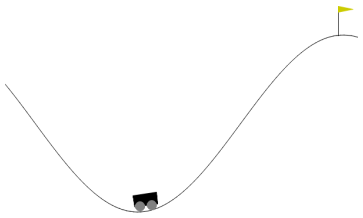
Homework 5

4771 Reinforcement Learning (Spring 2026)

Submission deadline **(hard deadline)**: 11:59pm, May 8

The latex template is here.

1 Playing Mountain Car Yourself



A description of Mountain Car can be found here. Execute this to play it yourself. Use left and right keys to control the car.

In Mountain Car, in each step, the agent can choose to accelerate to the left, not accelerate, or accelerate to the right. Before reaching the right hill top, the learner receives a reward of -1 per step.

An expert player should reach the goal in ≤ 120 steps.

2 Playing Mountain Car with Exploration Bonus

In this problem, we use reinforcement learning to train a Mountain Car player (discrete-action version). A starter code is **here**. This time, the starter code has two files: `HW5.py` and `mountaincar_bonus.py`, where the second file contains classes and functions related to exploration bonuses. When submitting, submit the two files separately (we will create separate entries for them).

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.

2.1 Exploration Bonus

We can apply exploration bonuses to any RL algorithms we discussed before—for example, DQN and PPO. The pseudocodes are provided in Algorithm 3 and Algorithm 4, respectively. The pseudocodes are similar as those in Homework 3 and Homework 4, except for the colored parts (blue or red), which are specific to exploration bonuses. The blue parts are already implemented in the starter code; the red part, i.e., the update of the bonus function, is left as your todo.

We consider two possible bonus functions $\xi(s)$: *tabular bonus* (Page 10 in this slide), and *random network distillation* (Page 14 in the same slide).

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 19 of Algorithm 3 or Line 43 of Algorithm 4) is performed as follows:

Algorithm 1 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 need to implement the TODO-1 in `get_bonus()` under the class `TabularExplorationBonus`.

Random Network Distillation (RND) 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 19 of Algorithm 3 or Line 43 of Algorithm 4) is performed by minimizing the prediction error:

Algorithm 2 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-2 in `get_bonus()` under the class `RandomNetworkDistillation`.

2.2 Tasks

Your task is to read the starter code and implement the bonus function updates, i.e., TODO-1 and TODO-2.

To run the code, type

```
HW5.py --seed <seed> --algorithm <algorithm> --exploration <exploration> [--nosaving]
```

Allowed parameters:

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

The code already implements DQN and PPO without bonuses (i.e., when `<exploration>` is `none`). If `--nosaving` is NOT set, then when running, the code will generate a folder

```
./models/<algorithm>_<exploration>_<seed>_<timestamp>/
```

and save **output figures** and **trained models** in it; if `--nosaving` is set, then the trained model will not be saved, while the output figures will be saved in the current folder. **It is important to save some trained models because they will be used in Section 3.**

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 and 4. 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 in the folder mentioned above. The trained model will also be saved per 100 episode (without overwriting each other).

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

(a) (5%) Run DQN with *no bonus* for 3000 episodes, i.e., run the commmand

```
HW5.py --seed <seed> --algorithm DQN --exploration none
```

for some `<seed>` you like. Paste the return figure and the heatmap figure here.

(b) (5%) Run PPO with *no bonus* for 3000 episodes, i.e., run the command

```
HW5.py --seed <seed> --algorithm PPO --exploration none
```

Paste the return figure and the heatmap figure here.

(c) (10%) Complete TODO-1 in the `TabularExplorationBonus` class and perform necessary hyperparameter tuning. Note that this class is in the file `mountaincar_bonus.py`. Run DQN with tabular bonus:

```
HW5.py --seed <seed> --algorithm DQN --exploration tabular
```

The default is 6000 episodes. **You may stop before reaching** 6000 **episodes if you are already satisfied with the score.** Paste the output figures (both the performance-over-time figure and the heatmap figure) here.

The score in this part will be calculated as

$$\min\left\{\frac{(\text{maximum running average within 6000 episodes})+200}{80},1\right\}\times 100\%.$$

In other words, if the maximum running average reaches -120 within 6000 episodes, then you get full credit.

(d) (10%) Use the same code in (c) to run PPO with tabular bonus:

```
HW5.py --seed <seed> --algorithm PPO --exploration tabular
```

Paste the output figures here. The score calculation is same as in (c).

(e) (10%) Complete the TODO-2 in the `RandomNetworkDistillation` class and perform necessary hyperparameter tuning. Note that this class is in the file `mountaincar_bonus.py`. Run DQN with RND bonus:

```
HW5.py --seed <seed> --algorithm DQN --exploration rnd
```

Paste the output figures here. The score calculation is same as in (c).

(f) (10%) Use the same code in (e) to run PPO with RND bonus:

```
HW5.py --seed <seed> --algorithm PPO --exploration rnd
```

Paste the output figures here. The score calculation is same as in (c).

3 Playing Mountain Car with Behavior Cloning

In this problem, we use Behavior Cloning to train an agent, where the expert is a model trained and saved in Section 2. The pseudocode of Behavior Cloning is Algorithm 6. This algorithm is discussed on Page 7 of this slide. The following command will be used:

```
HW5.py --seed <seed> --algorithm BC --expert_model_path <expert_model_path>
```

The `<expert_model_path>` is the path to the saved model in Section 2, for example, `<expert_model_path>` may look like

```
./models/PPO_tabular_10_20260415_072818/model_PPO_ep5700.pth
```

but you can pick any saved models that perform well. The code will generate one figure in real time that tracks the expert and the learner's performance over time.

(g) (30%) Complete TODO-3.1, TODO-3.2, TODO-3.3 indicated in Algorithm 6. Run Behavior Cloning for 1000 episodes and paste the output figure (tracking the performance of the expert and the learner over time) here. Your score will be calculated as

$$\min\left\{\frac{(\text{maximum running average within 1000 episodes})+200}{80}, 1\right\}\times 100\%.$$

A Appendix

A.1 DQN with Bonus

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 3. Note that the reward samples stored in the replay buffer should NOT include the bonus—see Line 56. 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 3 DQN with exploration bonus

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

2 Initialize the replay buffer $D = \{\}$.
3 Initialize the online network Q_θ with weights θ.
4 Initialize the target network $Q_{\overline{\theta}}$ with weights $\overline{\theta} = \theta$.

5 Sample $s_{\text{tmp}} \sim \rho$, where ρ is the initial state distribution.
6 **for** $k = 1, \dots$ **do**
7 $s_1 \leftarrow s_{\text{tmp}}$.
8 **for** $i = 1, 2, \dots, N$ **do**
9 Select action $a_i = \begin{cases} \operatorname{argmax}_a Q_\theta(s_i, a) & \text{with probability } 1-\epsilon, \\ \text{random action} & \text{with probability } \epsilon. \end{cases}$
10 Observe reward r_i and the next state s'_i.
11 $D \leftarrow D \cup \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$.
12 If s'_i is a terminal state, sample $s_{i+1} \sim \rho$; otherwise, set $s_{i+1} = s'_i$.
13 $s_{\text{tmp}} \leftarrow s_{N+1}$.

14 **for** $j = 1, \dots, M$ **do**
15 Randomly sample a minibatch $\mathcal{B} \subset \mathcal{D}$ with size $|\mathcal{B}| = B$.
16 Update the online network:

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

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

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

19 Update the bonus function $\xi(s)$ with $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$. // TODO-1 and TODO-2

A.2 PPO with Bonus

Adding an exploration bonus to PPO is also conceptually straightforward: just regard $R(s,a) + \xi(s)$ as the reward in the PPO update, where $R(s,a)$ is the original reward and $\xi(s)$ is the bonus. The pseudocode is Algorithm 4. In common implementation, we usually use *separate* value networks to represent the values functions corresponding to $R(s,a)$ and $\xi(s)$. Mathematically, this is same as using a single network to represent the value function corresponding to $R(s,a) + \xi(s)$; however, the former usually achieves a better performance. The value functions are denoted as $V_\phi^{\text{o}}(s)$ and $V_\varphi^{\text{b}}(s)$ for reward $R(s,a)$ and bonus $\xi(s)$, respectively. In Algorithm 4, we also adopt an advanced value estimator

called GAE which is not covered in the class. GAE takes a parameter $\lambda \in [0, 1]$ as input. When $\lambda = 0$, it becomes the TD estimator; when $\lambda = 1$, it becomes the MC estimator. In the starter code we use $\lambda = 0.95$.

Algorithm 4 PPO with exploration bonus

Parameters: N, M, B, α (policy learning rate), $\alpha^{\circ}, \alpha^{\mathsf{b}}$ (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_φ^{b} 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 $\text{GAE}_\lambda(\{s_i, a_i, r_i, s_i'\}_{i=1}^N, V_\phi^{\circ})$ (Algorithm 5).

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

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

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

  Define $A(s_i, a_i) = A^{\circ}(s_i, a_i) + A^{\mathsf{b}}(s_i, a_i)$ 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:

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

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

    Update value networks:

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

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

  Update the bonus function $\xi(s)$ with $\{(s_i, a_i, r_i, s_i')\}_{i=1}^N$. // TODO-1 and TODO-2

Algorithm 5 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**
  $\delta_i \leftarrow r_i + \gamma V(s_{i+1}) \mathbb{I}\{s_i' \text{ is not a terminal state}\} - V(s_i)$
  $A(s_i, a_i) \leftarrow \delta_i + \lambda\gamma A(s_{i+1}, a_{i+1}) \mathbb{I}\{s_i' \text{ is not a terminal state}\}$
return $\{A(s_i, a_i)\}_{i=1}^N$.

A.3 Behavior Cloning

Algorithm 6 Behavior Cloning

Parameters: N, M, B, α (learning rate).
Input: Expert policy π_{e}.

Initialize the replay buffer $D = \{\}$.
Initialize the policy network π_θ with weights θ.
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**
    Obtain an expert action $a_i \sim \pi_{\mathrm{e}}(\cdot|s_i)$
    Observe the next state s_i'.
    If s_i' is a terminal state, sample $s_{i+1} \sim \rho$; otherwise, set $s_{i+1} = s_i'$.
  $s_{\text{tmp}} \leftarrow s_{N+1}$.
  $D \leftarrow D \cup \{(s_i, a_i)\}_{i=1}^N$. // TODO-3.1
  for $j = 1, \ldots, M$ **do**
    Randomly sample a minibatch $\mathcal{B} \subset \mathcal{D}$ with size $|\mathcal{B}| = B$. // TODO-3.2
    Update the policy network: // TODO-3.3

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a) \in \mathcal{B}} \log \frac{1}{\pi_\theta(a|s)}.$$