Homework 4

6771 Reinforcement Learning (Fall 2025)

Submission deadline: 11:59pm, December 14
(**hard deadline**: December 17)

The latex template is here.

1 Upper Confidence Bound Value Iteration (UCBVI)

Consider a fixed-horizon, finite-state, finite-action MDP with state space $\mathcal{S}$, action set $\mathcal{A}$, horizon H, reward function $R : \mathcal{S} \times \mathcal{A} \to [0, 1]$, transition $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, and initial state distribution ρ. Let $S = |\mathcal{S}|$ and $A = |\mathcal{A}|$ denote the size of the state and action space, respectively.

Interaction protocol: In each episode $t = 1, 2, \ldots, T$, the learner uses policy $\pi^t = (\pi_1^t, \pi_2^t, \ldots, \pi_H^t)$ to interact with the MDP for H steps, generating trajectory $(s_1^t, a_1^t, r_1^t, \ldots, s_H^t, a_H^t, r_H^t)$.

Let $N_t(s, a)$ be the number of times the learner visits (s, a) before episode t. Let $\widehat{R}_t$ and $\widehat{P}_t$ be empirical estimates of R and P using the collected data before episode t. Consider the UCBVI algorithm:

Algorithm 1 UCBVI

1 **Parameter**: $\delta \in (0, 1)$.
2 **for** $t = 1, 2, \ldots, T$ **do**
3 Define $\xi_t(s, a) = 4H\sqrt{\frac{S \log(1/\delta)}{N_t(s,a)+1}}$.
4 `// Perform value iteration with bonus`
5 Define $\widetilde{V}_{H+1}^t(s) = 0$ for all s.
6 **for** $h = H, \ldots, 2, 1$ **do**

$$\widetilde{Q}_h^t(s, a) = \widehat{R}_t(s, a) + \xi_t(s, a) + \mathbb{E}_{s' \sim \widehat{P}_t(\cdot|s,a)} \left[\widetilde{V}_{h+1}^t(s')\right]$$
$$\widetilde{V}_h^t(s) = \min\left\{\max_a \widetilde{Q}_h^t(s, a), H\right\}$$

7 `// Interact with the MDP`
for $h = 1, 2, \ldots, H$ **do**
Observe s_h^t, and choose $a_h^t = \operatorname{argmax}_a \widetilde{Q}_h^t(s_h^t, a)$.

This is a standard value iteration algorithm with $\xi_t(s, a)$ defined in Line 3 as the exploration bonus. Below, we analyze the regret of this algorithm, defined as

$$\text{Regret} = \sum_{t=1}^{T} \left(\mathbb{E}_{s\sim\rho}\left[V_1^{\pi^\star}(s)\right] - \mathbb{E}_{s\sim\rho}\left[V_1^{\pi^t}(s)\right]\right).$$

Our analysis relies on the following concentration inequality, which we will directly use without proving it.

Lemma 1. *For any s, a, t, with probability at least $1 - \delta$, $\left|\widehat{R}_t(s, a) - R(s, a)\right| + H \left\|\widehat{P}_t(\cdot|s, a) - P(\cdot|s, a)\right\|_1 \le \xi_t(s, a)$.*

Furthermore, we generalize the standard V^π function as the following:

Definition 2. *For any sequence of functions $f_1(s,a), f_2(s,a), \ldots, f_H(s,a)$, define*

$$V_h^\pi(s; f_{1:H}) = \mathbb{E}\left[\sum_{k=h}^{H} f_k(s_k, a_k) \,\middle|\, a_k = \pi_k(s_k) \text{ and } s_{k+1} \sim P(\cdot|s_k, a_k) \text{ for } k \geq h\right].$$

When $f_1 = f_2 = \cdots = f_H$, we write it as $V_h^\pi(s; f)$.

Definition 2 recovers the standard V^π function when f_h's are all equal to R, the reward function of the MDP. In that case, we simply denote $V_h^\pi(s) = V_h^\pi(s; R)$.

(a) (5%) Assume the inequality in Lemma 1 holds for all s, a. Prove that $\widetilde{V}_h^t(s) \geq V_h^\star(s)$ for all h and s, where $\widetilde{V}_h^t(s)$ is defined in Algorithm 1.
Hint: Recall the Bellman equations $Q_h^\star(s,a) = R(s,a) + \mathbb{E}_{s' \sim P(\cdot|s,a)}[V_{h+1}^\star(s')]$ and $V_h^\star(s) = \max_a Q_h^\star(s,a)$. Prove the desired inequality by induction from $h = H$ to $h = 1$.

(b) (10%) Assume the inequality in Lemma 1 holds for all s, a. Prove that $\widetilde{V}_h^t(s) \leq V_h^{\pi^t}(s; R + 2\xi_t)$ for all h and s.
Hint: First, find $f_h(s,a)$ such that $\widetilde{Q}_h^t(s,a) = f_h(s,a) + \mathbb{E}_{s' \sim P(\cdot|s,a)}[\widetilde{V}_{h+1}^t(s')]$. Argue that $\widetilde{V}_h^t(s) \leq V_h^{\pi^t}(s; f_{1:H})$. On the other hand, show that $f_h(s,a) \leq R(s,a) + 2\xi_t(s,a)$. Finally, conclude $\widetilde{V}_h^t(s) \leq V_h^{\pi^t}(s; R + 2\xi_t)$.

(c) (5%) Assume the inequality in Lemma 1 holds for all s, a. Prove that

$$\mathbb{E}_{s\sim\rho}\left[V_1^\star(s)\right] - \mathbb{E}_{s\sim\rho}\left[V_1^{\pi^t}(s)\right] \leq \mathbb{E}_{s\sim\rho}\left[V_1^{\pi^t}(s; 2\xi_t)\right].$$

Hint: Use (a) and (b) to upper bound $V_1^\star(s)$.

(d) (10%) Prove that the expected regret can be upper bounded as

$$\mathbb{E}\left[\sum_{t=1}^{T}\left(\mathbb{E}_{s\sim\rho}\left[V_1^\star(s)\right] - \mathbb{E}_{s\sim\rho}\left[V_1^{\pi^t}(s)\right]\right)\right] \leq O(\sqrt{H^3S^2AT\log(1/\delta)}) + \delta SAHT.$$

Hint: Argue that

$$\mathbb{E}_{s\sim\rho}\left[V_1^{\pi^t}(s; 2\xi_t)\right] = \mathbb{E}\left[\sum_{h=1}^{H} 8H\sqrt{\frac{S\log(1/\delta)}{N_t(s_h^t, a_h^t)+1}} \,\middle|\, (s_1^t, a_1^t, \ldots, s_H^t, a_H^t) \text{ generated from } \pi^t\right].$$

Then upper bound $\sum_{t=1}^{T}\sum_{h=1}^{H}\sqrt{\frac{1}{N_t(s_h^t,a_h^t)+1}}$ by $O(\sqrt{SAHT})$, similar to 2(c) in Homework 1.

2 Mountain Car

In this problem, we use reinforcement learning to train a Mountain Car player (discrete-action version). A starter code is **here**. A description of this environment can be found here.

In Mountain Car, the agent receives a non-zero reward only by reaching the top of the right hill. However, the agent cannot reach the right hill by simply accelerating rightward—the engine is too weak. The only way to succeed is to build momentum by swinging back and forth. Mountain Car is a classic "sparse reward" environment where the local reward signal does not provide useful information about how to reach the goal.

In this problem, we use an "exploration bonus" to help the agent explore new states (characterized by position and velocity). This encourages the agent to try new acceleration strategies to reach previously unvisited positions or velocities.

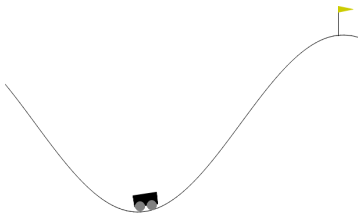

Figure 1: Mountain Car

2.1 Adding Bonus to RL Algorithms

We can apply exploration bonuses to value-iteration-based algorithms such as DQN and policy-iteration-based algorithms such as PPO. The pseudocodes are provided in Algorithm 2 and Algorithm 3, respectively. Most of the pseudocodes are the same as those in Homework 3, except for the colored parts (blue or red), which are specific to exploration bonuses. To reduce your workload, the blue parts are already implemented in the starter code. The red part, the update of the bonus function, is left as your TODO.

2.1.1 DQN

Adding an exploration bonus to DQN is straightforward: simply add the bonus $\xi(s)$ to the reward when performing the Q-network update. The pseudocode is shown in Algorithm 2. Note that the reward samples stored in the replay buffer should NOT include the bonus—see Line 18. This is because the exploration bonus is designed to decrease over time. Including the bonus in the replay buffer would cause the large bonuses from the early phase of training to continue affecting later training.

Algorithm 2 DQN with exploration bonus

8 **Parameters**: N, M, B, α (learning rate), τ (target network update rate), ϵ (exploration probability), c (bonus scale).

9 Initialize the replay buffer $D = \{\}$.
10 Initialize the online network Q_θ with weights θ.
11 Initialize the target network $Q_{\overline{\theta}}$ with weights $\overline{\theta} = \theta$.

12 Sample $s_{\text{tmp}} \sim \rho$, where ρ is the initial state distribution.
13 **for** $k = 1, \ldots$ **do**
14 $s_1 \leftarrow s_{\text{tmp}}$.
15 **for** $i = 1, 2, \ldots, N$ **do**
16 Select action $a_i = \begin{cases} \operatorname{argmax}_a Q_\theta(s_i, a) & \text{with probability } 1-\epsilon, \\ \text{random action} & \text{with probability } \epsilon. \end{cases}$
17 Observe reward r_i and the next state s'_i.
18 $D \leftarrow D \cup \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$.
19 If s'_i is a terminal state, sample $s_{i+1} \sim \rho$; otherwise, set $s_{i+1} = s'_i$.
20 $s_{\text{tmp}} \leftarrow s_{N+1}$.
21 **for** $j = 1, \ldots, M$ **do**
22 Randomly sample a minibatch $\mathcal{B} \subset \mathcal{D}$ with size $|\mathcal{B}| = B$.
23 Update the online network:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \left(Q_\theta(s,a) - r - c \cdot \xi(s) - \gamma \max_{a'} Q_{\overline{\theta}}(s', a') \right)^2$$

24 where $\xi(s)$ is the exploration bonus and $\max_{a'} Q_{\overline{\theta}}(s', a') \triangleq 0$ if s' is a terminal state.
25 Update the target network:

$$\overline{\theta} \leftarrow \tau\theta + (1-\tau)\overline{\theta}.$$

26 Update the bonus function $\xi(s)$ with $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$.

2.1.2 PPO

Adding an exploration bonus to PPO is also conceptually straightforward: the goal is to update the policy parameter θ to maximize $\frac{\pi_\theta(a_i|s_i)}{\pi_{\text{old}}(a_i|s_i)} A^{\pi_{\text{old}}}(s_i, a_i; R + \xi)$ (omitting the clipping), where $A^\pi(s, a; R + \xi)$ is the advantage function of policy π under the reward function $R(s, a) + \xi(s, a)$. Recall that when there is no bonus, we often simply write $A^\pi(s, a; R)$ as $A^\pi(s, a)$.

While it is possible to combine the original reward R and the bonus ξ to calculate a single advantage estimator, the RND paper uses separate value networks to estimate the advantage functions for the two different rewards, as do Algorithm 3 and the starter code. The RND paper provides justification for this approach in Section 2.3. Both approaches are theoretically correct, as the decomposition $A^\pi(s, a; R_1 + R_2) = A^\pi(s, a; R_1) + A^\pi(s, a; R_2)$ always holds for any two reward functions R_1 and R_2.

Algorithm 3 PPO with exploration bonus

Parameters: N, M, B, α (policy learning rate), $\alpha^\circ, \alpha^{\text{b}}$ (value learning rate), λ (GAE parameter), β (entropy bonus coefficient), ϵ (clipping parameter), c (bonus scale).

Initialize the policy network π_θ with weights θ

Initialize the value network V°_ϕ with weights ϕ.

Initialize the bonus value network V^{b}_φ with weights φ.

Sample $s_{\text{tmp}} \sim \rho$, where ρ is the initial state distribution.

for $k = 1, \ldots$ **do**

  $s_1 \leftarrow s_{\text{tmp}}$

  for $i = 1, 2, \ldots, N$ **do**

    Select action $a_i \sim \pi_\theta(\cdot|s_i)$.

    Observe reward r_i and the next state s'_i.

    If s'_i is a terminal state, sample $s_{i+1} \sim \rho$; otherwise, set $s_{i+1} = s'_i$.

  $s_{\text{tmp}} \leftarrow s_{N+1}$

  Compute $A^\circ(s_i, a_i)$ for all $i = 1, \ldots, N$ by calling $\text{GAE}_\lambda(\{s_i, a_i, r_i, s'_i\}_{i=1}^N, V^\circ_\phi)$ (Algorithm 4).

  Compute $A^{\text{b}}(s_i, a_i)$ for all $i = 1, \ldots, N$ by calling $\text{GAE}_\lambda(\{s_i, a_i, c \cdot \xi(s_i), s'_i\}_{i=1}^N, V^{\text{b}}_\varphi)$.

  Define $\hat{V}^\circ(s_i) = A^\circ(s_i, a_i) + V^\circ_\phi(s_i)$ for all $i = 1, \ldots, N$.

  Define $\hat{V}^{\text{b}}(s_i) = A^{\text{b}}(s_i, a_i) + V^{\text{b}}_\varphi(s_i)$ for all $i = 1, \ldots, N$.

  Define $A(s_i, a_i) = A^\circ(s_i, a_i) + A^{\text{b}}(s_i, a_i)$ for all $i = 1, \ldots, N$.

  Define $\hat{\pi}(a_i|s_i) = \pi_\theta(a_i|s_i)$ for all $i = 1, \ldots, N$.

  for $j = 1, \ldots, M$ **do**

    Randomly sample a minibatch $\mathcal{B} \subset \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ with size $|\mathcal{B}| = B$.

    Update the policy network:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \Big(\min\left\{ \varrho_\theta(s,a) A(s,a), \varrho^{\text{clip}}_\theta(s,a) A(s,a) \right\} \underbrace{- \beta \sum_{a'} \pi_\theta(a'|s) \log \pi_\theta(a'|s)}_{\text{entropy}} \Big),$$

    where $\varrho_\theta(s,a) \triangleq \frac{\pi_\theta(a|s)}{\hat{\pi}(a|s)}$ and $\varrho^{\text{clip}}_\theta(s,a) \triangleq \max\{1 - \epsilon, \min\{1 + \epsilon, \varrho_\theta(s,a)\}\}$.

    Update value networks:

$$\phi \leftarrow \phi - \alpha^\circ \nabla_\phi \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \left(V^\circ_\phi(s) - \hat{V}^\circ(s) \right)^2,$$

$$\varphi \leftarrow \varphi - \alpha^{\text{b}} \nabla_\varphi \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \left(V^{\text{b}}_\varphi(s) - \hat{V}^{\text{b}}(s) \right)^2.$$

  Update the bonus function $\xi(s)$ with $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$.

Algorithm 4 GAE$_\lambda$

Parameter: λ
Input: $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ and V.
Define $A(s_{N+1}, a_{N+1}) \triangleq 0$.
for $i = N, N-1, \ldots, 1$ **do**
  if *s'_i is a terminal state* **then**
    $\delta_i \leftarrow r_i - V(s_i)$
    $A(s_i, a_i) \leftarrow \delta_i$
  else
    $\delta_i \leftarrow r_i + \gamma V(s_{i+1}) - V(s_i)$
    $A(s_i, a_i) \leftarrow \delta_i + \lambda\gamma A(s_{i+1}, a_{i+1})$
return $\{A(s_i, a_i)\}_{i=1}^N$.

2.2 Bonus Construction

2.2.1 Tabular Bonus

To construct tabular bonuses, we discretize the state space into a finite number of bins. Let $g(s)$ denote the bin to which state s belongs, and let $n(g)$ denote the number of times bin g has been visited. The bonus is then defined as

$$\xi(s) = \frac{c}{\sqrt{n(g(s))}}$$

for some hyper-parameter $c > 0$. The update of $\xi(s)$ with new data $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ (Line 26 of Algorithm 2 or Line 50 of Algorithm 3) is performed as follows:

Algorithm 5 Tabular bonus update with $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$

for $i = 1, 2, \ldots, N$ **do**
  $n(g(s_i)) \leftarrow n(g(s_i)) + 1$.

In the starter code, the states are already discretized into bins. You are free to modify any aspect of the implementation, such as the number of bins. You need to implement the TODO in `get_bonus()` under the class `TabularExplorationBonus`.

2.2.2 Random Network Distillation

Random Network Distillation (RND) randomly initializes a fixed target network $f_\phi(s)$ that maps states to d-dimensional vectors, and trains a prediction network $f_\theta(s)$ to approximate it. The bonus is defined as

$$\xi(s) = c\|f_\theta(s) - f_\phi(s)\|_2^2$$

for some hyperparameter c.

The update of $\xi(s)$ with new data $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ (Line 26 of Algorithm 2 or Line 50 of Algorithm 3) is performed by minimizing the prediction error:

Algorithm 6 RND bonus update with $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$

$\theta \leftarrow \theta - \eta \frac{1}{N} \sum_{i=1}^N \nabla_\theta \|f_\theta(s_i) - f_\phi(s_i)\|_2^2$ with some learning rate η.

Since the state space may have different ranges across dimensions, we typically apply state normalization to scale each state dimension to $[-1, 1]$ before inputting to the f_θ and f_ϕ networks. You need to implement the TODO in `get_bonus()` under the class `RandomNetworkDistillation`.

2.3 Tasks

Your task is to read the starter code and implement the bonus function updates. The tasks are marked with TODO in the code. You are free to modify other parts of the code (e.g., using your own implementations of DQN and PPO from Homework 3). However, you cannot use other reward-shaping schemes, such as adding a reward to encourage the learner to reach positions with larger magnitude. The bonus should be novelty-based, i.e., encouraging the learner to reach previously unvisited states.

The code already implements DQN and PPO without bonuses. To run them, type

```
Explore.py --seed [seed] --algorithm [algorithm] --exploration [exploration] --figure [figure]
```

Allowed parameters:

- `algorithm: DQN | PPO`
- `exploration: none | tabular | rnd`

When running, the code will generate two figures in real time. The first figure tracks the per-episode return and its running average over a window of size 100, the same as in Homework 3. The second figure shows a state visitation heatmap (number of times the learner has visited each discretized state) and a bonus heatmap (current bonus value for each discretized state). The two figures will be automatically saved every 100 episodes, according to the figure name you specified in the command or a default name if no figure name is specified.

In the questions below, you will be asked to paste the two figures your code generates.

(a) (10%) Run DQN with NO bonus for 3000 episodes. Paste the return figure and the heatmap figure here.

(b) (10%) Run PPO with NO bonus for 6000 episodes. Paste the return figure and the heatmap figure here.

(c) (10%) Complete the TODO in the `TabularExplorationBonus` class and perform necessary hyperparameter tuning. Run DQN with tabular bonus. The default is 3000 episodes. You may stop before reaching 3000 episodes or run for more than 3000 episodes. Paste the return figure and the heatmap figure here.

Let T be the number of episodes you run. The score in this part will be calculated as

$$\min\left\{\frac{(\text{maximum running average within } T \text{ episodes}) + 200}{90}, 1\right\} \times \min\left\{\frac{3000}{T}, 1\right\} \times 100\%.$$

In other words, if the maximum running average reaches -110 within 3000 episodes, then you get full credit.

(d) (10%) Use the same code in (c) to run PPO with tabular bonus (default $= 6000$ episodes). Paste the return figure and the heatmap figure here.

The score in this part will be calculated as

$$\min\left\{\frac{(\text{maximum running average within } T \text{ episodes}) + 200}{90}, 1\right\} \times \min\left\{\frac{6000}{T}, 1\right\} \times 100\%.$$

(e) (10%) Complete the TODO in the `RandomNetworkDistillation` class and perform necessary hyperparameter tuning. Run DQN with RND bonus (default $= 3000$ episodes). Paste the return figure and the heatmap figure here. The score calculation is same as in (c).

(f) (10%) Use the same code in (e) to run PPO with RND bonus (default $= 6000$ episodes). Paste the return figure and the heatmap figure here. The score calculation is same as in (d).