Problem Set 6: Reinforcement Learning

6.4210/2 Robotic Manipulation — Fall 2026

Due: November 1, 2026

Out: October 22 **Due:** November 1

Reading: Lectures 10–11, Reinforcement Learning notes, not including model-based approaches

Before you start: update your Python environment The ps6 notebooks train neural networks and simulate cart-pole, which needs packages that earlier psets did not. Pull the latest handout and check the install instructions in its `README.md` (§3.3). If you made your venv before ps6, activate it and run

```
uv pip install torch gymnasium pandas ipykernel
```

(`torch` is already there if you added the ps5 packages.) On Colab, `setup()` installs them for you.

Enter the answers in the **Gradescope** blocks in Gradescope. The **Going further** prompts are for thought and discussion and require no submission. In the bookwork, unless stated otherwise, an episode consists of one action and one reward, so its return is just that reward. All logarithms are natural.

1 Bookwork

Problem 1 Representations for learning [8]

A machine has two hidden states, L and H, but its display always shows exactly the same observation o. Your actions are A or B. The reward and next hidden state are deterministic:

	A: (reward, next state)	B: (reward, next state)
L	$(0, H)$	$(1, L)$
H	$(4, H)$	$(5, L)$

The initial state is L and the discount is $\gamma = 1/2$. There is no termination. We will consider Q-learning with a table indexed only by the observation o and the action, without access to the hidden states or history. Recall that return is the discounted sum of future rewards.

Gradescope:

1.1 What is the discounted return starting in L if we always choose A? What if we always choose B? [2]
Always A: **4**; always B: **2**.

Solution: *A constant reward c from step k on is worth $\sum_{j \geq k} \gamma^j c = c\gamma^k/(1-\gamma)$. For always A: $L \xrightarrow{A} (0, H)$, then $H \xrightarrow{A} (4, H)$ forever, so the rewards are $0, 4, 4, \ldots$ and*

$$G_A = 0 + \frac{4\gamma}{1-\gamma} = \frac{4(1/2)}{1/2} = 4.$$

For always B: $L \xrightarrow{B} (1, L)$ forever, so the rewards are $1, 1, 1, \ldots$ and

$$G_B = \frac{1}{1-\gamma} = 2.$$

1.2 Suppose the hidden state is observed. What is the optimal policy mapping hidden state to action? [2]
$\pi^*(L) =$ **A**; $\pi^*(H) =$ **A**.

Solution: *Compare choosing A now with choosing B now and taking the same actions afterwards. Choosing B gives 1 more immediately, but leaves the machine in L instead of H. On the next step, either action pays 4 more in H than in L, worth $\gamma \cdot 4 = 2$ at the current step. After that action, the two trajectories reach the same hidden state, so all later rewards agree. Switching to B therefore changes the return by $1 - 2 = -1$. We should choose A in either state.*

1.3 After the first step, the display still shows the same observation o. Suppose the current action is A. What immediate reward follows in each case? [1]

The previous action was A: **4**; the previous action was B: **0**.

Solution: *The previous action determines the hidden state: A leaves the machine in H, and B leaves it in L. If we now choose A, we get*

$$\textit{previous } A \Rightarrow \textit{state } H \Rightarrow r(H, A) = 4,$$
$$\textit{previous } B \Rightarrow \textit{state } L \Rightarrow r(L, A) = 0.$$

The same observation and action give different rewards depending on history. The constant display sequence is trivially Markov, but the display is not a Markov state for predicting rewards and transitions*, which is what Q-learning requires.*

1.4 Now, imagine we collect a very long run by independently choosing A or B with probability $1/2$ at every step. Ignoring the initial step, the previous action is equally likely to be A or B. Since the current action is chosen independently of the previous action, samples of either current action come approximately half from L and half from H in a very long run. What is the average reward for each action? $\bar{r}_A =$ **2**; $\bar{r}_B =$ **3**. [1]

Solution: *For either current action, half the samples come from each hidden state. We can therefore average the two rewards:*

$$\bar{r}_A = \tfrac{1}{2}(0) + \tfrac{1}{2}(4) = 2,$$
$$\bar{r}_B = \tfrac{1}{2}(1) + \tfrac{1}{2}(5) = 3.$$

1.5 Q-learning stores two values, $q_A = Q[o, A]$ and $q_B = Q[o, B]$. For simplicity, assume it is learning from a replay buffer filled by the random run of part 1.4. Each entry in the buffer is one recorded step, written as a tuple (observation before, action, reward, observation after). The display always shows o and the hidden state is not recorded, so the buffer holds just four kinds of entry: $(o, A, 0, o)$, $(o, A, 4, o)$, $(o, B, 1, o)$ and $(o, B, 5, o)$. Assume these four kinds occur in exactly equal proportions. We sample entries uniformly from this fixed buffer. Assume the learning-rate schedule is chosen so that the Q-values converge. What are the resulting values of $q_A =$ **5** and $q_B =$ **6**? Which action does the greedy policy choose? **B**. [2]

Solution: *Each Q-learning update moves q_a toward $r + \frac{1}{2}\max(q_A, q_B)$. At convergence, each value equals its average target. Letting $m = \max(q_A, q_B)$, we get*

$$q_A = \bar{r}_A + \tfrac{1}{2}m = 2 + \tfrac{1}{2}m, \qquad q_B = \bar{r}_B + \tfrac{1}{2}m = 3 + \tfrac{1}{2}m.$$

Subtracting the two fixed-point equations tells us which action has the larger value:

$$q_B - q_A = \bar{r}_B - \bar{r}_A = 1 \quad \Rightarrow \quad m = q_B,$$
$$q_B = 3 + \tfrac{1}{2}q_B \quad \Rightarrow \quad q_B = 6,$$
$$q_A = 2 + \tfrac{1}{2}(6) = 5.$$

The greedy policy picks B, whose actual return is only 2 (part 1), instead of A's 4. The failure is not due to lack of exploration: all four state/action pairs occur. The averaged Bellman operator is a $1/2$ contraction, so repeated average backups converge to this wrong fixed point. Random replay with the usual decreasing step sizes converges to the same point.

Going further: Does the optimal policy from part 1.2 need to distinguish L from H? Why can Q-learning still fail when its table uses only o? Where did the benefit of choosing A disappear in the Q-learning target? What history would repair the representation?

Solution: *The moral of this story is that even if the observation is sufficient to represent the optimal policy, if it does not render the process* **Markov**, *then it may not support Q learning.*

The optimal policy chooses A in both states, so we do not need to distinguish L from H to act optimally. We do need that distinction to predict each action's return.

Choosing A improves the next reward by leaving the machine in H. But both actions have the same observed successor o, so Q-learning uses the same bootstrap term $\frac{1}{2}\max(q_A, q_B)$ *for both. The target loses the benefit of choosing A.*

Remembering whether the last action was A recovers the hidden state. The known initial state handles the first step.

Problem 2 All the rewards are positive [6]

Consider a single-state problem with two possible actions, L and R, with rewards $r(L) = 1$ and $r(R) = 3$. The policy is a probability distribution over the two actions, parameterized by two real-valued weights $z = (z_L, z_R)$, called *logits*. We convert these weights into action probabilities using the softmax function:

$$p_L = \frac{e^{z_L}}{e^{z_L} + e^{z_R}} \qquad p_R = 1 - p_L \ .$$

Start at $z = (0, 0)$. After taking an action a, the REINFORCE update is

$$g = r(a)\nabla_z \log p_a \qquad z \leftarrow z + \alpha g \ .$$

Here $\alpha > 0$ is the learning rate, and $g = (g_L, g_R)$ gives the update direction for the two weights, in the same order as z.

Gradescope:

2.1 Starting with the initial weights, compute the gradient of each action's log probability with respect to $z = (z_L, z_R)$. For example, $\log p_L = z_L - \log(e^{z_L} + e^{z_R})$. [2]

$$\nabla_z \log p_L = (\underline{\ \mathbf{1/2}\ }, \underline{\ \mathbf{-1/2}\ }) \qquad \nabla_z \log p_R = (\underline{\ \mathbf{-1/2}\ }, \underline{\ \mathbf{1/2}\ }) \ .$$

Solution: *Differentiating* $\log p_L = z_L - \log(e^{z_L} + e^{z_R})$, *we get* $\partial \log(e^{z_L} + e^{z_R})/\partial z_j = p_j$, *so*

$$\nabla_z \log p_L = (1 - p_L,\ -p_R),$$
$$\nabla_z \log p_R = (-p_L,\ 1 - p_R).$$

At $z = (0, 0)$, *we have* $p_L = p_R = 1/2$, *giving gradients* $(1/2, -1/2)$ *and* $(-1/2, 1/2)$.

2.2 Still using the initial weights, assume the learning rate α is 1. Compute the changes in the two weights, $g = r(a)\nabla_z \log p_a$, for each possible sampled action. [2]

Sampled action	g_L: change in z_L	g_R: change in z_R
L	**1/2**	**-1/2**
R	**-3/2**	**3/2**

For either sampled action, do both weights (the logits z_L and z_R) increase? **No**.

Solution: *We multiply each action's log-probability gradient by its reward:*

$$a = L: \quad g = 1 \cdot (1/2, -1/2) = (1/2, -1/2),$$
$$a = R: \quad g = 3 \cdot (-1/2, 1/2) = (-3/2, 3/2).$$

Each sampled update raises one logit and lowers the other. Positive reward does not make every component of the update positive.

2.3 At the initial weights $z = (0, 0)$, each action is sampled with probability $1/2$, so the expected update is the average of the two vectors from the previous part. Find $\mathbb{E}[g] = (\underline{\textbf{-1/2}}, \underline{\textbf{1/2}})$. [1]

Solution: *Since each action has probability $1/2$, we average the two updates:*

$$\mathbb{E}[g] = \tfrac{1}{2}\big[(1/2, -1/2) + (-3/2, 3/2)\big] = \tfrac{1}{2}(-1, 1) = (-1/2, 1/2).$$

The better action R gets the stronger push when sampled, so its expected logit update is positive; the worse action's is negative.

2.4 Add 100 to *both* rewards. What is the new expected gradient? $\mathbb{E}[g] = (\underline{\textbf{-1/2}}, \underline{\textbf{1/2}})$. [1]

Solution: *The rewards are now 101 and 103, so the sampled updates become*

$$\begin{aligned} a = L: \quad & g = 101 \cdot (1/2, -1/2) = (50.5, -50.5), \\ a = R: \quad & g = 103 \cdot (-1/2, 1/2) = (-51.5, 51.5), \\ & \mathbb{E}[g] = \tfrac{1}{2}(-1, 1) = (-1/2, 1/2). \end{aligned}$$

The mean is unchanged, although the variability grows dramatically.

Going further: If all rewards are positive, why doesn't REINFORCE just increase all the weights? Explain why shifting all rewards by a constant leaves the *expected* update unchanged in this one-step problem. Does it leave each sampled update unchanged?

Solution: *With softmax logits, a positive reward raises the sampled action's logit and lowers the other's. The probabilities must still sum to one.*

To see why a constant reward shift leaves the expected update unchanged, we use the zero-mean score identity:

$$\mathbb{E}_a[\nabla_z \log p_a] = \sum_a p_a \nabla_z \log p_a = \nabla_z \sum_a p_a = 0.$$

Adding c to every reward adds $c\nabla_z \log p_a$ to each sampled update. Individual updates change, but the added term has expectation zero.

More generally, $\partial \log p_a / \partial z_j = \mathbf{1}\{a = j\} - p_j$, so

$$\mathbb{E}[g_j] = p_j r_j - p_j \sum_k p_k r_k = p_j \Big(r_j - \sum_k p_k r_k \Big).$$

The expected update depends on each reward relative to the mean reward. Network weights need not be logits, but the zero-mean score identity still holds.

Problem 3 A Gaussian policy on a line [10]

Consider a single-state problem with one real-valued action $a \sim \mathcal{N}(\mu, \sigma^2)$ and reward $r(a) = -(a-1)^2$. The policy parameters are the mean μ and standard deviation $\sigma > 0$; both are learned. Its log density is

$$\log \pi_{\mu,\sigma}(a) = -\log \sigma - \frac{(a-\mu)^2}{2\sigma^2} - \tfrac{1}{2}\log(2\pi) \ .$$

For the parts below that do not specify a baseline, use REINFORCE without a baseline:

$$g = (g_\mu, g_\sigma) = r(a)\nabla_{(\mu,\sigma)} \log \pi_{\mu,\sigma}(a), \qquad (\mu, \sigma) \leftarrow (\mu, \sigma) + \alpha g \ .$$

Here $\alpha > 0$ is the learning rate, chosen small enough to keep $\sigma > 0$. Each numerical example starts independently at $(\mu, \sigma) = (0, 1)$.

Gradescope:

3.1 Holding the sampled action a fixed, differentiate the log density with respect to each policy parameter. Give expressions valid for any μ and $\sigma > 0$: [2]

$$\frac{\partial \log \pi_{\mu,\sigma}(a)}{\partial \mu} = \underline{\qquad \frac{a-\mu}{\sigma^2} \qquad}, \qquad \frac{\partial \log \pi_{\mu,\sigma}(a)}{\partial \sigma} = \underline{\qquad \frac{(a-\mu)^2 - \sigma^2}{\sigma^3} \qquad}.$$

Solution: *Holding a fixed, we differentiate the log density with respect to each parameter:*

$$\frac{\partial \log \pi}{\partial \mu} = \frac{a-\mu}{\sigma^2},$$
$$\frac{\partial \log \pi}{\partial \sigma} = -\frac{1}{\sigma} + \frac{(a-\mu)^2}{\sigma^3} = \frac{(a-\mu)^2 - \sigma^2}{\sigma^3}.$$

3.2 At $(\mu, \sigma) = (0, 1)$, calculate both components of g for each sampled action listed below. [2]

Sampled action	g_μ	g_σ
-1	**4**	**0**
0	**0**	**1**
2	**-2**	**-3**

Solution: *At $(\mu, \sigma) = (0, 1)$, the log-density gradient is $(a,\ a^2 - 1)$. Multiplying by $r(a) = -(a-1)^2$ gives*

$$\begin{aligned} a = -1: \quad & r = -4, && g = -4 \cdot (-1, 0) = (4, 0), \\ a = 0: \quad & r = -1, && g = -1 \cdot (0, -1) = (0, 1), \\ a = 2: \quad & r = -1, && g = -1 \cdot (2, 3) = (-2, -3). \end{aligned}$$

3.3 Starting at $(\mu, \sigma) = (0, 1)$, consider a small positive learning rate. Use your answers from the previous part to answer the following questions. Select one action for each question. [1]

Which sampled action produces an update that moves the mean *farther from* the reward-maximizing action $a = 1$?

A. $a = -1$.

B. $a = 0$.

C. $a = 2$.

Which sampled action produces an update that *increases* the standard deviation?

A. $a = -1$.

B. $a = 0$.

C. $a = 2$.

Solution: *For $a = 2$, the mean becomes $\mu = -2\alpha < 0$, moving farther from 1. For $a = -1$, it becomes 4α, which moves toward 1 for small α; for $a = 0$, it stays at zero.*

Only $a = 0$ has $g_\sigma > 0$, so only that sample increases σ.

3.4 Suppose the sampled action is 2 and $\alpha = 1/4$. After one update, what are the new parameters? [1]

$\mu =$ ____−1/2____; $\sigma =$ ____1/4____.

Solution: *For $a = 2$, we found $g = (-2, -3)$. With $\alpha = 1/4$, the update gives*

$$\mu = 0 + \tfrac{1}{4}(-2) = -\tfrac{1}{2},$$
$$\sigma = 1 + \tfrac{1}{4}(-3) = \tfrac{1}{4}.$$

The result describes one possible sample, not the expected update.

3.5 For checking the expected update, write $a = \mu + \sigma\xi$ with $\mathbb{E}[\xi] = 0$ and $\mathbb{E}[\xi^2] = 1$. Work out the expected reward: [2]

$$J(\mu, \sigma) = \mathbb{E}[r(a)] = \text{_____}-(\mu - 1)^2 - \sigma^2\text{_____}.$$

At the initial parameters, give $J(0, 1) =$ ____−2____ and the expected update direction, ordered (μ, σ):

$$\nabla_{(\mu,\sigma)} J(0, 1) = (\text{____}\mathbf{2}\text{____}, \text{____}-2\text{____}).$$

Solution: *We substitute $a = \mu + \sigma\xi$ into the reward, then take expectations. The linear term in ξ vanishes because $\mathbb{E}[\xi] = 0$, and $\mathbb{E}[\xi^2] = 1$:*

$$r(a) = -\left[(\mu - 1)^2 + 2(\mu - 1)\sigma\xi + \sigma^2\xi^2\right],$$
$$J(\mu, \sigma) = -(\mu - 1)^2 - \sigma^2,$$
$$J(0, 1) = -1 - 1 = -2,$$
$$\nabla_{(\mu,\sigma)} J = \big(2(1 - \mu),\, -2\sigma\big) = (2, -2) \text{ at } (0, 1).$$

In expectation, the mean moves toward the best action and the spread shrinks.

3.6 Now measure whether the reward was *better than expected*. Use the fixed baseline $b = J(0,1) = -2$ and replace $r(a)$ in the update by the advantage $r(a) - b$. The two actions $a = 1/2$ and $a = 3/2$ both earn reward $-1/4$, so their advantage is ____7/4____. One action is near the current mean and the other is farther away. For each, give $g_\sigma = (r(a) - b)\,\partial \log \pi / \partial \sigma$ and the effect of a small positive update on the variance σ^2. Enter *increase*, *decrease*, or *unchanged*. [2]

Sampled action	g_σ	Effect on σ^2
1/2	$-21/16$	**decrease**
3/2	$35/16$	**increase**

For any positive advantage, the boundary between these two effects is $|a - \mu| =$ ____σ____.

Solution: *At $(0,1)$, we have $\partial \log \pi / \partial \sigma = a^2 - 1$. Using the baseline $b = -2$, we get*

$$r(1/2) = r(3/2) = -\tfrac{1}{4}, \qquad r - b = -\tfrac{1}{4} + 2 = \tfrac{7}{4},$$
$$a = \tfrac{1}{2}: \quad g_\sigma = \tfrac{7}{4}\left(\tfrac{1}{4} - 1\right) = -\tfrac{21}{16} \ \ (\textit{decrease}),$$
$$a = \tfrac{3}{2}: \quad g_\sigma = \tfrac{7}{4}\left(\tfrac{9}{4} - 1\right) = \tfrac{35}{16} \ \ (\textit{increase}).$$

With a positive advantage, the sign of g_σ is the sign of $(a-\mu)^2 - \sigma^2$, which changes at $|a - \mu| = \sigma$. Since $\sigma > 0$, σ^2 moves in the same direction as σ. "Good" here means better than the baseline, even though the reward is negative.

Going further: Why does maximizing expected reward in this problem drive σ toward zero, even though a single better-than-expected action far from the mean can increase it?

Solution: *We found $J(\mu, \sigma) = -(\mu - 1)^2 - \sigma^2$, so $\partial J / \partial \sigma = -2\sigma < 0$. The expected gradient therefore shrinks the spread. The best expected reward is approached as $\mu = 1$ and $\sigma \to 0$.*

An individual update can still widen the distribution when a distant action performs better than expected. That update makes the promising action more likely, even though averaging over all samples gives a decrease in σ.

Problem 4 Why PPO stops rewarding a big change [6]

Consider a single-state problem in which each episode consists of one action and one reward. The actions are L and R. Unlike Problem 2, we parameterize the policy directly by the probability p: $p_R = p$ and $p_L = 1 - p$, with $0 \le p \le 1$. For this exercise we optimize the scalar probability p.

The old policy chooses each action with probability $1/2$. Suppose we collect two episodes and happen to sample each action once. This batch has fixed advantage estimates $\widehat{\text{adv}}_R = 100$ and $\widehat{\text{adv}}_L = -100$. Keep these estimates and the old probabilities fixed as we reuse the batch. The likelihood ratios compare each action's new probability to its old probability:

$$\rho_R = \frac{p}{1/2} = 2p \qquad \rho_L = \frac{1-p}{1/2} = 2(1-p) \ .$$

The unclipped objective averages the ratio times the estimated advantage over the two samples:

$$\mathcal{L}(p) = \tfrac{1}{2}[100\rho_R - 100\rho_L] \ .$$

PPO with $\epsilon = 0.2$ instead uses

$$\mathcal{L}_{\text{clip}}(p) = \tfrac{1}{2}\left[\min(100\rho_R, 100\,\text{clip}(\rho_R, 0.8, 1.2)) + \min(-100\rho_L, -100\,\text{clip}(\rho_L, 0.8, 1.2))\right] \ .$$

Here $\text{clip}(x, l, u)$ limits x to the interval $[l, u]$.

Gradescope:

4.1 Simplify $\mathcal{L}(p)$ and give its derivative: $\mathcal{L}'(p) =$ **200**. Where is its maximum on $[0, 1]$? $p =$ **1**. [1]

Solution: *Substituting the two likelihood ratios and simplifying gives*

$$\mathcal{L}(p) = \tfrac{1}{2}\big[200p - 200(1-p)\big] = 200p - 100,$$
$$\mathcal{L}'(p) = 200 > 0 \quad \Rightarrow \quad \text{maximum at } p = 1.$$

Repeatedly optimizing this batch drives the policy to always choose R, even though the data came from a policy that chose both.

4.2 Starting from $p = 1/2$, take one ascent step $p \leftarrow p + \alpha\mathcal{L}'(p)$ with learning rate $\alpha = 0.0025$. Give the new $p =$ **1** and ratio $\rho_R =$ **2**. [1]

Solution: *Using $\mathcal{L}' = 200$, we get*

$$p' = 0.5 + 0.0025(200) = 1,$$
$$\rho_R = 2p' = 2.$$

One step eliminates action L entirely.

4.3 For $p \geq 1/2$, at what p does the clipped objective first become flat? **0.6**. Evaluate $\mathcal{L}_{\text{clip}}(0.6) =$ **20** and $\mathcal{L}_{\text{clip}}(1) =$ **20**. [2]

Solution: *For a positive advantage, clipping caps the term at $100(1.2)$. For a negative advantage with $\rho \leq 1$, the term is $-100\max(\rho, 0.8)$. For $p \geq 1/2$, $\rho_R = 2p \geq 1$ and $\rho_L = 2(1-p) \leq 1$, so*

$$\mathcal{L}_{\text{clip}}(p) = \tfrac{1}{2}\big[100\min(2p, 1.2) - 100\max(2(1-p), 0.8)\big].$$

Both terms stop changing exactly when $2p \geq 1.2$ and $2(1-p) \leq 0.8$, that is, at $p \geq 0.6$.

$$\tfrac{1}{2} \leq p \leq 0.6: \quad \mathcal{L}_{\text{clip}}(p) = 200p - 100,$$
$$p \geq 0.6: \quad \mathcal{L}_{\text{clip}}(p) = \tfrac{1}{2}(120 - 80) = 20.$$

Every $p \in [0.6, 1]$ is a maximizer; clipping does not prefer 0.6 to larger values.

4.4 Now use the smaller step size $\alpha = 0.0002$. Start again from $p = 1/2$ and take three ascent steps using the *clipped* objective. Give the third iterate $p =$ **0.62** and the gradient there **0**. [1]

Solution: *The clipped objective has derivative 200 below $p = 0.6$ and 0 above it. Each step below the plateau adds $0.0002(200) = 0.04$, so*

$$p_1 = 0.50 + 0.04 = 0.54,$$
$$p_2 = 0.54 + 0.04 = 0.58,$$
$$p_3 = 0.58 + 0.04 = 0.62 \quad (\text{on the plateau, gradient } 0).$$

There won't be any further updates due to this batch. Without clipping, the gradient would still be 200 at $p = 0.62$, so the next update would increase p to 0.66.

4.5 Would using the clipped objective have prevented the single large step in part 4.2? **No**. [1]

Solution: *At $p = 1/2$, both ratios equal 1 and lie inside $[0.8, 1.2]$. The clipped and unclipped objectives agree nearby, so $\mathcal{L}'_{\text{clip}}(1/2) = 200$. We therefore take the same step, $0.0025(200) = 0.5$, and land at $p = 1$, past the start of the plateau. Clipping does not impose a hard constraint on probability ratios; the step size still matters.*

Going further: What if those advantages came from a small, lucky batch? Why might it be useful to collect new data before the policy becomes almost deterministic? Does PPO clipping guarantee that the true expected return improves?

Solution: *A fresh batch lets us evaluate the new policy and replace noisy advantage estimates before the policy becomes nearly deterministic. Clipping limits the old batch's incentive to make a large change, but cannot correct inaccurate estimates. It therefore does not guarantee an improvement in true expected return.*

Problem 5 Does the policy need to be random? [8]

A *deterministic* policy always chooses the same action for the same input; a *stochastic* policy draws an action from a conditional probability distribution.

Gradescope:

5.1 Assume a robot that observes the full Markov state. There are finitely many states and actions, bounded rewards, and discount $0 < \gamma < 1$. The reward and transition rules do not change over time. Transitions may be random. The objective is expected discounted return. Among all policies that can use the full state, which statement is correct? [1]

A. A deterministic optimal policy exists if we use Q-learning, but not if we use policy gradient.

B. A deterministic optimal policy exists only if transitions are deterministic.

C. A deterministic optimal policy exists, regardless of which learning algorithm we choose.

D. Every optimal policy must be deterministic.

Solution: *By Bellman optimality, we can choose one action in* $\arg\max_a Q^*(s,a)$ *for each state. Breaking ties consistently gives a deterministic optimal policy, even when transitions are random. The choice of learning algorithm does not change which policies are optimal.*

Ties can also allow stochastic optimal policies, so not every optimal policy must be deterministic. Randomness may still be useful during training for exploration.

5.2 Now a robot must issue two commands in a row to open a latch. The only commands are A and B, and the episode ends after the second command. Only the sequence A then B succeeds and earns reward 1; every other sequence earns 0. There is no other reward or discounting. The observation is identical at both steps. We try restricting the policy to a fixed mapping from *only this observation* to action probabilities. Which policy has the highest success probability in this class? [2]

A. Always choose A.

B. Choose A or B with equal probability at each step.

C. Always choose B.

D. All policies in this class have the same success probability.

Solution: *Let* $p = P(A)$ *at each step. The two draws are independent, so*

$$P(\textit{success}) = P(A \textit{ first})\, P(B \textit{ second}) = p(1-p),$$

$$\frac{d}{dp}\, p(1-p) = 1 - 2p = 0 \quad \Rightarrow \quad p = \tfrac{1}{2}, \quad P = \tfrac{1}{4}.$$

Both deterministic policies ($p = 0$ *or* 1) *succeed with probability* 0*. The observation hides the task's progress; it is not a Markov state. Every optimal policy in this class is stochastic, although the physical task is deterministic.*

5.3 For the same latch, now we allow the policy to remember whether it has already issued the first command. Which statement is correct? [1]

A. Randomness is still necessary because the observations have not changed.

B. No policy can achieve success probability greater than 1/4.

C. A deterministic policy can succeed with probability 1: choose A first and B second.

Solution: *The policy "A if no command yet, otherwise B" is deterministic and always produces* A, B*, so it succeeds with probability* 1*. The need for randomness in part 5.2 came from restricting the policy's input, not from a general rule that partial observability requires stochastic policies.*

5.4 Return to the observation-only latch policy in part 5.2. Compare ordinary Q-learning with a table $Q(o, a)$ to REINFORCE using complete two-command attempts collected using its current policy. Which statement is correct? [1]

A. Q-learning's usual Markov-state guarantees apply because the physical task is deterministic.

B. REINFORCE requires Markov observations, so it cannot estimate a policy gradient here.

C. Switching from Q-learning to policy gradient changes whether a deterministic optimal observation-only policy exists.

D. REINFORCE can estimate the gradient of return for this policy class; ordinary Q-learning on o lacks its usual Markov-state guarantees.

Solution: *REINFORCE uses each sampled trajectory's return and action log probabilities. It does not bootstrap from a value of o, so it can estimate the gradient for this policy class.*

Q-learning treats o as if it summarized the relevant past. Since the observation omits the task's progress, the usual Markov-state guarantees do not apply.

Neither algorithm changes the returns achievable by the policy class, and a valid gradient estimate does not guarantee that optimization finds the best policy.

5.5 Consider behavior cloning (BC) with three ordered steering commands, $-1, 0, +1$. For one fixed observation, the demonstration frequencies are $1/4, 1/2, 1/4$. This distribution is *unimodal*: it has one peak, at 0. Allow the policy to assign any probabilities to the three commands. Fit these probabilities by maximum likelihood, meaning that we maximize the average log probability assigned to the demonstrated commands. Which fitted policy is optimal for this BC objective? [1]

A. Always output 0, since there is only one mode.

B. Output $-1, 0, +1$ with probabilities $1/4, 1/2, 1/4$.

C. Output -1 or $+1$ with equal probability, never 0.

D. The answer depends on whether the environment's transitions are Markov.

Solution: *Writing the demonstration frequencies as $q = (1/4, 1/2, 1/4)$, we maximize $\sum_k q_k \log \pi_k$ subject to $\sum_k \pi_k = 1$. By Gibbs' inequality, the maximum occurs at $\pi = q$.*

A unimodal distribution need not be deterministic: commands other than the mode still occur. Assigning probability zero to any demonstrated command would give log likelihood $-\infty$. A deterministic distribution would maximize likelihood if every demonstration used the same command. No task reward is specified here, so matching the demonstrations does not make a claim about return.

5.6 In a one-step, fully observed task, a robot selects a route around an obstacle. Command -1 goes left and $+1$ goes right; both succeed and earn reward 1. Command 0 goes straight, hits the obstacle, and earns 0. These are the only commands. At the same input, half the demonstrations use -1 and half use $+1$: the data are *multimodal*, with two peaks. Compare three objectives: minimize squared error to demonstrated commands; maximize demonstration log likelihood; maximize task reward. For squared-error BC, fit a deterministic command; for maximum-likelihood BC, fit a probability distribution over the three commands. For task reward, allow either deterministic or stochastic policies. Which statement is correct? [2]

A. All three objectives require a stochastic policy because the data have two modes.

B. All three objectives are optimized by always choosing either -1 or $+1$.

C. Squared error and likelihood both prefer command 0; task reward prefers randomness.

D. Squared error prefers always choosing 0; likelihood prefers an equal mixture of -1 and $+1$; task reward admits a deterministic optimum.

Solution: *The three objectives give different answers:*

- *Squared error is minimized by the mean command, $c = \mathbb{E}[a_{\text{demo}}] = 0$, which hits the obstacle. Even a randomized prediction a, independent of the demonstration, has $\mathbb{E}[(a_{\text{demo}} - a)^2] = 1 + \mathbb{E}[a^2]$, minimized by always choosing 0.*
- *Maximum likelihood matches the demonstration frequencies: $\pi = (1/2, 0, 1/2)$.*
- *Expected task reward is $\pi_{-1} + \pi_{+1} \leq 1$. Always choosing -1 or always choosing $+1$ attains reward 1, as does any mixture of the two.*

Multimodal demonstrations do not imply that randomness is needed to maximize reward, and averaging good actions need not give a good action.

2 Computational questions

This homework requires experimenting with RL methods. We have tried to scope it so that it doesn't require more than an hour of compute on a CPU laptop or Colab. Cart-pole has short local training runs. For pushing, we supply full training results and a real on-policy batch for one short coding update, so the required experiment does not repeat hours of simulator interaction.

Python files and notebooks Each problem uses two files:

- **A Python file** (`reinforce.py` or `push_math.py`). Write your code here.
- **A Jupyter notebook** (`cartpole.ipynb` or `ppo_push.ipynb`). It imports your functions and runs the checks, training, plots and replays. Your solutions do not go here; at most you change a setting the notebook points out, such as `CONDITION`.

Running the notebooks

- In VS Code, open the notebook and select the course Python environment as its kernel. Setup is in the course `README.md` (§3.3 locally, §4 on Colab).
- In the Jupyter notebook, run cells individually, in order. **Do not use "Run All" without reading the cells first**: training, fitting, and Meshcat replay cells run immediately.
- Finish the functions in the Python file and run the notebook's checks on them before starting training.
- The setup cell turns on IPython's autoreload, so the notebook picks up edits to the Python file the next time a cell calls the edited function; there is no need to restart the kernel. If a change still does not seem to take effect, restart the kernel and rerun the setup cell.

Saved results and data

- Repeating a training or evaluation cell reuses saved results only when its code, settings, data, and output files match. To repeat an experiment deliberately, add `retrain=True` to its call; earlier outputs are archived.
- Replays perform a fresh simulation each time.
- Supplied data and reference results are in `ps6/data`; your runs, plots, and replay HTML files are saved in `ps6/results`.
- Problem 7 uses full PPO training runs of 8,192 pushes each, which take about three hours apiece. Their results are supplied in `ps6/data`; you will analyze them but do not need to rerun them.

Writing the losses (in the Python files)

- The code uses PyTorch for neural-network training. The loss functions operate on arrays supporting familiar arithmetic, indexing, `.sum()`, and `.mean()`. These arrays let the training code that calls your loss (`train` in `reinforce.py`, `clone_cautious` and `update_from_batch` in `residual_push.py`) compute derivatives automatically; your code only expresses the mathematical loss.
- Keep their array type when computing losses, rather than converting to NumPy or a Python float. Where NumPy uses `np.exp`, `np.minimum`, or `np.clip`, use `torch.exp`, `torch.minimum`, or `torch.clamp`.
- `reward_to_go` and `gae_advantages` use ordinary NumPy arrays.
- The function documentation explains the few library calls needed.

Problem 6 Cart-pole [6]

Notebook: `ps6/cartpole.ipynb`. Complete two functions in `ps6/reinforce.py`, `reward_to_go` and `policy_loss` (part 6.1); use the notebook for local runs, reference learning curves, statistical analysis, and replay. Enter the answers in each **Gradescope** block in Gradescope. Upload the learning-curve plot requested in part 6.3; keep your other plots and replay observations in the notebook. Your two locally trained policies will have a non-deterministic performance score. Parts marked *by hand* are logic checks: work them out from the definitions. The notebook's check cell tests your code on different values, so it does not give these answers. Parts marked *from the notebook* are read from its printed output.

A pole is attached by a free hinge to a cart that moves along a horizontal track. The controller applies a force to the cart, pushing it left or right to keep the pole balanced upright without leaving the track.

Our Drake cart-pole observes $(x, \theta, \dot{x}, \dot{\theta})$, with $\theta = 0$ upright, and chooses a force of ± 100 N every 0.02 seconds. Each transition yields a reward of 1. An episode ends at $|x| > 2.4$ m, $|\theta| > 12°$, or 500 steps. We maximize undiscounted return ($\gamma = 1$) over these episodes. The supplied actor uses only the physical state; it does not observe how many steps remain.

6.1 **Implementing the policy-gradient update** Start by reading the code in `ps6/reinforce.py`, which has a standalone implementation of the REINFORCE algorithm. Its two missing functions, `reward_to_go` and `policy_loss`, compute the quantities defined below. Review the definition of *reward-to-go* and what it's used for in REINFORCE. [3]

Our objective, after running our current policy for B complete episodes, is to take a gradient step to minimize

$$-\frac{1}{B}\sum_{i=1}^{B}\sum_{t=0}^{T_i-1}\log \pi_\theta(a_{it} \mid s_{it})[G_{it} - b(s_{it}, t)].$$

The factor $1/B$ averages over episodes. `policy_loss` computes this loss. Here s_{it} and a_{it} are the state and sampled action at time t in episode i. `train` concatenates the B episodes into one batch, so `policy_loss` receives arrays of length $N = T_1 + \cdots + T_B$, the total number of transitions in the batch. The loss sums over all N entries but divides by B, not N. `train` also computes $\log \pi_\theta$ for each recorded action and holds the recorded returns and baseline predictions fixed when differentiating this loss with respect to the policy weights. Without a baseline, use $b = 0$.

Gradescope: *By hand.* For one episode with rewards $(1, 1, 1)$, give the reward-to-go vector:

$$(\underline{\quad \mathbf{3} \quad}, \underline{\quad \mathbf{2} \quad}, \underline{\quad \mathbf{1} \quad}).$$

As a separate check of `policy_loss`, consider two episodes of lengths 2 and 1. The concatenated inputs are `action_log_probs`$= (-0.2, -1.0, -0.4)$, `returns`$= (2, 1, 1)$, and `baseline`$= (1, 2, 0)$. The number of episodes is $B = 2$. What scalar loss should the function return? $\underline{\quad -0.2 \quad}$.

With a positive advantage $G_{it} - b(s_{it}, t)$, that sample's contribution encourages the probability of its recorded action to *increase* or *decrease*? $\underline{\quad \textbf{increase} \quad}$.

Solution: *We sum the rewards from each step onward to get the reward-to-go vector. For the separate loss calculation, we subtract the baseline from each return, multiply by the recorded log probability, and average over the $B = 2$ episodes:*

$$\begin{aligned} G &= (1+1+1,\ 1+1,\ 1) = (3, 2, 1), \\ G - b &= (2-1,\ 1-2,\ 1-0) = (1, -1, 1), \\ \textstyle\sum \log \pi \cdot (G - b) &= (-0.2)(1) + (-1.0)(-1) + (-0.4)(1) = 0.4, \\ \textit{loss} &= -0.4/2 = -0.2. \end{aligned}$$

Divide by the two episodes, not the three transitions. Minimizing $-\log \pi \cdot (G - b)$ with $G - b > 0$ raises $\log \pi$ of the recorded action.

With the hand checks done, complete `reward_to_go` and `policy_loss` in `reinforce.py`, then run the notebook's check cell.

Watch a random policy Run the Drake/Meshcat replay cell in the notebook with `CONDITION` set to `"random"`. This policy chooses each force with probability 1/2 at every step. Watch the replay and describe how it fails: does the pole exceed its allowed angle, or does the cart leave the track? The red markers show the cart-position limits. Check your observation against the printed ending reason, and record a brief description in the notebook. If the episode reaches the time limit instead, report that outcome.

Now, let's train a policy! We'll do it once without a baseline and once with it, using the notebook's fixed training schedule. Use random seed 100 for both runs. A seed initializes the random-number generators for network initialization and sampling. These two runs form one *training pair*: they start with identical policy-network weights, and we evaluate them on the same starting states. Their learned policies and training trajectories can diverge; pairing can reduce variability in the comparison.

The notebook plots mean sampled evaluation return against training rounds and compares the final policies on 20 other starting states, shared by both policies. Inspect a Drake replay of each policy as well. One training pair lets you see the process, but is generally not enough to establish which method performs better across training runs. The following part uses many supplied pairs to assess that evidence.

6.2 **Comparing methods across training runs** Use the supplied study in `ps6/data/cartpole`; do not retrain these policies. It repeats the preceding comparison with 30 training seeds, giving 30 pairs of trained policies. Each final policy is evaluated by sampling actions in 100 episodes. These 100 measurements describe one trained policy; they are not 100 independent training runs. [2]

For pair i, average the 100 returns separately for each policy and form

$$D_i = \bar{R}_{i,\text{baseline}} - \bar{R}_{i,\text{plain}}.$$

Use the final planned checkpoint (round 60), not a checkpoint selected because it favors one method. Compute the mean of the 30 differences, a paired bootstrap 95% interval, and a two-sided paired t-test of $H_0 : \mathbb{E}[D_i] = 0$, using the paired t-test described in §5.2 ("Evaluating a policy," under "Paired trials") of the *Learning from demonstrations* notes. The notebook's helpers use 5000 bootstrap resamples and random seed 813 for this interval. Appendix A explains how the bootstrap interval is computed.

Gradescope: *By hand.* For this analysis, how many independent paired differences are there? $n =$ **30**. What are the t-test's degrees of freedom? **29**.

From the notebook (`compare_runs`), give the following values for the supplied data. Round returns and interval endpoints to two decimal places, t to three decimal places, and p to three significant figures (scientific notation is allowed).

Mean difference $\bar{D}$ (environment steps)	**39.10**
Paired bootstrap interval, lower endpoint	**23.19**
Paired bootstrap interval, upper endpoint	**58.34**
Paired t statistic	**4.290**
Two-sided t-test p-value	**0.000181**

By hand. What should one bootstrap sampling unit contain: (A) one evaluation episode, (B) both trained policies' results for one training seed, or (C) one environment transition? Enter A, B, or C: **B**.

Solution: *The independent units are the 30 training pairs, so $n = 30$ and there are 29 degrees of freedom. The 100 evaluation episodes per policy estimate that policy's return; they are not independent training runs.*

We apply the one-sample t-test to the paired differences $D_i = \bar{R}_{i,\text{baseline}} - \bar{R}_{i,\text{plain}}$ at round 60, using $t = \bar{D}/(s_D/\sqrt{n})$. The supplied data give $\bar{D} = 39.10$, $t = 4.2896008$, $p = 0.0001813209$, and the bootstrap interval $[23.189625, 58.342483]$. For the bootstrap, we resample whole pairs, keeping both policies' results for each selected seed together.

6.3 **Learning curves across training runs** Using the same supplied study, plot the two mean learning curves with pointwise bootstrap bands over training runs, retaining whole curves when resampling. Each band describes uncertainty in the mean at that checkpoint; it does not give simultaneous coverage of the entire learning curve. Upload this plot to Gradescope. The notebook shows it and also saves it as `results/cartpole_reference/learning_uncertainty.png`. [1]

Solution: *We resample whole training pairs, keeping every checkpoint of each sampled run. At each checkpoint, we take the mean over the resampled runs. The distribution of those means over bootstrap resamples gives that checkpoint's band.*

The baseline curve is above the plain curve by the end of training. The bands are pointwise; they do not give simultaneous coverage of the entire curve.

Things to think about:

- What would happen if we didn't supply $\dot{x}$ and $\dot{\theta}$ as input?
- Why is it okay, in this problem, to set $\gamma = 1$?
- At the final checkpoint, is using a baseline statistically significantly better, worse, or neither, than not using one? Use a significance level of 0.05 and explain how your paired t-test results support your answer.
- The baseline method has a higher mean return late in training. Does that mean every run with a baseline outperforms every run without one? What do the thin lines in the plot from part 6.3 tell us that the mean curves alone do not?
- What does the paired bootstrap 95% confidence interval tell us about the size and direction of the baseline's effect? What would it mean if the interval included zero? Does the interval describe the spread of individual episode returns or uncertainty in a mean difference?
- What assumptions does the paired t-test make about independence and the distribution of the data? Do those assumptions concern the individual episode returns or the paired differences D_i? Are they reasonable here? Consider the number of independent pairs and the distribution of their differences when explaining your answer.

Problem 7 Improving a cautious policy with residual PPO [23]

Notebook: `ps6/ppo_push.ipynb`.

A handwritten controller pushes a block to a target using cautious, short pushes. We use behavior cloning (BC) to imitate it, then PPO to learn a correction to its push distances while keeping the BC network fixed. Can learning from rewards improve on accurate imitation?

You will complete five functions in `ps6/push_math.py`, fit a BC policy, and perform one PPO update using supplied episodes. Then you will analyze results from longer training runs that we have already completed. The supplied code handles the networks, training loops, and data analysis. The replays and longer runs all use the same supplied BC policy; your short BC fit is a separate coding exercise.

Watch the policies Set `RUN_DRAKE_REPLAYS=True` to watch the supplied BC policy before implementing the missing functions. Near the end of the notebook, compare that recording with a policy improved by PPO, starting from the same state and goal. Compare the number of pushes and time to reach the goal, using the same playback speed. The PPO policy was chosen for its performance on a separate validation set; the statistical comparison uses all ten final policies. These replays require Drake; the coding exercises do not.

The pushing task Each command chooses one of the block's side faces and a stick travel distance $\ell \in [0, 0.125]$ m. The supplied controller approaches that face, pushes perpendicular to it, retracts, and waits for the block and finger to stop. We measure the time for this whole process (which takes time even when the push has 0 length.) The policy and critic observe the block's position and orientation, the finger's position, and the goal location after the system comes to rest. The 13 inputs are defined in `LearningPush.observation` in `push_env.py`.

The handwritten controller, which we call the *teacher*, chooses the face corresponding to the largest absolute component of the goal error in the block's coordinate frame. It limits the intended block displacement to 4 cm. A calibration accounts for stick travel before the block starts moving. A command requesting the maximum 4 cm block displacement therefore uses about 8.35 cm of stick travel. The teacher reaches the goal reliably but spends time on many short pushes.

Success means reaching within 0.025 m of the goal; orientation does not matter. An attempt ends when the block reaches the goal, falls, or leaves the allowed workspace. We also stop collecting an attempt after 32 commands to limit computation, even though the task could continue. That cutoff is different from a terminal state. For a valid action, with distance to the goal d_t and total simulated duration $\Delta\tau_t$, the reward is

$$r_t = \frac{d_t - d_{t+1}}{0.15} + \mathbf{1}\{\text{success}\} - 0.02\Delta\tau_t - \mathbf{1}\{\text{fall or escape}\}.$$

Invalid commands earn -0.2 without moving the simulator. We use $\gamma = 1$. Evaluation uses the rewards actually observed, including unfinished attempts.

Use natural logs, success fractions in $[0, 1]$, and three decimal places unless stated otherwise. Work out the *by hand* questions from the definitions, and read the *from the notebook* answers from its output. The notebook checks your functions using different numerical examples. The local BC fit and PPO update use fixed random seeds and supplied data, so a correct implementation reproduces the keyed values; only fitting time varies. The update need not improve performance to receive credit.

7.1 Progress and caution [2]

Gradescope: *By hand.* For T valid commands, the summed progress contribution is which expression?

A. $(d_0 - d_T)/0.15$**.**

B. $d_0/0.15$, regardless of final error.

C. $(d_0 - d_T)/(0.15T)$.

D. $\sum_t |d_t - d_{t+1}|/0.15$.

For $d_0 = 0.30$ m and $d_T = 0.015$ m this contribution is **1.900**. Suppose BC exactly reproduces the teacher's commands. Why might subsequent PPO training improve its reward?

A. Zero imitation error guarantees maximum task reward.

B. PPO can learn longer useful pushes that reach the goal in less total time.

C. Detours accumulate additional net progress reward.

Solution: *The progress terms telescope:*

$$\sum_{t=0}^{T-1} \frac{d_t - d_{t+1}}{0.15} = \frac{d_0 - d_T}{0.15} = \frac{0.30 - 0.015}{0.15} = 1.900.$$

Detours therefore earn no extra net progress. The time charge is separate: the teacher's short pushes require extra approaches, retractions, and waits. Fewer, longer useful pushes can reduce the charged time, even when the initial policy matches the demonstrations exactly.

7.2 **A one-parameter residual policy** We keep BC's rule for selecting a face: given state s, sample f from its predicted face probabilities. PPO changes only how far to push on that face. We use a one-dimensional Gaussian to choose a variable z, which we will convert to an allowed push distance below. [4]

To make learning efficient, we hand-pick one useful feature: the goal error perpendicular to the chosen face, e_f, measured in meters. We scale it as

$$g(s,f) = \text{clip}\left(\frac{|e_f| - 0.07}{0.05}, 0, 1\right).$$

Thus $g = 0$ for errors up to 7 cm, increases linearly to 1 at 12 cm, and stays at 1 for larger errors. The idea is to preserve BC's mean near the goal while allowing a correction when a longer push might help.

Let $\mu_{\text{BC},f}(s)$ be BC's predicted mean for z on face f. Our Gaussian has mean and standard deviation

$$\mu_\theta(s,f) = \mu_{\text{BC},f}(s) + \theta g(s,f), \qquad \sigma(s,f) = 0.2 + 0.3g(s,f).$$

The added term $\theta g(s,f)$ is the *residual*, or correction. The standard deviation grows from 0.2 near the goal to 0.5 farther away. We want to try a wider range of pushes when there is more room to move, and use less randomness near the goal, where precision matters. These standard deviations describe z, not distances in meters. The only policy parameter we tune is θ, starting at zero; it can become positive or negative. We also train a critic to estimate returns.

Finally, we sample z and convert it to stick travel:

$$z \sim \mathcal{N}\big(\mu_\theta(s,f), \sigma(s,f)^2\big), \qquad \ell = 0.125\,\text{sigmoid}(z), \qquad \text{sigmoid}(z) = \frac{1}{1+e^{-z}}.$$

The sigmoid keeps the commanded distance between 0 and 0.125 m, the allowed travel range. The Gaussian is therefore applied before this conversion; the resulting physical distance is not itself Gaussian.

There are more scaling constants here than we would like. The error thresholds and exploration scales are hand-chosen design choices, not values derived from an optimality argument. They let us keep the learning exercise small: PPO only has to tune one correction parameter.

Complete `joint_log_prob`. For each recorded action, it returns

$$\texttt{joint_log_prob} = \log p(f, z \mid s),$$

the log of the product of the face probability and the Gaussian density of z for that face.

Gradescope: *By hand.* If $|e_f| = 0.095$ m, $g =$ **0.500**. If $|e_f| \le 0.07$ m, the correction to the mean is **0.000**, for every θ. For $z = 0$, stick travel in meters (four decimals) is **0.0625**. For a recorded action, the unchanged face probability is 0.2, the old conditional Gaussian density is 0.5 and the new density is 0.4. For the stored pair (f, z), the old joint log density is **-2.303**; the new/old likelihood ratio is **0.800**. Which factor cancels from the new/old likelihood ratio, and why?

A. The face probability, because PPO does not change the face-selection policy.

B. The Gaussian density, because the recorded action is the same.

C. Both factors, because the BC network is unchanged.

Solution: *We substitute into the formulas for g and stick travel. For the recorded action, the joint density is the face probability times the conditional Gaussian density:*

$$\begin{aligned} g &= \text{clip}(0.025/0.05, 0, 1) = 0.500, \\ |e_f| \le 0.07: \quad g &= 0 \;\Rightarrow\; \theta g = 0.000, \\ \ell(0) &= 0.125 \cdot \tfrac{1}{2} = 0.0625\ m, \\ \log(0.2 \cdot 0.5) &= \log 0.1 = -2.303, \\ \rho &= \frac{0.2 \cdot 0.4}{0.2 \cdot 0.5} = 0.800. \end{aligned}$$

The face probability is unchanged, so it cancels. The Gaussian density can change as we update θ, even at the same recorded action. Near the goal, $g = 0$ preserves the BC mean; farther away, PPO can change it.

7.3 **Behavior cloning** Complete `bc_loss` by adding the mean negative log probability of the demonstrated face to the mean squared stick-travel error divided by $(0.10\ \text{m})^2$. Use the distance prediction for the demonstrated face. [3]

We supply 64 training demonstration episodes and 16 validation episodes (886 and 221 transitions). Every demonstration reaches the goal.

Run `clone_cautious` to fit a BC policy to these demonstrations. It trains for 400 epochs and keeps the network with the lowest validation loss.

Gradescope: *By hand.* For demonstrated-face probabilities $(0.8, 0.5)$ and distance errors $(0.01, -0.02)$ m, the mean BC loss is **0.483**. *From the notebook* (`clone_cautious`). Your fit: validation face accuracy, as a fraction **0.995**; distance mean absolute error (cm) **0.128**; fitting time (s) **varies**. *By hand.* Why do we assign each whole demonstration episode to either the training set or the validation set, rather than randomly splitting individual transitions between the two sets?

A. Adjacent states can be nearly duplicates, making a random split of individual transitions overly optimistic.

B. Validation labels must all be failures.

C. The network cannot accept different episodes in one batch.

Does zero imitation error guarantee maximum task reward? Enter yes or no: **no**.

Solution: *The BC loss adds the mean negative face log probability to the mean squared distance error, scaled by 0.10 m:*

$$-\tfrac{1}{2}(\log 0.8 + \log 0.5) = \tfrac{1}{2}(0.22314 + 0.69315) = 0.45815,$$
$$\tfrac{1}{2}\left(0.1^2 + 0.2^2\right) = 0.025,$$
$$\textit{loss} = 0.45815 + 0.025 = 0.483145\ldots$$

Adjacent states within an episode are nearly duplicates, so whole-episode validation tests imitation on new trajectories. Prediction error and closed-loop reward are different quantities, so zero imitation error need not maximize task reward. The 400-epoch fit is seeded, so a correct `bc_loss` gives validation face accuracy 0.995 (220 of 221 labels) and distance MAE 0.128 cm; only the fitting time depends on the machine.

7.4 **Generalized advantage estimation** Complete `gae_advantages` using the recorded rewards and critic values for one episode. At success or physical failure, the next-state value is zero. At the 32-command cutoff, we keep $V_{\text{old}}(s_T)$ to estimate the return from continuing. With $\gamma = 1$, work backwards using [4]

$$\delta_t = r_t + V_{\text{old}}(s_{t+1}) - V_{\text{old}}(s_t),$$
$$\widehat{\text{adv}}_t = \delta_t + \lambda \widehat{\text{adv}}_{t+1}, \qquad \widehat{\text{adv}}_T = 0, \quad \lambda = 0.95.$$

Return these advantages without rescaling them. The supplied `batch_targets` calls your function separately for each episode, then subtracts the batch mean and divides by the batch standard deviation. For the critic, it instead uses the return targets $\widehat{G}_t = \sum_{j=t}^{T-1} r_j + V_{\text{old}}(s_T)$, again setting the last value to zero at a terminal state. These targets are not rescaled.

Gradescope: *By hand.* For rewards $(0, 1)$, old values $(0.2, 0.4)$ and next-values $(0.4, 0)$, $\widehat{\text{adv}}_0 =$ **0.770**. If the episode instead ends at a computational cutoff with final old value 0.5, then $\widehat{\text{adv}}_0 =$ **1.245** and $\widehat{G}_0 =$ **1.500**. A computational cutoff should always replace the last next-value by zero. Enter true or false: **false**.

Solution: *We compute* $\delta_t = r_t + V(s_{t+1}) - V(s_t)$*, then work backwards using* $\widehat{\text{adv}}_t = \delta_t + \lambda\widehat{\text{adv}}_{t+1}$*:*

$$\begin{aligned} \delta_0 &= 0 + 0.4 - 0.2 = 0.200, \qquad \delta_1 = 1 + 0 - 0.4 = 0.600, \\ \widehat{\text{adv}}_0 &= 0.2 + 0.95(0.6) = 0.770, \\ \textit{cutoff: } \delta_1 &= 1 + 0.5 - 0.4 = 1.1, \qquad \widehat{\text{adv}}_0 = 0.2 + 0.95(1.1) = 1.245, \\ \widehat{G}_0 &= 0 + 1 + 0.5 = 1.500. \end{aligned}$$

At a cutoff, the task could continue, so we retain $V_{\text{old}}(s_T)$*. Only a terminal state has no future reward.*

7.5 **The clipped PPO update** Complete `ppo_actor_loss` and `critic_loss`. For the same recorded actions, $\rho_t = \exp(\log \pi_\theta - \log \pi_{\text{old}})$. The actor minimizes [4]

$$-\frac{1}{N}\sum_t \min\big(\rho_t \widehat{\text{adv}}_t, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\widehat{\text{adv}}_t\big).$$

Here N is the number of transitions passed to the loss function. Use the advantages supplied by `batch_targets` without rescaling them again. The old log probabilities, advantages, and targets stay fixed while we update the policy.

For `critic_loss`, return the mean squared error between the critic's predictions and $\widehat{G}_t$. The supplied update code adds $1/4$ times that loss to the actor loss, so leave the factor $1/4$ out of your function.

Gradescope: *By hand.* For $\epsilon = 0.2$, ratios $(1.4, 0.7, 1.1, 0.9)$ and fixed advantages $(1, -1, 2, -2)$, the mean actor loss is **-0.200**. For predicted values $(1, 3)$ and targets $(2, 1)$, `critic_loss` is **2.500**. Clipping guarantees every updated ratio lies in $[0.8, 1.2]$. Enter true or false: **false**.

Solution: *For each actor term, we take the smaller of* $\rho\,\widehat{\text{adv}}$ *and* $\text{clip}(\rho, 0.8, 1.2)\,\widehat{\text{adv}}$*. The actor loss is the negative mean of those terms:*

$$\begin{aligned} \textit{terms} &= \min(1.4, 1.2),\ \min(-0.7, -0.8),\ \min(2.2, 2.2),\ \min(-1.8, -1.8) \\ &= 1.2,\ -0.8,\ 2.2,\ -1.8, \\ \textit{actor loss} &= -\tfrac{1}{4}(1.2 - 0.8 + 2.2 - 1.8) = -0.200, \\ \textit{critic loss} &= \tfrac{1}{2}\big[(1-2)^2 + (3-1)^2\big] = 2.500. \end{aligned}$$

Minimizing the negative mean maximizes the clipped objective. Clipping can make the objective flat, but does not constrain every updated ratio or guarantee a higher return.

7.6 **One short coding experiment** Run `update_from_batch`. It loads a saved policy and episodes collected by that policy, computes the advantages and return targets, and uses your losses to update the policy and critic. It makes up to four passes through the batch, stopping early if the policy changes too much. No new simulation is required. Repeating the call reuses a cached result; with `retrain=True`, it repeats the exercise from the same saved policy. [1]

Gradescope: *From the notebook* (`update_from_batch`). Your update: initial θ **0.691**; updated θ **0.728**; optimizer steps **12**. *By hand.* To continue PPO training after changing the policy, can we use arbitrary supplied episodes instead of collecting new episodes with the updated policy? Enter yes or no: **no**.

Solution: *The initial* $\theta = 0.691$ *is stored with the supplied batch. The update uses a fixed random seed and gives* $\theta = 0.728$ *after 12 optimizer steps: the 185 transitions form three minibatches of at most 64, repeated for four epochs, with no early stop from the KL check.*

PPO reuses episodes collected by the policy being updated for a limited number of optimization steps. Supplied episodes from other policies cannot generally replace new episodes from our updated policy. Clipping does not remove that requirement.

7.7 **Uploading the coding experiment** Upload the `update_from_batch` output from part 7.6 and your BC validation table from part 7.3 to Gradescope. [1]

7.8 **Results from longer training runs** We supply three initial runs, labeled `exploratory` in the notebook, with saved policies after about 4,096 and 8,192 pushes. Training continues without restarting at the midpoint. Compare the two stages on the same 64 test goals. [3]

Use the *greedy* policies: choose the most probable face and use the Gaussian mean for z, without sampling. The notebook labels these stages `midpoint` and `final`. The test goals were not used to choose policies or training settings.

Gradescope: The function `aggregate_table` averages each policy's results over the 64 goals, then averages the three runs equally. *From the notebook* (`aggregate_table`), enter mean total simulated seconds (two decimals): BC **87.92**; residual at 4,096 **71.58**; residual at 8,192 **63.22**. Final success fraction is **1.000**. *By hand.* Using the unrounded times from the notebook, compute the time saved by the final policy as a percentage of BC's time:

$$100\,\frac{T_{\mathrm{BC}} - T_{\mathrm{final}}}{T_{\mathrm{BC}}} = \textbf{28.097}\,\%.$$

To check whether the improvement holds in new training runs, we also supply ten independent runs, labeled `confirmatory`. The number of runs and the 8,192-push budget were fixed in advance. All ten start from the same supplied BC policy and are evaluated on the same 64 test goals.

For each run i, let D_i be the final greedy policy's mean test return minus BC's mean return on those goals. Apply a two-sided one-sample t-test of $\mathbb{E}[D_i] = 0$ to the ten differences. Compute a bootstrap interval by resampling these differences (5,000 resamples, seed 815), keeping all ten runs in the analysis. The notebook also reports a sign test, which checks how often the differences are positive without assuming normality.

Gradescope: *By hand.* How many independent training-run differences are there? $n =$ **10**; degrees of freedom **9**. *From the notebook* (`training_comparison`): mean reward gain **0.479**; 95% interval [**0.457**, **0.500**]. $t =$ **41.817**; two-sided $p =$ **1.28** $\times 10^{-11}$ (enter the number multiplying 10^{-11}, to three significant figures). *By hand.* Which uncertainty does this analysis estimate?

- **A. Variation between PPO training runs starting from this BC policy and evaluated on these goals.**
- B. Variation between new BC fits and arbitrary new environments.
- C. 640 independent PPO training runs.

Report the effect and observed p-value whether or not the test rejects.

Solution: *The exploratory study's* `aggregate_table` *gives mean times of* 87.92 *s for BC,* 71.58 *s after 4,096 steps, and* 63.22 *s after 8,192 steps, with final success* 1.000*. Using the unrounded data, the time reduction* $(T_{\mathrm{BC}} - T_{\mathrm{final}})/T_{\mathrm{BC}}$ *is* 28.097%*. The rounded times give* $(87.92 - 63.22)/87.92 =$ 28.094%*.*

For the confirmatory study, the independent units are the ten trained policies, so $n = 10$ *and we have* 9 *degrees of freedom. We apply a one-sample t-test to each policy's final-greedy mean return minus BC's mean return on the same goals. The* `training_comparison` *results are gain* 0.479*, interval* $[0.457, 0.500]$*,* $t = 41.817$*, and* $p = 1.28e-11$*.*

The interval describes uncertainty in the mean improvement over repeated PPO training runs, with this BC policy and test set held fixed. The t-test is exact for independent, normally distributed differences and otherwise approximate. The sign test does not need the normality assumption. Neither analysis measures variation from fitting new BC policies or collecting new demonstrations.

Going further: Why do we evaluate with the greedy policy rather than drawing actions from its distribution?

7.9 **Uploading the training results** Upload the learning curves and individual run-mean table from part 7.8 to Gradescope. [1]

Going further: Why is this still reinforcement learning when only one actor parameter changes?

Solution: *We update θ using task rewards and states visited during sampled multi-step trajectories. There is no corrected-action label to fit, so the update is reinforcement learning even though the actor has only one learned parameter.*

The simple policy class makes the update easy to inspect. Policy gradients are not the only way to optimize a scalar correction.

2.1 Additional problems for 6.4212

Problem 8 A baseline can reduce noise 6.4212 only [9]

Return to Problem 2, with rewards $1, 3$ and equal action probabilities. Look only at the R component of the gradient. With a constant baseline b, it is

$$g_{R,b} = (r(a) - b)\nabla_{z_R} \log p_a \quad .$$

Here the subscript R identifies the component associated with the weight z_R, as in Problem 2, and b identifies the baseline used. The notation p_a means p_L when the sampled action is L and p_R when it is R. The expression $\nabla_{z_R} \log p_a$ differentiates with respect to the weight z_R for either sampled action.

The log-probability derivative is $-1/2$ for L and $+1/2$ for R. For the baseline argument, see "REINFORCE with baseline" in the Reinforcement Learning notes.

Use $\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

Gradescope:

8.1 Without a baseline ($b = 0$), give $\mathbb{E}[g_{R,0}] =$ **1/2** and $\mathrm{Var}(g_{R,0}) =$ **1**. [2]

Solution: *We first compute the two equally likely gradient samples, then their mean and variance:*

$$\begin{aligned} g_{R,0}(L) &= 1 \cdot (-\tfrac{1}{2}) = -\tfrac{1}{2}, \qquad g_{R,0}(R) = 3 \cdot \tfrac{1}{2} = \tfrac{3}{2}, \\ \mathbb{E}[g_{R,0}] &= \tfrac{1}{2}\left(-\tfrac{1}{2} + \tfrac{3}{2}\right) = \tfrac{1}{2}, \\ \mathrm{Var}(g_{R,0}) &= \tfrac{1}{2}\left(\tfrac{1}{4} + \tfrac{9}{4}\right) - \tfrac{1}{4} = 1. \end{aligned}$$

8.2 Use the policy's exact mean reward, $b = 2$. Give $g_{R,2}(L) =$ **1/2**, $g_{R,2}(R) =$ **1/2**, and $\mathrm{Var}(g_{R,2}) =$ **0**. [2]

Solution: *Subtracting $b = 2$ from each reward gives*

$$\begin{aligned} g_{R,2}(L) &= (1-2)(-\tfrac{1}{2}) = \tfrac{1}{2}, \\ g_{R,2}(R) &= (3-2)(\tfrac{1}{2}) = \tfrac{1}{2}, \\ \mathrm{Var}(g_{R,2}) &= 0. \end{aligned}$$

The baseline makes L worse than expected and R better than expected. Discouraging L and encouraging R both shift probability toward R, and here the two updates agree exactly. That is a property of this tiny example, not a general guarantee.

8.3 Usually we do not know the mean reward. Consider this two-stage experiment, keeping the policy fixed at $p_L = p_R = 1/2$ throughout: [2]

1. Sample m independent actions and average their rewards to obtain the baseline b.
2. Sample one *additional, independent* action a, observe its reward $r(a)$, and compute $g_{R,b} = (r(a) - b)\nabla_{z_R} \log p_a$. This new reward is *not* included in the average defining b.

A "fresh action" means a new independent draw; it can still be L or R, including an action seen in the earlier samples.

If we repeat the *whole experiment*, the first stage produces a different estimate b in different repetitions. We must account for this uncertainty as well as the randomness of the final action. A *conditional* mean or variance holds b fixed and averages only over the final action. An *unconditional* mean or variance averages over both stages, including the possible values of b.

Each earlier reward has mean 2 and variance 1, so $\mathbb{E}[b] = 2$ and $\mathrm{Var}(b) = 1/m$. For a fixed b, the two equally likely gradient samples are $(b-1)/2$ and $(3-b)/2$. First find their conditional mean and variance as functions of b:

$\mathbb{E}[g_{R,b} \mid b] =$ 1/2, $\mathrm{Var}(g_{R,b} \mid b) =$ $(b-2)^2/4$.

Then account for the variation in b between repetitions, using $\mathrm{Var}(X) = \mathbb{E}[\mathrm{Var}(X \mid Y)] + \mathrm{Var}(\mathbb{E}[X \mid Y])$. For $m = 4$, give the unconditional $\mathbb{E}[g_{R,b}] =$ **1/2** and $\mathrm{Var}(g_{R,b}) =$ **1/16**.

Solution: *For a fixed b, the two samples are $(b-1)/2$ and $(3-b)/2$. We first average over the final action, then use the law of total variance to include the uncertainty in b:*

$$\begin{aligned}
\mathbb{E}[g_{R,b} \mid b] &= \tfrac{1}{2}\left(\tfrac{b-1}{2} + \tfrac{3-b}{2}\right) = \tfrac{1}{2}, \\
\mathrm{Var}(g_{R,b} \mid b) &= \tfrac{1}{2}\left(\tfrac{b-2}{2}\right)^2 + \tfrac{1}{2}\left(\tfrac{2-b}{2}\right)^2 = \tfrac{(b-2)^2}{4}, \\
\mathbb{E}[g_{R,b}] &= \tfrac{1}{2}, \\
\mathrm{Var}(g_{R,b}) &= \mathbb{E}\left[\tfrac{(b-2)^2}{4}\right] + \mathrm{Var}\left(\tfrac{1}{2}\right) = \tfrac{\mathrm{Var}(b)}{4} + 0 = \tfrac{1}{4m} = \tfrac{1}{16}.
\end{aligned}$$

The expectation is over the first-stage samples that determine b; substituting $b = \mathbb{E}[b] = 2$ would wrongly discard their uncertainty.

8.4 Instead, take just one fresh sample and use its own reward as its baseline: $b = r(a)$. What gradient results? **0**. Does the expected value of this update equal the true gradient component $1/2$? **No**. [1]

Solution: *Using the sample's own reward as its baseline gives $r(a) - b = 0$ for every sample, so $g = 0$ and $\mathbb{E}[g] = 0 \neq 1/2$. The baseline depends on the sampled action, which breaks the zero-mean argument for action-independent baselines.*

8.5 Now try a baseline for the Gaussian in Problem 3 at $(\mu, \sigma) = (0, 1)$, looking only at the mean parameter's gradient $g_{\mu,b}$. For $a \sim \mathcal{N}(0,1)$, use $\mathbb{E}[a^2] = 1$, $\mathbb{E}[a^4] = 3$, $\mathbb{E}[a^6] = 15$, and zero odd moments. Show that [2]

$$\mathbb{E}[g_{\mu,b}] = 2 \qquad \mathrm{Var}(g_{\mu,b}) = (b+4)^2 + 14 \ .$$

Give the variance with no baseline, **30**; with the mean-return baseline $b = -2$, **18**; and with $b = -4$, **14**. Does the mean return always give the lowest-variance baseline? **No**.

Solution: *At $(0,1)$, the mean parameter's log-density derivative is a, so $g_{\mu,b} = (r(a) - b)a$. We expand and use the supplied moments:*

$$\begin{aligned}
g_{\mu,b} &= (r(a) - b)\, a = -a^3 + 2a^2 - (1+b)a, \\
\mathbb{E}[g_{\mu,b}] &= 2\mathbb{E}[a^2] = 2, \\
\mathbb{E}[g_{\mu,b}^2] &= \mathbb{E}[a^6] + 4\mathbb{E}[a^4] + (1+b)^2\mathbb{E}[a^2] + 2(1+b)\mathbb{E}[a^4] \\
&= 15 + 12 + (1+b)^2 + 6(1+b) = 34 + 8b + b^2, \\
\mathrm{Var}(g_{\mu,b}) &= 34 + 8b + b^2 - 4 = (b+4)^2 + 14.
\end{aligned}$$

(The cross terms $-4a^5$ and $-4(1+b)a^3$ have zero mean.) Then $b = 0$ gives 30, $b = -2$ gives 18, and $b = -4$ gives 14. The mean-return baseline helps but is not optimal: large scores amplify some samples more than others.

Going further: The last part shows that the mean return is not always the best constant baseline. Write $s(a) = \partial \log \pi_{\mu,\sigma}(a)/\partial\mu$, so that $g_{\mu,b} = (r(a) - b)\, s(a)$. Show that the constant baseline minimizing $\mathrm{Var}(g_{\mu,b})$ is

$$b^* = \frac{\mathbb{E}[r(a)\, s(a)^2]}{\mathbb{E}[s(a)^2]} \ ,$$

and check that it gives $b^* = -4$ for the Gaussian. When does b^* equal the mean return?

Solution: *The mean $\mathbb{E}[(r-b)s] = \mathbb{E}[rs]$ does not depend on b, so minimize the second moment:*

$$\mathbb{E}[(r-b)^2 s^2] = \mathbb{E}[r^2 s^2] - 2b\,\mathbb{E}[rs^2] + b^2 \mathbb{E}[s^2],$$
$$\frac{d}{db}: \quad 0 = -2\mathbb{E}[rs^2] + 2b\,\mathbb{E}[s^2] \quad \Rightarrow \quad b^* = \frac{\mathbb{E}[rs^2]}{\mathbb{E}[s^2]}.$$

For the Gaussian, $s = a$ and $r = -(a-1)^2$:

$$b^* = \frac{-\mathbb{E}[a^4 - 2a^3 + a^2]}{\mathbb{E}[a^2]} = -(3+1) = -4.$$

b^* weights each reward by s^2, so it equals $\mathbb{E}[r]$ when s^2 is the same for every action, as in the two-action parts, where $s = \pm 1/2$ and $b = 2$ removed all the variance.

A Bootstrap confidence intervals

This section is a quick summary. For a more authoritative account, read the Berkeley Stat 20 notes on bootstrapping.

Statistical methods let us estimate confidence intervals around a quantity, like the mean of a distribution. A 95% confidence interval is the range of values such that: if we repeatedly gathered this amount of data from this process and estimated an interval this way, the true value would end up in this interval 95% of the time.

Classical statistical methods make strong assumptions about the data distribution. The *bootstrap* method does not. It treats our observed sample as a small population and repeatedly samples from it.

When we are interested in deciding whether method A is better than method B, if we can run them both in the same conditions, then we can do statistics on their *per-test difference*, and get more useful information out of less data.

For the cart-pole comparison, each data point is the difference in mean evaluation return of one training pair (same network initialization and data),

$$D_i = \bar{R}_{i,\text{baseline}} - \bar{R}_{i,\text{plain}}, \qquad i = 1, \ldots, n, \qquad n = 30.$$

Our estimate of the population mean difference is $\bar{D} = \frac{1}{n}\sum_i D_i$. A positive difference means the baseline worked better.

Our notebook computes a *percentile bootstrap* 95% confidence interval as follows:

1. Draw n values from $D_1, \ldots, D_n$ **with replacement**, uniformly at random. The resample has the same size as the original sample, but some observations may appear several times and others may be omitted.
2. Compute the mean of this resample and save it as one bootstrap estimate of the mean.
3. Repeat these two steps 5000 times, using a new random resample each time. This produces 5000 bootstrap mean estimates.
4. Take the 2.5th and 97.5th percentiles of these means as the lower and upper endpoints. Approximately 2.5% of the bootstrap means lie below the interval and 2.5% lie above it.

For a tiny illustration, if the observed differences were $(-2, 1, 3, 6)$, one possible resample would be $(3, -2, 3, 6)$. The original mean is 2; this resample's mean is 2.5.

Paired testing Both methods' results for a training seed form one sampling unit; taking their difference first preserves that pairing. The 100 evaluation episodes of one policy are not 100 independent training runs. For learning curves, the same sampled pair IDs are used at every checkpoint, and percentiles are computed separately at each checkpoint. These are *pointwise* intervals, not a guarantee of simultaneous coverage of the whole curve. For the pushing experiments' goal-bootstrap intervals, the units are instead goal IDs, sampled jointly across controllers; those intervals describe variation across goals with the fitted policies held fixed.

What does 95% mean? The intended interpretation is that, across repeated collection of new data and construction of intervals, about 95% of the intervals cover the true population mean difference. Bootstrap coverage is approximate and depends on having independent, representative sampling units. The interval describes uncertainty about the mean, not the range containing 95% of episode returns. More bootstrap resamples make its estimated endpoints more stable; do not decrease the size of the interval.