Assignment 3: Q-Learning and Actor-Critic Algorithms

Due October 18, 11:59 pm

1 Multistep Q-Learning

Consider the N-step variant of Q-learning described in lecture. We learn $Q_{\phi_{k+1}}$ with the following updates:

$$y_{j,t} \leftarrow \left(\sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{j,t'} \right) + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi_k} \left(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N} \right) \tag{1}$$

$$\phi_{k+1} \leftarrow \arg\min_{\phi \in \Phi} \sum_{j,t} \left(y_{j,t} - Q_\phi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t}) \right)^2 \tag{2}$$

In these equations, j indicates an index in the replay buffer of trajectories $\mathcal{D}_k$. We first roll out a batch of B trajectories to update $\mathcal{D}_k$ and compute the target values in (1). We then fit $Q_{\phi_{k+1}}$ to these target values with (2). After estimating $Q_{\phi_{k+1}}$, we can then update the policy through an argmax:

$$\pi_{k+1}\left(\mathbf{a}_t \mid \mathbf{s}_t\right) \leftarrow \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_{\phi_{k+1}}\left(\mathbf{s}_t, \mathbf{a}_t\right) \\ 0 \text{ otherwise.} \end{cases} \tag{3}$$

We repeat the steps in eqs. (1) to (3) K times to improve the policy. In this question, you will analyze some properties of this algorithm, which is summarized in Algorithm 1.

Algorithm 1 Multistep Q-Learning

Require: iterations K, batch size B
1: initialize random policy π_0, sample $\phi_0 \sim \Phi$
2: **for** $k = 0 \dots K-1$ **do**
3: Update $\mathcal{D}_{k+1}$ with B new rollouts from π_k
4: compute targets with (1)
5: $Q_{\phi_{k+1}} \leftarrow$ update with (2)
6: $\pi_{k+1} \leftarrow$ update with (3)
7: **end for**
8: **return** π_K

1.1 TD-Learning Bias (2 points)

We say an estimator $f_{\mathcal{D}}$ of f constructed using data $\mathcal{D}$ sampled from process P is *unbiased* when $\mathbb{E}_{\mathcal{D}\sim P}[f_{\mathcal{D}}(x) - f(x)] = 0$ at each x.

Assume $\hat{Q}$ is a noisy (but unbiased) estimate for Q. Is the Bellman backup $\mathcal{B}\hat{Q} = r(s,a) + \gamma \max_{a'} \hat{Q}(s',a')$ an unbiased estimate of $\mathcal{B}Q$?

☐ Yes

☐ No

Solution:

我们知道：

* $\hat{Q}$ 是无偏的：

$$\mathbb{E}[\hat{Q}(s,a)] = Q(s,a)$$

但是现在我们不是直接用 $\hat{Q}$，而是用了一个非线性操作：

$$\max_{a'} \hat{Q}(s', a')$$

问题就出在这里。

—

□ 非线性操作破坏无偏性

“max”是一个**非线性算子**，因此：

$$\mathbb{E}[\max_{a'} \hat{Q}(s', a')] \neq \max_{a'} \mathbb{E}[\hat{Q}(s', a')]$$

换句话说，即使每个动作的 $\hat{Q}(s', a')$ 都是无偏的，取最大值之后，期望值会 **偏高**（overestimate）。

—

举个简单例子

假设只有两个动作 a_1, a_2，它们的真实值一样：

$$Q(s', a_1) = Q(s', a_2) = 1$$

而估计有噪声：

$$\hat{Q}(s', a_1) = 1 + \epsilon_1, \quad \hat{Q}(s', a_2) = 1 + \epsilon_2$$

其中 ϵ_i 是零均值噪声（即无偏）。

那我们有：

$$\mathbb{E}[\max(\hat{Q}_1, \hat{Q}_2)] = 1 + \mathbb{E}[\max(\epsilon_1, \epsilon_2)] > 1$$

所以即使单个估计无偏，取最大值后整体**偏高**。

—

结论

因此：

$$\boxed{\mathcal{B}\hat{Q} = r + \gamma \max_{a'} \hat{Q}(s', a')}$$

并不是 $\mathcal{B}Q = r + \gamma \max_{a'} Q(s', a')$ 的无偏估计。也就是说，**Bellman backup 有正偏差（overestimation bias）**。

正确答案是：**No**。

1.2 Tabular Learning (6 points total)

At each iteration of the algorithm above after the update from eq. (2), Q_{ϕ_k} can be viewed as an estimate of the true optimal Q^*. Consider the following statements:

I. $Q_{\phi_{k+1}}$ is an unbiased estimate of the Q function of the last policy, Q^{π_k}.

II. As $k \to \infty$ for some fixed B, Q_{ϕ_k} is an unbiased estimate of Q^*, i.e., $\lim_{k\to\infty} \mathbb{E}\big[Q_{\phi_k}(s, a) - Q^*(s, a)\big] = 0$.

III. In the limit of infinite iterations and data we recover the optimal Q^*, i.e., $\lim_{k,B\to\infty} \mathbb{E}\left[\|Q_{\phi_k} - Q^*\|_\infty\right] = 0$.

We make the additional assumptions:

- The state and action spaces are finite.

- Every batch contains at least one experience for each action taken in each state.
- In the tabular setting, Q_{ϕ_k} can express any function, i.e., $\{Q_{\phi_k} : \phi \in \Phi\} = \mathbb{R}^{S \times A}$.

When updating the buffer $\mathcal{D}_k$ with B new trajectories in line 3 of Algorithm 1, we say:

- When learning *on-policy*, $\mathcal{D}_k$ is set to contain only the set of B new rollouts of π (so $|\mathcal{D}_k| = B$). Thus, we only train on rollouts from the current policy.
- When learning *off-policy*, we use a fixed dataset $\mathcal{D}_k = \mathcal{D}$ of B trajectories from another policy π'.

Indicate which of the statements **I-III** always hold in the following cases. No justification is required.

	I.	**II.**	**III.**
1. $N = 1$ and …			
(a) on-policy in tabular setting	☐	☐	☐
(b) off-policy in tabular setting	☐	☐	☐
2. $N > 1$ and …			
(a) on-policy in tabular setting	☐	☐	☐
(b) off-policy in tabular setting	☐	☐	☐
3. In the limit as $N \to \infty$ (no bootstrapping) …			
(a) on-policy in tabular setting	☐	☐	☐
(b) off-policy in tabular setting	☐	☐	☐

Solution: 好～我们把题目里的三条断言 **I–III**，在 6 个场景（N=1/N>1/$N\to\infty$ × on/off-policy）**逐一**解释清楚。先回顾三条断言含义：

* **I**：一次更新后，$Q_{\phi_{k+1}}$ 是**上一轮策略** π_k 的 Q^{π_k} 的**无偏估计**。

* **II**：固定批量大小 B，当 $k \to \infty$ 时，Q_{ϕ_k} 是 Q^* 的**无偏估计**（$\lim_{k\to\infty} \mathbb{E}[Q_{\phi_k} - Q^*] = 0$）。

* **III**：当 $k, B \to \infty$（无限迭代且每次有无限数据）时，恢复最优 Q^*（$\lim_{k,B\to\infty} \mathbb{E}\|Q_{\phi_k} - Q^*\|_\infty = 0$）。

题目假设：**tabular** 表示能力完备；状态动作有限；每个训练 batch 对每个 (s, a) 都有至少一次样本（覆盖性）。

—

预备：几个算子与 N 步目标

* **策略评估算子**：$(\mathcal{T}^\pi Q)(s, a) = r(s, a) + \gamma\, \mathbb{E}_{s'} \mathbb{E}_{a' \sim \pi}[Q(s', a')]$

* **最优算子**：$(\mathcal{T}^* Q)(s, a) = r(s, a) + \gamma\, \mathbb{E}_{s'}[\max_{a'} Q(s', a')]$（收敛不动点为 Q^*）

* **N 步最优 backup**（题里目标）：

$$y_t = \sum_{i=0}^{N-1} \gamma^i r_{t+i} + \gamma^N \max_a Q(s_{t+N}, a).$$

on-policy 时，前 $N-1$ 步动作来自 π_k；off-policy 时，来自行为策略 π'。

要点：只要目标里出现 **max**，它对应的是朝 **最优算子 $\mathcal{T}^*$** 的更新，而不是对 π_k 的评估 $\mathcal{T}^{\pi_k}$。

在强化学习里，一个策略 π 决定了动作选择方式：

$$a_t \sim \pi(\cdot|s_t)$$

● on-policy

* 你在学习 Q^π（当前策略的价值函数）。* 数据样本也是由同一个策略 π 产生的。

即：

采样策略 = 学习目标策略 = π

举例: 你用当前的 ε -greedy 策略与环境交互，并用这些数据来更新自己的 Q 值。→ 这是 on-policy（比如 **SARSA** 算法）。

—

● off-policy

* 你想学习 Q^π（目标策略的价值函数），但数据是由另一个策略 π′ 产生的。

即：

采样策略 π' ≠ 目标策略 π

举例：你在看别人玩游戏（π′），但你想学习最优策略 π。→ 这是 off-policy（比如 **Q-learning** 算法）。

On-policy 更新（例如 SARSA）：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big]$$

下一步动作 a_{t+1} 是按照**当前策略 π** 选的。

—

Off-policy 更新（例如 Q-learning）：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\big]$$

下一步使用 $\max_{a'}$，表示学习的是**最优策略 π**，而不是生成数据的行为策略。

—

1) $N = 1$

(a) on-policy（每次用 π_k 采样 B 条新轨迹，仅用本次数据训练）

* **I：不成立。** 一步目标是 $r + \gamma \max_{a'} Q_k(s', a')$，这是 $\mathcal{T}^* Q_k$ 的样本近似，估计的是最优策略而不是上一轮的策略，不是 $\mathcal{T}^{\pi_k} Q_k$。所以并非在无偏评估 Q^{π_k}。

* **II：不成立。** B 固定、每次只用当轮有限样本，且目标含 max（**maximization bias**）。即便 $k \to \infty$，“无偏估计 Q^*”太强，$\mathbb{E}[\max_{a'} \hat{Q}(s', a')] \neq \max_{a'} \mathbb{E}[\hat{Q}(s', a')]$ 达不到。

* **III：成立。** 在 tabular + 完全覆盖 + $B \to \infty$ 的极限下，每次 backup 逼近 $\mathcal{T}^*$，而 $\mathcal{T}^*$ 在 $\|\cdot\|_\infty$ 下是 γ-收缩映射，迭代收敛到唯一不动点 Q^*。

(b) off-policy（固定一个来自 π' 的数据集 $\mathcal{D}$）

* **I：不成立。** 同上，目标是朝 $\mathcal{T}^*$ 而非 $\mathcal{T}^{\pi_k}$；且数据分布与 π_k 不同，更不满足“无偏评估 Q^{π_k}”。

* **II：不成立。** 理由同上。

* **III: 不成立。** 如果数据对每个 (s, a) 有覆盖，**Fitted Q-Iteration（FQI）**：$B \to \infty$ 时经验 backup → $\mathcal{T}^*$，重复迭代收敛到 Q^*（与行为策略无关，因为一步 backup 中并不需要修正分布）。$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$ 这个不包含任何来自行为策略 π′ 的动作选择，即时奖励 r_t 只由当前动作状态决定和下一个状态 s_{t+1} 是由分布环境 $P(s_{t+1}|s_t, a_t)$ 决定的。但是此处的：Learning off-policy with fixed dataset $\mathcal{D}$ of B trajectories from another policy π'，如果数据集 $\mathcal{D}$ 中没有覆盖所有的 (s, a) 对，那么就无法保证收敛到 Q^*。因此 III 不成立。

2) $N > 1$（有限多步）

(a) on-policy

* **I：不成立。** 目标：前 N 步是真实回报，但**第 N 步用 max 自举**，它并非 $\mathcal{T}^{\pi_k}$（后者应在每一步都对 π_k 取期望）。

* **II：不成立。** 依旧是有限样本 + max 带来的偏差，固定 B 不能保证“无偏到 Q^*”。

* **III：成立。** 在 tabular、覆盖且 $B \to \infty$ 的极限下，N 步最优 backup 的期望等价于把 $\mathcal{T}^*$ 置于第 N 步（可视作 γ^N-收缩），配合贪心改进形成广义策略迭代，最终收敛到 Q^*。

(b) off-policy

* **I：不成立。** 前 $N-1$ 步的奖励来自行为策略 π' 的轨迹，然而目标要“朝最优”，分布不匹配且无修正，**不是**对 Q^{π_k} 的无偏评估。

* **II：不成立。** 同上。

* **III：不成立。** 多步 off-policy **如果不做重要性采样校正**，即使数据无限，也会收敛到错误的固定点（N 步回报混入了 π' 的决策），不能保证得到 Q^*。

当 $N > 1$ 时，

$$y_t^{(N)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{N-1} r_{t+N-1} + \gamma^N \max_a Q(s_{t+N}, a)$$

问题：这些 r_{t+i} 来自谁？

来自行为策略 π' 在环境中“连续选择的动作”所产生的轨迹。

于是：

- 奖励序列的统计特性取决于 π'；
- 也就是说，

$$\mathbb{E}_{\pi'}[\, r_t + \gamma r_{t+1} + \cdots + \gamma^{N-1} r_{t+N-1}\,] \;\neq\; \mathbb{E}_{\pi^*}[\cdots].$$

换句话说：

你用 π' 的行为轨迹去估计 π^* 的未来回报，期望错了。

这导致多步目标不再是 $\mathcal{T}^*$ 的期望，而是某个混合算子：

$$\tilde{\mathcal{T}}_N^{\pi'}(Q) = \mathbb{E}_{\pi'}\left[\sum_{i=0}^{N-1} \gamma^i r_{t+i} + \gamma^N \max_a Q(s_{t+N}, a)\right]$$

这个算子的固定点不再是 Q^*。

—

3) $N \to \infty$（纯 Monte Carlo，无自举）

(a) on-policy

* **I：成立。** 目标就是完整回报 G_t；on-policy 下 $\mathbb{E}[G_t|s,a] = Q^{\pi_k}(s,a)$，MC 评估对 Q^{π_k} **无偏**。

* **II：不成立。** 题目要求“固定 B”，每轮只用 B 条新样本做评估并立刻贪心改进，带噪声的改进过程不保证“$\lim_{k\to\infty}$ 无偏到 Q^*”；这个断言比“收敛”更强，一般不成立。

* **III：成立。** $B \to \infty$ 时，MC 对 Q^{π_k} 的估计一致；与贪心改进交替（MC Policy Iteration）在 tabular 下收敛到 Q^*。

(b) off-policy

* **I：不成立。** MC **off-policy** 若不做重要性采样，$\mathbb{E}_{\pi'}[G_t] \neq Q^{\pi_k}$，评估有系统偏差。想得到 Q^{π_k}，必须用但是采样是在 π' 下进行的轨迹，需要用重要性采样校正。

* **II：不成立。** 同理上，没有重要性采样校正，MC off-policy 评估有偏差，不能保证无偏到 Q^*。

* **III：不成立。** 无 IS 校正时，MC off-policy 不能保证一致性，更谈不上收敛到 Q^*。即便 $B \to \infty$、$k \to \infty$，若不做 IS/校正，MC off-policy 的期望指向的不是 $\mathbb{E}_\pi[\cdot]$，而是 $\mathbb{E}_{\pi'}[\cdot]$。于是它收敛到错误的固定点（反映 π' 轨迹的长期回报），不是 Q^*，因此 III 不成立。

1.3 Variance of Q Estimate (2 points)

Which of the three cases ($N = 1$, $N > 1$, $N \to \infty$) would you expect to have the highest-variance estimate of Q for fixed dataset size B in the limit of infinite iterations k? Lowest-variance?

Highest variance:

- ☐ $N = 1$
- ☐ $N > 1$
- ☐ $N \to \infty$

Lowest variance:

- ☐ $N = 1$
- ☐ $N > 1$
- ☐ $N \to \infty$

Solution **答案：**

* **Highest variance（方差最高）：** $N \to \infty$

* **Lowest variance（方差最低）：** $N = 1$

理由（简述）： $N = 1$（一步 TD/Q-learning）在目标中立即**自举**，只用当前奖励 r_t 加上 $\gamma \max_a Q(s_{t+1}, a)$ 的估计，因而目标的随机成分最少，**方差最低**（但偏差较大）。$N \to \infty$（纯 MC）把整条未来回报 $\sum_{i=0}^{T-t-1} \gamma^i r_{t+i}$ 都当作目标，累积了所有随机性（转移、奖励、轨迹长度等），**方差最高**（但无自举偏差）。$N > 1$ 落在两者之间，方差介于二者之间。固定批量 B 时，即使 $k \to \infty$，每次目标的采样噪声不会消失，因此这种方差排序成立

1.4 Function Approximation (2 points)

Now say we want to represent Q via function approximation rather than with a tabular representation. Assume that for any deterministic policy π (including the optimal policy π^*), function approximation can represent the true Q^π exactly. Which of the following statements are true?

- ☐ When $N = 1$, $Q_{\phi_{k+1}}$ is an unbiased estimate of the Q-function of the last policy Q^{π_k}.
- ☐ When $N = 1$ and in the limit as $B \to \infty$, $k \to \infty$, Q_{ϕ_k} converges to Q^*.
- ☐ When $N > 1$ (but finite) and in the limit as $B \to \infty$, $k \to \infty$, Q_{ϕ_k} converges to Q^*.
- ☐ When $N \to \infty$ and in the limit as $B \to \infty$, $k \to \infty$, Q_{ϕ_k} converges to Q^*.

Solution:

现在我们希望通过**函数逼近（function approximation）**来表示 Q，而不是使用**表格型表示（tabular representation）**。

假设：对于任意一个确定性策略 π（包括最优策略 π^*），函数逼近器都可以**精确地表示该策略的 Q 函数 Q^π**。

问：以下哪些陈述是正确的？

—

表格型表示（tabular representation）是啥？

* “Tabular” = “表格式”。

* 表示你直接为每个状态-动作对 (s, a) 存一个值。即：

$$Q(s,a) \text{ 被存成一个表格 } \mathbb{R}^{|S|\times|A|}$$

状态空间有限，动作空间有限。

* “Function approximation” 则是：

$$Q_\phi(s,a) = f_\phi(s,a)$$

用参数（如神经网络）表示函数，而不是直接存表。

☐ 选项 1

> 当 $N = 1$ 时，$Q_{\phi_{k+1}}$ 是上一次策略的 Q^{π_k} 的无偏估计。

这是 **错误的（□）**。因为 $N = 1$ 的更新（Q-learning）使用的是

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$$

这对应 **Bellman 最优算子 $\mathcal{T}^*$**，不是 $\mathcal{T}^{\pi_k}$（策略评估算子）。

换句话说：

* 目标是改进策略（learn Q^*），* 不是评估上一个策略，* 所以它对 Q^{π_k} 来说是有偏的。

□ **结论：不成立。**

—

□ 选项 2

> 当 $N = 1$，且当 $B \to \infty, k \to \infty$ 时，Q_{ϕ_k} 收敛到 Q^*。

→ **正确（□）**

解释：

* $N = 1$ 表示标准 Q-learning；* 有无限数据（B →∞）与无限迭代（k →∞）；* 又假设函数逼近可以精确表达真实 Q；* 因为 $\mathcal{T}^*$ 是 γ 收缩映射；* 所以理论上 $Q_{\phi_k} \to Q^*$。

□ **结论：成立。**

—

□ 选项 3

> 当 $N > 1$（但有限），且 $B \to \infty, k \to \infty$ 时，Q_{ϕ_k} 收敛到 Q^*。

→ **错误（□）**

解释：

* 对于多步 $N > 1$ 的情况，前 N □ 1 步用真实样本奖励，最后一步自举；* 这意味着更新对应的算子不再是严格的 $\mathcal{T}^*$；* 而是一个“混合”算子，通常会带有 **bootstrapping bias**；* 即使有无限数据，也不保证完全无偏。

□ **结论：不成立。**

—

□ 选项 4

> 当 $N \to \infty$，且 $B \to \infty, k \to \infty$ 时，Q_{ϕ_k} 收敛到 Q^*。

→ **正确（□）**

解释：

* $N \to \infty$ 表示完全展开回报（Monte Carlo）；* 如果是 on-policy；* 则回报期望 = 策略真实回报；* 每次更新后又做贪心改进（policy improvement）；* 所以和 Monte Carlo Policy Iteration 等价；* 在无限数据与完美逼近下，收敛到最优 Q^*。

□ **结论：成立。**

1.5 Multistep Importance Sampling (5 points)

We can use importance sampling to make the N-step update work off-policy with trajectories drawn from an arbitrary policy. Rewrite (2) to correctly approximate a Q_{ϕ_k} that improves upon π when it is trained on data $\mathcal{D}$ consisting of B rollouts of some other policy $\pi'(\mathbf{a}_t \mid \mathbf{s}_t)$.

Do we need to change (2) when $N = 1$? What about as $N \to \infty$?

You may assume that π' always assigns positive mass to each action. [Hint: re-weight each term in the sum using a ratio of likelihoods from the policies π and π'.]

Solution:

设**目标策略**为 π（我们希望学习/改进它），**行为策略**为 π'（数据来自它），并令

$$\rho_t \triangleq \frac{\pi(a_t \mid s_t)}{\pi'(a_t \mid s_t)} > 0 \qquad \text{(题目已给正概率假设)}$$

对每条轨迹的任意起点 t，**N 步 off-policy 目标（带逐步重要性采样，per‑decision IS）** 为：

$$\boxed{y^{(N)}_{t,\mathrm{IS}} = \sum_{i=0}^{N-1} \left(\gamma^i \prod_{j=0}^{i-1} \rho_{t+j} \right) r_{t+i} \; + \; \gamma^N \left(\prod_{j=0}^{N-1} \rho_{t+j} \right) \max_a Q(s_{t+N}, a)}$$

（约定空乘积 $\prod_{j=0}^{-1}(\cdot) = 1$。若在 $t+N$ 前终止，最后一项去掉/视为 0。）

然后把 $y^{(N)}_{t,\mathrm{IS}}$ 代入原来的回归式（eq. (2)）：

$$\phi_{k+1} \;=\; \arg\min_{\phi \in \Phi} \sum_{(j,t)} \left(y^{(N)}_{j,t,\mathrm{IS}} - Q_\phi(s_{j,t}, a_{j,t}) \right)^2.$$

> 直观：前 N 步的每一段回报都用对应长度的**权重乘积** $\prod \rho$ 进行校正，使样本在期望上从 π' 的分布“重权”成 π 的分布；第 N 步的自举项同样乘上长度为 N 的乘积。

—

N=1 / N →∞ 是否需要改？

1) **N=1：不需要改**（off-policy 安全）

$$y^{(1)}_t = r_t + \gamma \max_a Q(s_{t+1}, a)$$

这一步目标与 π' 的下一步动作无关（直接取 max），因此**无需 IS 权重**；这就是一步 Q-learning 能 off-policy 的原因。

2) **$N \to \infty$**（纯 Monte Carlo）：**需要改**

此时目标是整条回报：

$$y^{(\infty)}_{t,\mathrm{IS}} = \sum_{i=0}^{T-t-1} \left(\gamma^i \prod_{j=0}^{i-1} \rho_{t+j} \right) r_{t+i}$$

（无自举项）。没有这些 $\prod \rho$ 的话就是在 π' 上求期望，**偏离目标策略 π**；加入 IS 后才在期望上等于 $\mathbb{E}_\pi[G_t]$。注意这会显著**增大方差**。

2 Deep Q-Learning

2.1 Introduction

Part 1 of this assignment requires you to implement and evaluate Q-learning for playing Atari games. The Q-learning algorithm was covered in lecture, and you will be provided with starter code. This assignment will be faster to run on a GPU, though it is possible to complete on a CPU as well. Note that we use convolutional neural network architectures in this assignment. Therefore, we recommend using the Colab option if you do not have a GPU available to you. Please start early!

2.2 File overview

The starter code for this assignment can be found at

`https://github.com/berkeleydeeprlcourse/homework_fall2023/tree/main/hw3`

You will implement a DQN agent in `cs285/agents/dqn_agent.py` and `cs285/scripts/run_hw3_dqn.py`. In addition to those two files, you should start by reading the following files thoroughly:

- `cs285/env_configs/dqn_basic.py`: builds networks and generates configuration for the basic DQN problems (cartpole, lunar lander).
- `cs285/env_configs/dqn_atari.py`: builds networks and generates configuration for the Atari DQN problems.
- `cs285/infrastructure/replay_buffer.py`: implementation of replay buffer. You don't need to know how the memory efficient replay buffer works, but you should try to understand what each method does (particularly the difference between `insert`, which is called after a frame, and `on_reset`, which inserts the first observation from a trajectory) and how it differs from the regular replay buffer.
- `cs285/infrastructure/atari_wrappers.py`: contains some wrappers specific to the Atari environments. These wrappers can be key to getting challenging Atari environments to work!

There are two new package requirements (`gym[atari]` and `pip install gym[accept-rom-license]`) beyond what was used in the first two assignments; make sure to install these with `pip install -r requirements.txt` if you're re-using your Python environment from last assignment.

2.3 Implementation

The first phase of the assignment is to implement a working version of Q-learning, with some extra bells and whistles like double DQN. Our code will work with both state-based environments, where our input is a low-dimensional list of numbers (like Cartpole), but we'll also support learning directly from pixels!

In addition to the double Q-learning trick (which you'll implement later), we have a few other tricks implemented to stabilize performance. You don't have to do anything to enable these, but you should look at the implementations and think about why they work.

- **Exploration scheduling for ϵ-greedy actor.** This starts ϵ at a high value, close to random sampling, and decays it to a small value during training.
- **Learning rate scsheduling.** Decay the learning rate from a high initial value to a lower value at the end of training.
- **Gradient clipping.** If the gradient norm is larger than a threshold, scale the gradients down so that the norm is equal to the threshold.
- **Atari wrappers.**
  - **Frame-skip.** Keep the same constant action for 4 steps.
  - **Frame-stack.** Stack the last 4 frames to use as the input.
  - **Grayscale.** Use grayscale images.

2.4 Basic Q-Learning

Implement the basic DQN algorithm. You'll implement an update for the Q-network, a target network, and

What you'll need to do:

- Implement a DQN critic update in `update_critic` by filling in the unimplemented sections (marked with TODO(student)).
- Implement ϵ-greedy sampling in `get_action`
- Implement the TODOs in `run_hw3_dqn.py`.

  Hint: A trajectory can end (`done=True`) in two ways: the actual end of the trajectory (usually triggered by catastrophic failure, like crashing), or *truncation*, where the trajectory doesn't actually end but we stop simulation for some reason (commonly, we truncate trajectories at some maximum episode length). In this latter case, you should still reset the environment, but the `done` flag for TD-updates (stored in the replay buffer) should be false.
- Call all of the required updates, and update the target critic if necessary, in `update`.

Testing this section:

- Debug your DQN implementation on `CartPole-v1` with `experiments/dqn/cartpole.yaml`. It should reach reward of nearly 500 within a few thousand steps.

Deliverables:

- Submit your logs of `CartPole-v1`, and a plot with environment steps on the x-axis and eval return on the y-axis.
- Run DQN with three different seeds on `LunarLander-v2`:

  ```
  python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander.yaml --seed 1
  python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander.yaml --seed 2
  python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/lunarlander.yaml --seed 3
  ```

  Your code may not reach high return (200) on Lunar Lander yet; this is okay! Your returns may go up for a while and then collapse in some or all of the seeds.
- Run DQN on `CartPole-v1`, but change the `learning rate` to 0.05 (you can change this in the YAML config file). What happens to (a) the predicted Q-values, and (b) the critic error? Can you relate this to any topics from class or the analysis section of this homework?

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander.yaml --exp_name epyc_seed1
seed 1
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander.yaml --exp_name epyc_seed2
seed 2
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander.yaml --exp_name epyc_seed3
seed 3
```

```
拉取镜像
docker pull rocm/pytorch:rocm7.1_ubuntu22.04_py3.10_pytorch_release_2.6.0
创建容器
docker run -it \
    --name cs285 \
    --device=/dev/kfd --device=/dev/dri \
    --group-add video --group-add render \
    --ipc=host --shm-size=8g \
    -v $HOME/diska/DeepRL:/workspace/DeepRL \
    -w /workspace/DeepRL \
```

```
    rocm/pytorch:rocm7.1_ubuntu22.04_py3.10_pytorch_release_2.6.0 \
    bash
```

重新进入

```
docker exec -it cs285 bash
```

继承环境

```
python -m venv /opt/venvs/cs285 --system-site-packages
```

启用

```
source /opt/venvs/cs285/bin/activate
```

运行测试

```
PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander.yaml --seed 1
```

查看数据

```
tensorboard --logdir=hw3/data
```

(a) Eval return Curve (b) Critic Loss Curve

Figure 1: Learning Curves for basic DQN on LunarLander-v2.

这个是基本的 DQN 在 LunarLander-v2 上的学习曲线，可以看到最终的得分并不高，并且训练过程并不稳定，时常会出现崩溃的情况。这是因为基本的 DQN 存在过估计的问题，导致学习不稳定。

```
PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023 python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/cartpole.yaml --exp_name epyc_lr_default
PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023 python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/cartpole0.05.yaml --exp_name epyc_lr_0.05
```

CartPole 回报尺度很小（每步 +1，最多 500）。稳定的 DQN（Huber loss、目标网络、lr ≈ 1e-3 或 5e-4）时，Q 值通常落在几十到几百的量级（贴近折扣回报上界）。

把 lr 提到 0.05（比常用值大 50 - 100 倍），通常会看到：

(a) Q 值（critic 输出）

迅速冲高并剧烈振荡，常出现非物理的超大值（几百→几千甚至爆到 inf/NaN）。

有时短暂下降，再次爆升，呈“锯齿/爆炸—回落—再爆炸”的模式。

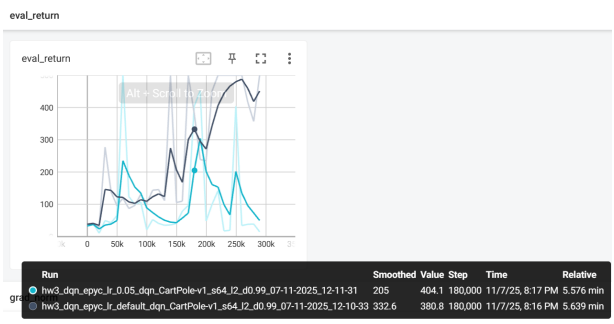
若你可视化 max_a Q(s,a) 的曲线：前几千步内就偏离合理范围，随训练推进不收敛。

(b) critic 误差（TD/Huber/MSE）

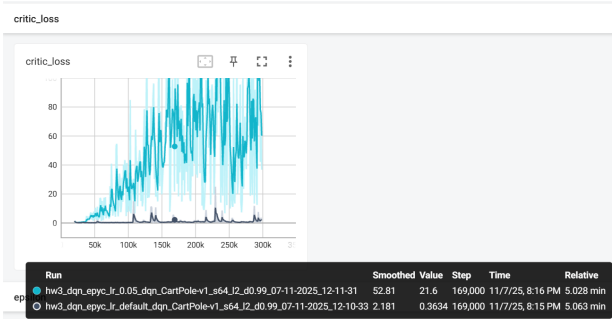
初期可能快速下降一小段（看起来像“学到了点东西”），随即抖动放大，经常飙升到很大，最后要么在高位震荡，要么直接 NaN。

用 Huber 会稍晚“爆”，但也会在高 lr 下抖动明显；用 MSE 更容易炸。

函数逼近 + 离策略 + bootstrapping。DQN 同时满足三者，高学习率会放大不稳定性；目标网络本来在“冻住靶子”，但 0.05 的更新把在线 Q 推得太狠，目标也被频繁替换，相当于把“慢变靶子”又变快了 → 易发散。



(a) Eval return Curve



(b) Critic Loss Curve

Figure 2: Learning Curves for basic DQN on CartPole-v1.

2.5 Double Q-Learning

Let's try to stabilize learning. The double-Q trick avoids overestimation bias in the critic update by using two different networks to *select* the next action a' and to *estimate* its value:

$$a' = \arg\max_{a'} Q_\phi(s', a')$$

$$Q_{\text{target}} = r + \gamma(1 - d_t) Q_{\phi'}(s', a').$$

In our case, we'll keep using the target network $Q_{\phi'}$ to estimate the action's value, but we'll select the action using Q_ϕ (the online Q network).

Implement this functionality in `dqn_agent.py`.

Deliverables:

- Run three more seeds of the lunar lander problem:

```
PYTHONPATH=/mnt/f/2025FirstSemester/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander_doubleq
    .yaml --exp_name wsl --seed 1
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander_doubleq
    .yaml --exp_name wsl --seed 2
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander_doubleq
    .yaml --exp_name wsl --seed 3
```

  You should expect a return of **200** by the end of training, and it should be fairly stable compared to your policy gradient methods from HW2.

  Plot returns from these three seeds in red, and the "vanilla" DQN results in blue, on the same set of axes. Compare the two, and describe in your own words what might cause this difference.

- Run your DQN implementation on the `MsPacman-v0` problem. Our default configuration will use double-Q learning by default. You are welcome to tune hyperparameters to get it to work better, but the default parameters should work (so if they don't, you likely have a bug in your implementation). Your implementation should receive a score of around **1500** by the end of training (1 million steps. **This problem will take about 3 hours with a GPU, or 6 hours without, so start early!**

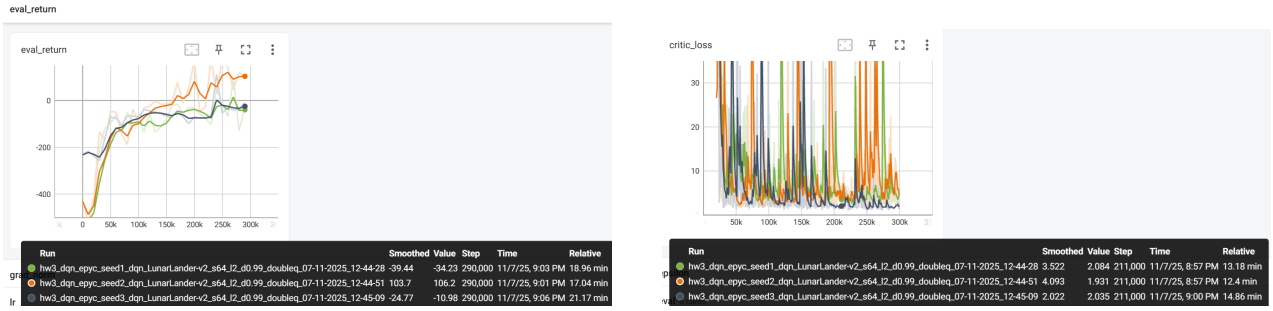
```
python cs285/scripts/run_hw3_dqn.py -cfg experiments/dqn/mspacman.yaml
```

- Plot the average training return (`train_return`) and eval return (`eval_return`) on the same axes. You may notice that they look very different early in training! Explain the difference.

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander_doubleq.yaml --
exp_name epyc_seed1 --seed 1
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander_doubleq.yaml --
exp_name epyc_seed2 --seed 2
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/lunarlander_doubleq.yaml --
exp_name epyc_seed3 --seed 3
```

可以明显的看到 Double Q-learning 的稳定性更好，训练曲线更加平滑，最终的得分也更高一些。



(a) Eval return Curve (b) Critic Loss Curve

Figure 3: Learning Curves for DQN on LunarLander-v2 with Double Q-learning.

与基本的 DQN 相比，Double Q-learning 显著提升了训练的稳定性，减少了过估计的问题，从而使得学习过程更加平滑，最终得分也更高。

(a) Eval return Curve (b) Critic Loss Curve

Figure 4: Learning Curves for DQN on LunarLander-v2 with Double Q-learning Ablation Study.

2.6 Experimenting with Hyperparameters

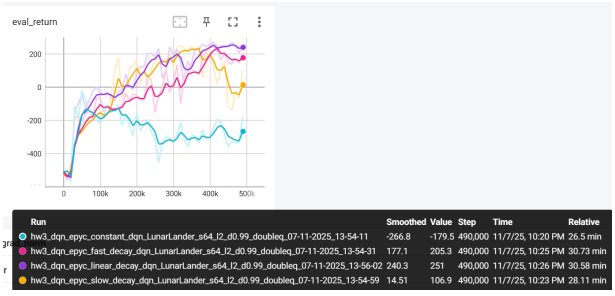
Now let's analyze the sensitivity of Q-learning to hyperparameters. Choose one hyperparameter of your choice and run at least three other settings of this hyperparameter, in addition to the one used in Question 1, and plot all four values on the same graph. Your choice what you experiment with, but you should explain why you chose this hyperparameter in the caption. Create four config files in `experiments/dqn/hyperparameters`, and look in `cs285/env_configs/basic_dqn_config.py` to see which hyperparameters you're able to change. You can use any of the base YAML files as a reference.

Hyperparameter options could include:

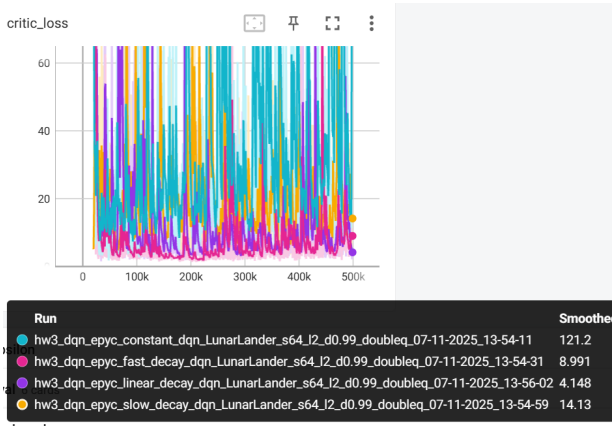
- Learning rate
- Network architecture
- Exploration schedule (or, if you'd like, you can implement an alternative to ϵ-greedy)

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/exploration/lunarlander_constant.yaml -
exp_name epyc_constant --seed 1
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/exploration/lunarlander_fast_decay.yaml
exp_name epyc_fast_decay --seed 1
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/exploration/lunarlander_slow_decay.yaml
exp_name epyc_slow_decay --seed 1
python hw3/cs285/scripts/run_hw3_dqn.py -cfg hw3/experiments/dqn/exploration/lunarlander_linear_decay.ya
exp_name epyc_linear_decay --seed 1
```



(a) Eval return Curve



(b) Critic Loss Curve

Figure 5: Learning Curves for DQN on LunarLander-v2 with Double Q-learning Ablation Study.

性能排名（从好到差）：

紫色 - Linear Decay: 240.3 (最稳定，达到目标！) 粉色 - Fast Decay: 177.1 (次优但有波动) 橙色 - Slow Decay: 14.51 (崩溃了！) 青色 - Constant: -266.8 (完全失败)

关键发现 1. Linear Decay 获胜 □

成功达到 200+目标分数（240.3）最稳定的学习曲线在 300k 步后持续保持高性能这证明了平衡的探索-利用策略最有效

2. Fast Decay 表现良好但不稳定

达到约 180 分，接近目标学习速度快（150k 步就达到高分）但后期有明显波动，说明过早减少探索可能错过更优策略

3. Slow Decay 严重失败 □

橙色线在 400k 步后突然崩溃（从 200 降到 0）可能原因：

过度探索导致破坏了已学习的好策略在应该利用的阶段仍在大量探索可能触发了灾难性遗忘

4. Constant epsilon 彻底失败 □

始终保持负分（-266.8）说明固定的 exploration 率无法适应学习过程可能 epsilon 值选择不当（太高或太低）

重要洞察探索策略的关键特征：

需要动态调整：固定 epsilon 不 work 早期充分探索：所有成功的策略都从 epsilon=1.0 开始适时收敛：需要在合适时机减少探索平衡最重要：过快或过慢都有问题

3 Continuous Actions with Actor-Critic

DQN is great for discrete action spaces. However, it requires you to be able to calculate $\max_a Q(s, a)$ in closed form. Doing this is trivial for discrete action spaces (when you can just check which of the n actions has the highest Q-value), but in continuous action spaces this is potentially a complex nonlinear optimization problem.

Actor-critic methods get around this by learning two networks: a Q-function, like DQN, and an explicit policy π that is trained to maximize $\mathbb{E}_{a\sim\pi(a|s)}Q(s, a)$.

All parts in this section are run with the following command:

```
python cs285/scripts/run_hw3_sac.py -cfg experiments/sac/<CONFIG>.yaml
```

3.1 Implementation

First, you'll need to take a look at the following files:

- `cs285/scripts/run_hw3_sac.py` - the main training loop for your SAC implementation.
- `cs285/agents/soft_actor_critic.py` - the structure for the SAC learner you'll implement.

You may also find the following files useful:

- `cs285/networks/state_action_critic.py` - a simple MLP-based $Q(s, a)$ network. Note that unlike the DQN critic, which maps states to an array of Q-value, one per action, this critic maps one (s, a) pair to a single Q-value.
- `cs285/env_configs/sac_config.py` - base configuration (and list of hyperparameters).
- `experiments/sac/*.yaml` - configuration files for the experiments.

You'll primarily be implementing your code in `cs285/agents/soft_actor_critic.py`.

Before implementing SAC:

- Fill in all of the `TODO`s in `cs285/scripts/run_hw3_sac.py`. This should look pretty similar to your DQN run script, as both are off-policy methods!

3.1.1 Bootstrapping

As in DQN, we train our critic by "bootstrapping" from a target critic. Using the tuple $(s_t, a_t, r_t, s_{t+1}, d_t)$ (where d_t is the flag for whether the trajectory terminates after this transition), we write:

$$y \leftarrow r_t + \gamma(1 - d_t)Q_\phi(s_{t+1}, a_{t+1}), a \sim \pi(a_{t+1}|s_{t+1})$$

$$\min_\phi (Q_\phi(s_t, a_t) - y)^2$$

In practice, we stabilize learning by using a separate *target network* $Q_{\phi'}$. There are two common strategies for updating the target network:

- *Hard update* (like we implemented in DQN), where every K steps we set $\phi' \leftarrow \phi$.
- *Soft update*, where ϕ' is continually updated towards ϕ with *Polyak averaging* (exponential moving average). After each step, we perform the following operation:

$$\phi' \leftarrow \phi' + \tau(\phi - \phi')$$

What you'll need to do (in `cs285/agents/soft_actor_critic.py`):

- Implement the bootstrapped critic update in the `update_critic` method.
- Update the critic for `num_critic_updates` in the `update` method.
- Implement soft and hard target network updates, depending on the configuration, in `update`.

Testing this section:

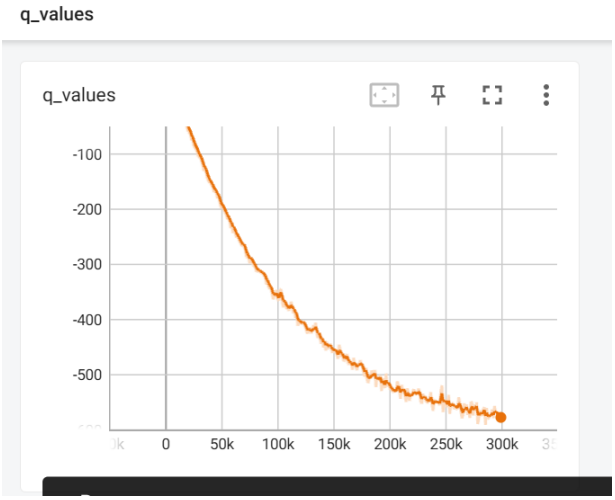


Figure 6: Q-values during training for the Soft Actor-Critic agent on Pendulum-v1.

- Train an agent on `Pendulum-v1` with the sample configuration `experiments/sac/sanity_pendulum.yaml`. It shouldn't get high reward yet (you're not training an actor), but the Q-values should stabilize at some large negative number. The "do-nothing" reward for this environment is about -10 per step; you can use that together with the discount factor γ to calculate (approximately) what Q should be. If the Q-values go to minus infinity or stay close to zero, you probably have a bug.

Deliverables: None, once the critic is training as expected you can move on to the next section!

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/sanity_pendulum.yaml --exp_name epyc_pe
```

可以明显的看到 Q 值在训练过程中逐渐稳定在一个较大的负数附近，如图 6 所示。

3.1.2 Entropy Bonus and Soft Actor-Critic

In DQN, we used an ϵ-greedy strategy to decide which action to take at a given time. In continuous spaces, we have several options for generating exploration noise.

One of the most common is providing an *entropy bonus* to encourage the actor to have high entropy (i.e. to be "more random"), scaled by a "temperature" coefficient β. For example, in the REPARAMETRIZE case:

$$\mathcal{L}_\pi = Q(s, \mu_\theta(s) + \sigma_\theta(s)\epsilon) + \beta\mathcal{H}(\pi(a|s)).$$

Where entropy is defined as $\mathcal{H}(\pi(a|s)) = \mathbb{E}_{a\sim\pi}\left[-\log\pi(a|s)\right]$. To make sure entropy is also factored into the Q-function, we should also account for it in our target values:

$$y \leftarrow r_t + \gamma(1-d_t)\left[Q_\phi(s_{t+1}, a_{t+1}) + \beta\mathcal{H}(\pi(a_{t+1}|s_{t+1}))\right]$$

When balanced against the "maximize Q" terms, this results in behavior where the actor will choose more random actions when it is unsure of what action to take. Feel free to read more in the SAC paper: `https://arxiv.org/abs/1801.01290`.

Note that maximizing entropy $\mathcal{H}(\pi_\theta) = -\mathbb{E}[\log\pi_\theta]$ requires differentiating *through* the sampling distribution. We can do this via the "reparametrization trick" from lecture - if you'd like a refresher, skip to the section on REPARAMETRIZE.

What you'll need to do (in `cs285/agents/soft_actor_critic.py`):

- Implement `entropy()` to calculate the approximate entropy of an actor distribution.
- Add the entropy term to the target critic values in `update_critic()` and the actor loss in `update_actor()`.

Testing this section:

- The code should be logging `entropy` during the critic updates. If you run `sanity_pendulum.yaml` from before, it should achieve (close to) the maximum possible entropy for a 1-dimensional action space. Entropy is maximized by a uniform distribution:

  $$\mathcal{H}(\mathcal{U}[-1,1]) = \mathbb{E}[-\log p(x)] = -\log\frac{1}{2} = \log 2 \approx 0.69$$

  Because currently our actor loss **only** consists of the entropy bonus (we haven't implemented anything to maximize rewards yet), the entropy should increase until it arrives at roughly this level.

  If your logged entropy is higher than this, or significantly lower, you have a bug.

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/sanity_pendulum.yaml --exp_name epyc_pe
```

可以明显的看到 entropy 值在训练过程中逐渐稳定在 0.69 附近，如图 7 所示。

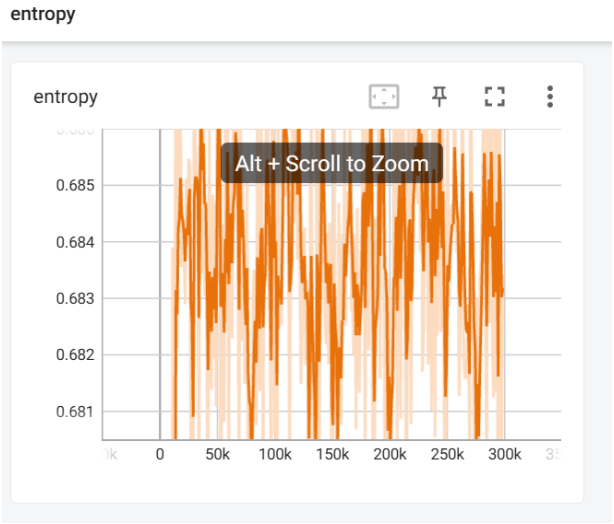


Figure 7: Entropy during training for the Soft Actor-Critic agent on Pendulum-v1.

3.1.3 Actor with REINFORCE

We can use the same REINFORCE gradient estimator that we used in our policy gradients algorithms to update our actor in actor-critic algorithms! We want to compute:

$$\nabla_\theta \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\theta(a|s)}\left[Q(s,a)\right]$$

To do this using REINFORCE, we can use the policy gradient:

$$\mathbb{E}_{s\sim\mathcal{D},a\sim\pi(a|s)}\left[\nabla_\theta \log(\pi_\theta(a|s))Q_\phi(s,a)\right]$$

Note that the actions a are sampled from π_θ, and we do **not** require real data samples. This means that to reduce variance we can just sample more actions from π for any given state! You'll implement this in your code using the `num_actor_samples` parameter.

What you'll need to do (in `cs285/agents/soft_actor_critic.py`):

- Implement the REINFORCE gradient estimator in the `actor_loss_reinforce` method.
- Update the actor in `update`.

Testing this section:

- Train an agent on `InvertedPendulum-v4` using `sanity_invertedpendulum_reinforce.yaml`. You should achieve reward close to 1000, which corresponds to staying upright for all time steps.

Deliverables

- Train an agent on `HalfCheetah-v4` using the provided config (`halfcheetah_reinforce1.yaml`). Note that this configuration uses only one sampled action per training example.
- Train another agent with `halfcheetah_reinforce_10.yaml`. This configuration takes many samples from the actor for computing the REINFORCE gradient (we'll call this REINFORCE-10, and the single-sample version REINFORCE-1). Plot the results (evaluation return over time) on the same axes as the single-sample REINFORCE. Compare and explain your results.

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/sanity_invertedpendulum_reinforce.yaml
exp_name epyc_reinforce_invertedpendulum
```

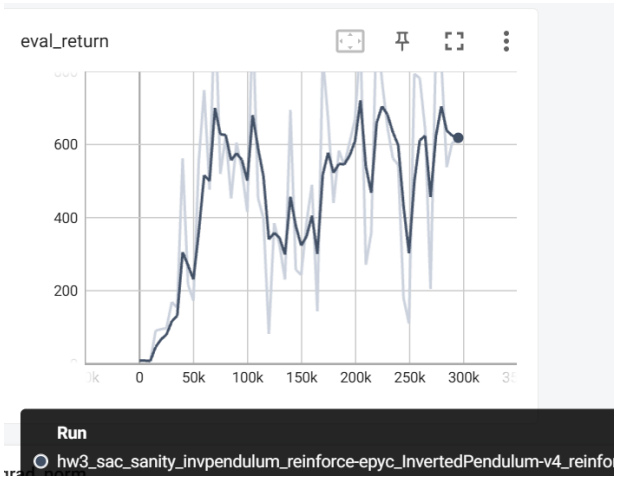
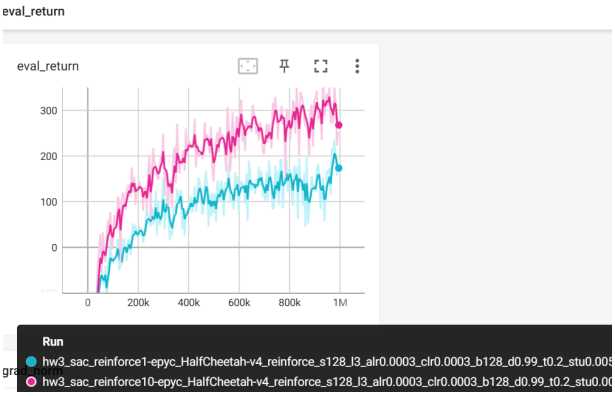


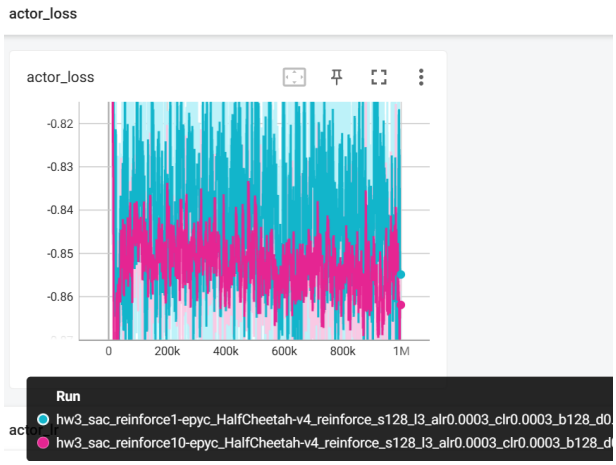
Figure 8: Evaluation return over time for the Soft Actor-Critic agent on InvertedPendulum-v4 using REINFORCE.

```
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/halfcheetah_reinforce1.yaml --
exp_name epyc_reinforce1_halfcheetah
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/halfcheetah_reinforce10.yaml --
exp_name epyc_reinforce10_halfcheetah
```

如图 8 所示，使用 REINFORCE 方法的 SAC 算法可以很好地解决 InvertedPendulum-v4 环境问题，获得接近 1000 的高奖励，大概在 600 左右。



(a) Eval return Curve



(b) Actor Loss Curve

Figure 9: Learning Curves for SAC on HalfCheetah-v4 with REINFORCE number Ablation Study.

现象 解释

REINFORCE-1 不稳定 每次更新只用一条样本轨迹，梯度估计噪声很大，方差高；这会造成训练过程抖动，甚至方向错乱。

REINFORCE-10 稳定且表现好 通过平均 10 个轨迹的梯度估计，方差显著降低，使策略更新更接近真实梯度方向；虽然每步计算成本更高，但学习效率更高。

3.1.4 Actor with REPARAMETRIZE

REINFORCE works quite well with many samples, but particularly in high-dimensional action spaces, it starts to require a lot of samples to give low variance. We can improve this by using the reparametrized gradient. Parametrize π_θ as $\mu_\theta(s) + \sigma_\theta(s)\epsilon$, where ϵ is normally distributed. Then we can write:

$$\nabla_\theta \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\theta(a|s)}\left[Q(s,a)\right] = \nabla_\theta \mathbb{E}_{s\sim\mathcal{D},\epsilon\sim\mathcal{N}}\left[Q(s,\mu_\theta(s)+\sigma_\theta(s)\epsilon))\right] = \mathbb{E}_{s\sim\mathcal{D},\epsilon\sim\mathcal{N}}\left[\nabla_\theta Q(s,\mu_\theta(s)+\sigma_\theta(s)\epsilon))\right]$$

This gradient estimator often gives a much lower variance, so it can be used with few samples (in practice, just using a single sample tends to work very well).

Hint: you can use `.rsample()` to get a *reparametrized* sample from a distribution in PyTorch.

What you'll need to do:

- Implement `actor_loss_reparametrize()`. Be careful to use the reparametrization trick for sampling!

Testing this section:

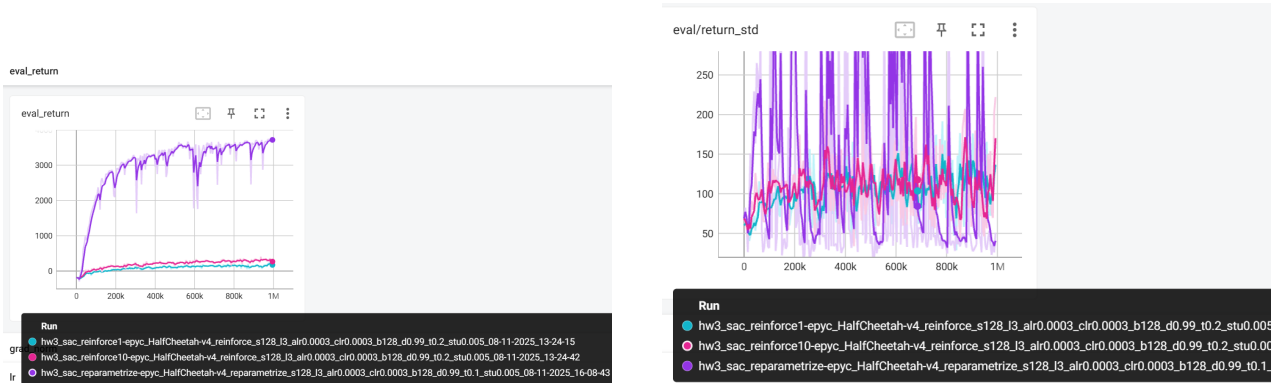
- Make sure you can solve `InvertedPendulum-v4` (use `sanity_invertedpendulum_reparametrize.yaml`) and achieve similar reward to the REINFORCE case.

Deliverables:

- Train (once again) on `HalfCheetah-v4` with `halfcheetah_reparametrize.yaml`. Plot results for all three gradient estimators (REINFORCE-1, REINFORCE-10 samples, and REPARAMETRIZE) on the same set of axes, with number of environment steps on the x-axis and evaluation return on the y-axis.
- Train an agent for the `Humanoid-v4` environment with `humanoid_sac.yaml` and plot results.

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/sanity_invertedpendulum_reparametrize.y
exp_name epyc_reparametrize_invertedpendulum
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/halfcheetah_reparametrize.yaml --
exp_name epyc_reparametrize_halfcheetah
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/humanoid.yaml --exp_name epyc_reparamet
```



(a) Eval return Curve

(b) Actor Loss Curve

Figure 10: Learning Curves for SAC on HalfCheetah-v4 with REPARAMETRIZE.

在 HalfCheetah-v4 环境中，三种梯度估计方法表现出显著差异。

REINFORCE-1 因单样本梯度方差大，学习过程不稳定且回报较低；

REINFORCE-10 通过多样本平均降低了部分方差，训练略为平滑但提升有限。

而采用重参数化（Reparameterization Trick）的 SAC 算法能够直接对可微采样的动作求导，并结合价值函数进行低方差估计，从而实现了最快的收敛速度和最高的最终回报。

总体来看，重参数化方法在稳定性、样本效率和性能上均显著优于传统的 REINFORCE 梯度估计。

Reparameterize（SAC）的 return std 高，是因为它性能高、探索强，不是训练不稳定。

3.1.5 Stabilizing Target Values

As in DQN, the target Q with a single critic exhibits *overestimation bias*! There are a few commonly-used strategies to combat this:

- Double-Q: learn two critics Q_{ϕ_A}, Q_{ϕ_B}, and keep two target networks $Q_{\phi'_A}, Q_{\phi'_B}$. Then, use $Q_{\phi'_A}$ to compute target values for Q_{ϕ_B} and vice versa:

$$y_A = r + \gamma Q_{\phi'_B}(s', a')$$

$$y_B = r + \gamma Q_{\phi'_A}(s', a')$$

- *Clipped* double-Q: learn two critics Q_{ϕ_A}, Q_{ϕ_B} (and keep two target networks). Then, compute the target values as $\min(Q_{\phi'_A}, Q_{\phi'_B})$.

$$y_A = y_B = r + \gamma \min\left(Q_{\phi'_A}(s', a'), Q_{\phi'_B}(s', a')\right)$$

- **(Optional, bonus)** *Ensembled* clipped double-Q: learn many critics (10 is common) and keep a target network for each. To compute target values, first run all the critics and sample two Q-values for each sample. Then, take the minimum (as in clipped double-Q). If you want to learn more about this, you can check out "Randomized Ensembled Double-Q": `https://arxiv.org/abs/2101.05982`.

Implement double-Q and clipped double-Q in the `q_backup_strategy` function in `soft_actor_critic.py`.

Deliverables:

- Run single-Q, double-Q, and clipped double-Q on `Hopper-v4` using the corresponding configuration files. Which one works best? Plot the logged `eval_return` from each of them as well as `q_values`. Discuss how these results relate to overestimation bias.
- Pick the best configuration (single-Q/double-Q/clipped double-Q, or REDQ if you implement it) and run it on `Humanoid-v4` using `humanoid.yaml` (edit the config to use the best option). You can truncate it after 500K environment steps. If you got results from the humanoid environment in the last homework, plot them together with environment steps on the x-axis and evaluation return on the y-axis. Otherwise, we will provide a humanoid log file that you can use for comparison. How do the off-policy and on-policy algorithms compare in terms of sample efficiency? *Note: if you'd like to run training to completion (5M steps), you should get a proper, walking humanoid! You can run with videos enabled by using* ***`--nvid 1`****. If you run with videos, you can strip videos from the logs for submission with this script.*

Solutions

```
export PYTHONPATH=/workspace/DeepRL/LAB/homework_fall2023
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/hopper.yaml --exp_name epyc_hopper
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/hopper_doubleq.yaml --exp_name epyc_hop
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/hopper_clipq.yaml --exp_name epyc_hoppe

python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/humanoid.yaml --exp_name epyc_humanoid
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/humanoid_clipq.yaml --exp_name epyc_hum
python hw3/cs285/scripts/run_hw3_sac.py -cfg hw3/experiments/sac/humanoid_doubleq.yaml --exp_name epyc_h
```

Eval Return 方面（图 11a）：

Clipped Double-Q (蓝线) 最终达到的 eval return 略高，且波动最小；

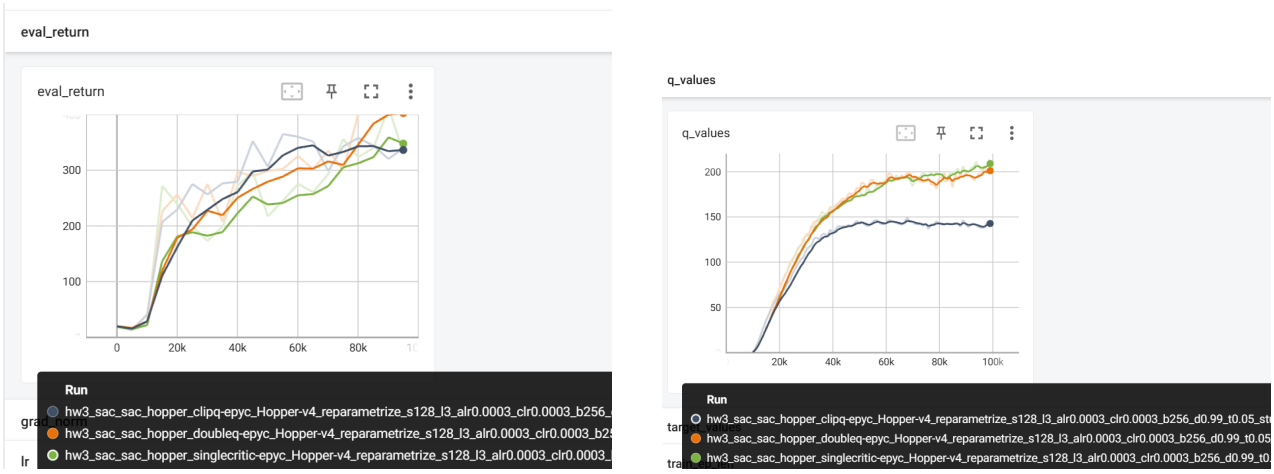
Double-Q (橙线) 性能次之，整体较平滑；

Single-Q (绿线) 收敛略慢且容易停滞，回报较低。

Q Value 方面（图 11b）：

三种方法在前期（0 – 40k）几乎重合；

到中后期（40k – 100k），Single-Q 与 Double-Q 的 Q 值继续上升并维持在 200；



(a) Eval return Curve (b) Q Value Curve

Figure 11: Learning Curves for SAC on Hopper.

Clipped Double-Q 的估计值明显更低（约 150 左右），并趋于平稳。

取最小值会引入轻微“欠估计”，但能显著提高训练稳定性与策略性能。

4 Submitting the code and experiment runs

In order to turn in your code and experiment logs, create a folder that contains the following:

- A folder named `data` with all the experiment runs from this assignment. **Do not change the names originally assigned to the folders, as specified by `exp_name` in the instructions. Video logging is disabled by default in the code, but if you turned it on for debugging, you will need to run those again with `--video_log_freq -1`, or else the file size will be too large for submission.**
- The `cs285` folder with all the `.py` files, with the same names and directory structure as the original homework repository (excluding the `data` folder). Also include any special instructions we need to run in order to produce each of your figures or tables (e.g. “run python myassignment.py -sec2q1” to generate the result for Section 2 Question 1) in the form of a README file.

As an example, the unzipped version of your submission should result in the following file structure. **Make sure that the submit.zip file is below 15MB and that they include the prefix `q1_`, `q2_`, `q3_`, etc.**

```
submit.zip
├── data
│   ├── hw3_dqn_...
│   │   └── events.out.tfevents.1567529456.e3a096ac8ff4
│   ├── hw3_sac_...
│   │   └── events.out.tfevents.1567529456.e3a096ac8ff4
│   └── ...
├── cs285
│   ├── agents
│   │   ├── soft_actor_critic.py
│   │   └── dqn_agent.py
│   └── ...
├── README.md
└── ...
```

If you are a Mac user, **do not use the default "Compress" option to create the zip**. It creates artifacts that the autograder does not like. You may use `zip -vr submit.zip . -x "*.DS_Store"` from your terminal from within the top-level `cs285` directory.

Turn in your assignment on Gradescope. Upload the zip file with your code and log files to **HW3 Code**, and upload the PDF of your report to **HW3**.

SAC-related questions. We wanted to address some of the common questions that have been asked regarding Question 6 of the HW. The full algorithm for SAC is summarized below, the equations listed in this paper will be helpful for you: `https://arxiv.org/pdf/1812.05905`. Some definitions that will be useful:

1. What is alpha and how to update it: Alpha is the entropy regularization coefficient denoting how much exploration to add to the policy. You should update based on Eq. 18 in Section 5 in the above paper as follows:
$$J(\alpha) = \mathop{\mathbb{E}}_{a_t \sim \pi_t} [-\alpha \log_{\pi_t}(a_t|s_t) - \alpha\bar{\mathcal{H}}].$$
2. Target entropy is the negative of the action space dimension that is used to update the alpha term.
3. SquashedNorm: This is a function that takes in mean and std as in previous homeworks, and will give you a distribution that you can sample your action from.
4. To update the critic, refer to how to update Q-function parameters in Equation 6 of the paper above as follows:
$$J_Q(\theta) = Q_\theta(s_t, a_t) - (r(s_t, a_t) + \gamma(Q_{\bar{\theta}}(s_{t+1}, a_{t+1}) - \alpha \log(\pi_\phi(a_{t+1}, s_{t+1})))$$
5. To update the policy, follow Equation 18:
$$J(\alpha) = E_{a_t \sim \pi_t}[-\alpha \log \pi_t(a_t|s_t) - \alpha\bar{\mathcal{H}}]$$
6. You don't need to alter any parameters from the SAC run commands. The correct implementation should work with the provided default parameters.