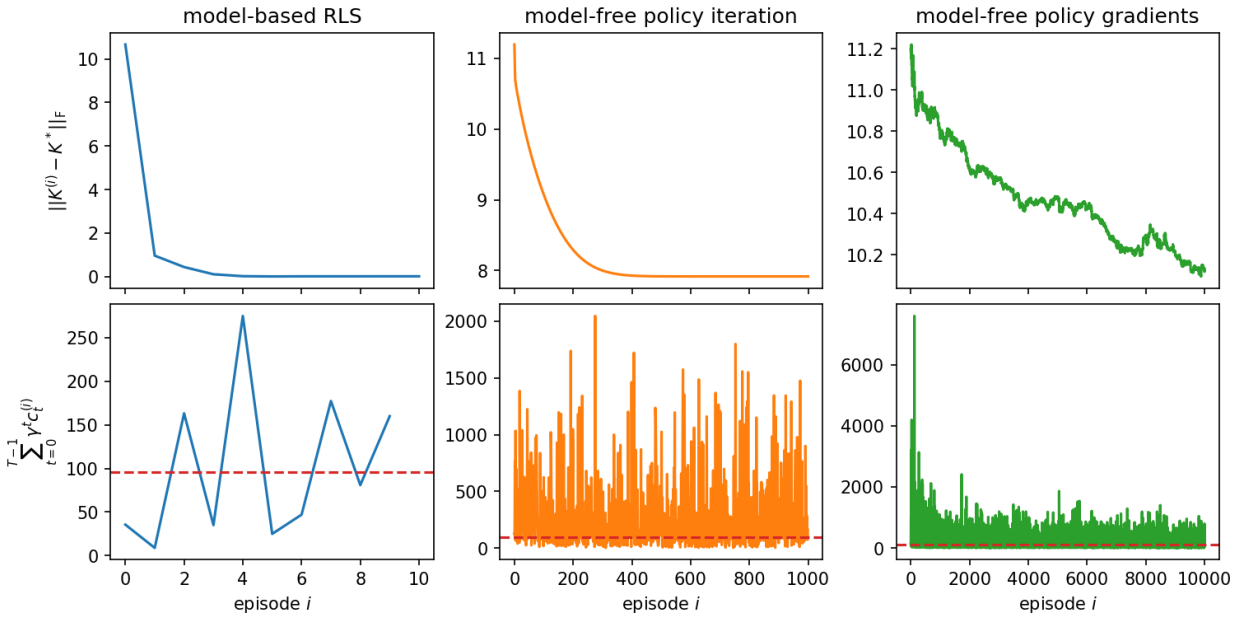

Figure 2: Gain errors and discounted cost sums for learning LQR (zoomed).

With these results in mind, your task is to answer the following conceptual questions:

(a) Each of the three described learning-based methods required access to some amount of problem-specific information. For each of these methods, describe the problem information that was used in the design of the algorithm. How does the amount of assumptions for each of the methods compare to the performance of the methods?

(b) Suppose we had available additional information/assumptions about the problem, e.g., prior belief distributions for A, B, Q, or R. To what degree could these beliefs be incorporated into each of these learning methodologies?

4.3 Lunar lander. As we have observed in the context of learning LQR, a naïve Monte Carlo policy gradient method (i.e., REINFORCE) can be quite unstable/slowly converging due to high estimator variance. As discussed in lecture, a popular variance reduction technique is to implement an actor-critic method, in which a value function is used as a baseline (referred to as the critic, which predicts the expected return associated with actions), with the policy referred to as the actor (which selects the actions). In particular, in this problem we will consider an advantage actor-critic algorithm which, with some details omitted, is constructed as follows:

- To improve the parameters θ of the actor π_θ, we combine the Q-function policy gradient with the value function baseline and obtain the following formulation for our policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\delta^\pi \nabla_\theta \log \pi_\theta(\mathbf{u}_t \mid \mathbf{x}_t)\right], \tag{1}$$

  with δ^π an estimate of the advantage $A^\pi(\mathbf{x}, \mathbf{u}) = Q^\pi(\mathbf{x}, \mathbf{u}) - V^\pi(\mathbf{x})$.
- To improve the parameters w of the critic V_w^π (which aims to estimate V^π) we take gradient descent steps to minimize $(\delta^\pi)^2$ (note that V_w^π is used in the computation of the advantage estimate δ^π).

As RL algorithm implementations can be quite time-consuming to run/debug, in this example[1] we

[1]View the notebook through Google Colab to see example output even without waiting to run the code.

have provided an already complete, albeit somewhat barebones[2], implementation of an advantage actor-critic algorithm. Your task in this problem is to review the code (a task you will undoubtedly become familiar with in your professional careers, even if it seems a bit unusual for a homework assignment) and answer the following conceptual questions:

(a) What estimator δ^π of the advantage is this code using? Recalling our discussion in lecture, where does this choice fall on the spectrum of the bias-variance tradeoff?

Note: In reviewing the code, you may observe the distinction between `standardized_returns` and `returns` (the former being what's passed into `train_step_for_episode` as "returns"). For the purposes of this question, you may disregard this transformation, i.e., consider the `standardized_returns = returns`.

(b) Now considering the note from part (a), why might we wish to standardize the returns before learning?

Hint: "Learning values across many orders of magnitude" by van Hasselt et al. (3) may provide some insight.

(c) `jax.lax.stop_gradient` is a function that acts like the identity function (i.e., leaves its input argument unchanged) but prevents the backpropagation of gradients through itself. That is, `stop_gradient` makes its input argument appear like a constant, as far as autodifferentiation is concerned. What is the significance of using this function in computing the `actor_loss` in `train_loss_for_episode`?

(d) Considering now the broader landscape of optimal and learning control, how else might you approach this lunar lander problem? In a short paragraph (maximum 4-5 sentences), sketch out an alternative approach for solving the lunar lander problem and discuss the strengths/weaknesses you imagine it having vs. the model-free RL approach here. Feel free to make whatever modeling assumptions you'd like (i.e., known vs. unknown dynamics, deterministic vs. stochastic dynamics, known vs. unknown reward, etc.).

[2]Many improvements over the method implemented are possible and can improve performance. A non-exhaustive list is given below, feel free to experiment with these (you can also try them out on more complicated environments such as `BipedalWalker-v3`):

- Experience replay: store transitions (x_t, u_t, r_t, x_{t+1}) in a buffer. Old examples are deleted as we store new transitions. To update the parameters of our network, we sample a (mini-)batch from the buffer and perform the stochastic gradient update on this batch.
- Try out different advantage estimators, e.g., the TD advantage estimate or n-step advantage estimate that bootstrap learning with the value function.
- Play with the structure of generalized policy iteration, i.e., the current implementation does one step each (synchronously) of policy improvement and policy evaluation; this ratio could be adjusted.
- Using decoupled policy or value networks (currently they are different heads on the same trunk network).
- Implement a more recent method (e.g., Proximal Policy Optimization (PPO), Soft Actor Critic (SAC), Deep Deterministic Policy Gradient (DDPG), and many more variants/alternatives) from the deep RL literature!

References

[1] Bradtke, S. J. and Ydstie, B. E. and Barto, A. G., *Adaptive linear quadratic control using policy iteration*. American Control Conference 1994.

[2] Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*. MIT Press 2018.

[3] van Hasselt, H. and Guez, A. and Hessel, M. and Mnih, V. and Silver, D., *Learning values across many orders of magnitude*. Conf. on Neural Information Processing Systems 2016.