Homework 3

6771 Reinforcement Learning (Fall 2025)

Submission deadline: 11:59pm, Nov 16

The latex template can be found here.

1 Performance Difference Lemma

In the discounted setting, the Performance Difference Lemma states that for any two stationary policies π' and π and any initial distribution ρ, it holds that

$$\mathbb{E}_{s\sim\rho}[V^{\pi'}(s)] - \mathbb{E}_{s\sim\rho}[V^{\pi}(s)] = \sum_{s,a} d_\rho^{\pi'}(s)\left(\pi'(a|s) - \pi(a|s)\right) Q^\pi(s,a).$$

Here,

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[\sum_{h=1}^{\infty}\gamma^{h-1}R(s_h,a_h)\,\middle|\, s_1 = s \text{ and } a_h\sim\pi(\cdot|s_h) \text{ for all } h\ge 1\right] \\ Q^\pi(s,a) &= \mathbb{E}\left[\sum_{h=1}^{\infty}\gamma^{h-1}R(s_h,a_h)\,\middle|\, (s_1,a_1) = (s,a) \text{ and } a_h\sim\pi(\cdot|s_h) \text{ for all } h\ge 2\right] \\ d_\rho^\pi(s) &= \mathbb{E}\left[\sum_{h=1}^{\infty}\gamma^{h-1}\mathbb{I}\{s_h = s\}\,\middle|\, s_1 \sim \rho \text{ and } a_h\sim\pi(\cdot|s_h) \text{ for all } h\ge 1\right] \end{aligned}$$

Below, we prove this lemma. Define a sequence of *Markov* policies:

$$\begin{aligned} \pi_0 &= (\pi,\pi,\pi,\pi\ldots), \\ \pi_1 &= (\pi',\pi,\pi,\pi\ldots), \\ \pi_2 &= (\pi',\pi',\pi,\pi\ldots), \end{aligned}$$

and so on. In words, π_h is a policy that executes π' in steps 1 through h, and executes π thereafter.

(a) (5%) Fix any integer $k\ge 1$. Show that

$$\begin{aligned} &\mathbb{E}_{s\sim\rho}\left[V^{\pi_k}(s)\right] - \mathbb{E}_{s\sim\rho}\left[V^{\pi_{k-1}}(s)\right] \\ &= \mathbb{E}\left[\sum_{h=k}^{\infty}\gamma^{h-1}R(s_h,a_h)\,\middle|\, s_1\sim\rho \text{ and } a_h\sim\pi'(\cdot|s_h) \text{ for all } 1\le h\le k \text{ and } a_h\sim\pi(\cdot|s_h) \text{ for all } h\ge k+1\right] \\ &\quad - \mathbb{E}\left[\sum_{h=k}^{\infty}\gamma^{h-1}R(s_h,a_h)\,\middle|\, s_1\sim\rho \text{ and } a_h\sim\pi'(\cdot|s_h) \text{ for all } 1\le h\le k-1 \text{ and } a_h\sim\pi(\cdot|s_h) \text{ for all } h\ge k\right] \end{aligned}$$

(b) (5%) Show that the above difference can be expressed as

$$\begin{aligned} &\mathbb{E}_{s\sim\rho}\left[V^{\pi_k}(s)\right] - \mathbb{E}_{s\sim\rho}\left[V^{\pi_{k-1}}(s)\right] \\ &= \sum_s \gamma^{k-1}\Pr\left\{s_k = s\,\middle|\, s_1\sim\rho \text{ and } a_h\sim\pi'(\cdot|s_h) \text{ for all } 1\le h\le k-1\right\}\sum_a\left(\pi'(a|s)-\pi(a|s)\right)Q^\pi(s,a). \end{aligned}$$

(c) (5%) Conclude that

$$\mathbb{E}_{s\sim\rho}[V^{\pi'}(s)] - \mathbb{E}_{s\sim\rho}[V^{\pi}(s)] = \sum_s d_\rho^{\pi'}(s) \sum_a \left(\pi'(a|s) - \pi(a|s)\right) Q^\pi(s,a).$$

2 RL in CartPole

In this problem, we will implement DQN and PPO to solve the classic CartPole environment. Download the starter code here. Like in HW1/HW2, we make value-based and policy-based algorithms share a Python class due to the structural similarity between a Q-network and a policy network. You're welcome to modify the code structure if you find a better design.

You can definitely find many good implementations of DQN and PPO online, but we encourage you to first complete your own implementation (you may seek help from LLMs). Afterwards, you're encouraged to compare your work with open-source implementations—for example, stable-baseline3. These reference codes often include useful tricks and engineering details that are not often discussed explicitly in textbooks or lectures.

Environment We focus on the CartPole environment, which is a game provided by the OpenAI Gym library. Specifically, we focus on the version of CartPole-v1. A brief introduction is given below, and more information about the environment can be found in `https://www.gymlibrary.dev/environments/classic_control/cart_pole/`.

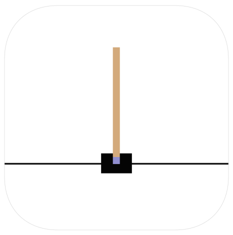

Figure 1: CartPole

To get familiar with the environment, you can run the function `generate_videos()` provided in the starter code, which generates a video of the game being played by a random policy. You may adapt this code to generate videos of your trained policy instead.

Starter Code The default command-line usage in the starter code is:

```
RL.py --seed [seed] --algorithm [algorithm] --figure [figure]
```

where `[seed]` specifies the random seed used to control both the algorithm's and the environment's randomness. This allows you to debug and compare different algorithms under the same initial conditions. `[algorithm]` indicates the algorithm to use—one of `DQN`, `DDQN`, `PPO`, or `RAND`. `[figure]` is the filename for saving the output plot (e.g., `output.png`). The starter code automatically saves the figure every 100 episodes.

You will be asked to include this output figure in your homework submission for several questions below.

We have already implemented a random policy that selects actions uniformly at random. To get started, try running:

```
RL.py --seed 0 --algorithm RAND --figure output_RAND.png
```

2.1 DQN

We will begin by using the standard DQN algorithm to play CartPole. The algorithm is shown in Algorithm 1.

It is common to set $N = M = 1$ in Algorithm 1, since DQN reuses past data via experience replay and does not need to wait until a large batch of new data is collected before performing an update. The parameters N and M are explicitly included to make the structure more comparable to that of PPO.

In the starter code, the exploration rate ϵ is time-varying and gradually decays to zero. Its scheduling is controlled by the parameters `EPS_START`, `EPS_END`, and `EPS_DECAY` in the starter code. Recommended hyperparameter values are provided in the starter code.

To reduce your workload, we have implemented the ϵ-greedy action selection rule and the replay buffer.

(a) (10%) Implement Algorithm 1 to play CartPole. Paste the output plot here, which should show both the return per episode and its running average over a window of 100 episodes.

Your score for this part will be computed as:

$$\min\left\{\frac{\text{maximum running average within 2000 episodes}}{475}, 1\right\} \times 100\%. \tag{1}$$

That is, if within the first 2000 episodes there exists a window of 100 episodes where the running average return exceeds 475, you will receive full credit. If you do not reach this threshold within 2000 episodes, you may also increase the total number of episodes in your plot, but your score will be scaled by a factor of $\frac{2000}{\text{total number of episodes used}}$. There is no need to average results over multiple random seeds—just choose a seed that yields good performance.

You may try some of the tips listed below, as some people have found them useful. This will make your implementation deviate from the original algorithm outlined in Algorithm 1.

- Gradient clipping: Clip the gradient if its magnitude exceeds a threshold (e.g., 0.5).
- Using Huber loss: Replace the squared loss in Eq. (2) with Huber loss (see this blog for a brief explanation).

(b) (5%) List the hyperparameters you use in (a):

Hyperparameter	Value
`MINIBATCH_SIZE` (B in Algorithm 1)	
`LR` (α in Algorithm 1)	
`EPS_START`	
`EPS_END`	
`EPS_DECAY`	
`TAU` (τ in Algorithm 1)	

If you use any trick beyond what is described in Algorithm 1, please briefly describe it here. Also, list any additional hyperparameters you introduced in the table above.

Algorithm 1 DQN (with ϵ-greedy exploration)

```
1  Parameters: $N, M, B, \alpha$ (learning rate), $\tau$ (target network update rate), $\epsilon$ (exploration probability).

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

5  Sample $s_{\text{tmp}} \sim \rho$, where $\rho$ is the initial state distribution.
6  for $k = 1, \ldots$ do
7      $s_1 \leftarrow s_{\text{tmp}}$
8      for $i = 1, 2, \ldots, N$ 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         $D \leftarrow D \cup \{(s_i, a_i, r_i, s'_i)\}$.
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 $j = 1, \ldots, M$ do
15         Randomly sample a minibatch $\mathcal{B} \subset \mathcal{D}$ with size $|\mathcal{B}| = B$.
16         Update the online network:
               $$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \Big( Q_\theta(s,a) - r - \gamma \max_{a'} Q_{\overline{\theta}}(s', a') \Big)^2 \quad (2)$$
17         where $\max_{a'} Q_{\overline{\theta}}(s', a') \triangleq 0$ if $s'$ is a terminal state.
18         Update the target network:
               $$\overline{\theta} \leftarrow \tau\theta + (1-\tau)\overline{\theta}.$$
```

2.2 Double DQN

Now change Eq. (2) in Algorithm 1 to the following:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_{i=1}^{B} \big(Q_\theta(s_i, a_i) - r_i - \gamma Q_{\overline{\theta}}(s'_i, a'_i) \big)^2 \qquad \text{with } a'_i \triangleq \operatorname*{argmax}_a Q_\theta(s'_i, a),$$

which yields Double DQN.

(c) (10%) Implement Double DQN to play CartPole. *Use the same random seed and the same choices of hyperparameters as in (a).* Paste the output plot here. The score for this part will be calculated using the same formula as in Eq. (1).

2.3 Proximal Policy Optimization

Next, we will use PPO to play CartPole. The PPO algorithm is shown in Algorithm 2.

(d) (20%) Implement PPO to play CartPole, and paste the output plot here. Your score for this part will be computed as:

$$\min\left\{\frac{\text{maximum running average within 6000 episodes}}{475}, 1\right\} \times 100\%.$$

Note that the allowed number of episodes to reach the desired performance is increased to 6000. As in part (a), you may go beyond 6000 episodes, but your score will be scaled by $\frac{6000}{\text{number of episodes you use}}$. Recommended choices of the hyperparameters are provided in the starter code.

As in DQN, there are some practical tips that go beyond Algorithm 2, which many people find useful:

- **Normalizing the advantage**: Instead of directly using the advantage estimates $\hat{A}(s_i, a_i)$ returned by GAE, normalize them as follows:

$$A'(s_i, a_i) = \frac{\hat{A}(s_i, a_i) - \mu}{\sigma + 10^{-8}}$$

  where $\mu = \frac{1}{N}\sum_{i=1}^{N} \hat{A}(s_i, a_i)$ and $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{A}(s_i, a_i) - \mu)^2}$. The 10^{-8} could be replaced by other small number.
- Gradient clipping.

We highlight the first tip, as we've also observed improvement in performance when using it.

(e) (5%) List the hyperparameters you use in (d):

Hyperparameter	Value
N (N in Algorithm 2)	
M (M in Algorithm 2)	
MINIBATCH_SIZE (B in Algorithm 2)	
LR (α and α' in Algorithm 2)	List both if you use $\alpha \neq \alpha'$
LAMBDA (λ in Algorithm 2)	
BETA (β in Algorithm 2)	
EPS (ϵ in Algorithm 2)	

If you use any trick beyond what is described in Algorithm 2, please briefly describe it here. Also, list any additional hyperparameters you introduced in the table above.

Remark Line 13 in Algorithm 2 reuses the advantage estimate $\hat{A}(s_i, a_i)$ to construct $\hat{V}(s_i)$, which is then used to train the value function in Line 20. This effectively makes Line 20 a TD(λ) update, using the same λ as in the advantage estimation for policy learning.

However, as we've mentioned in the class, these two λ values used in policy learning and value learning need not be the same. You may also explore the case where they differ. To do so, just compute a separate advantage estimate $\widetilde{A}(s_i, a_i)$ with Algorithm 3 using a different λ', and define $\hat{V}(s_i) = \widetilde{A}(s_i, a_i) + V_\phi(s_i)$. This results in a TD($\lambda'$) update for the value function.

For example, the paper that proposed GAE for policy learning actually uses TD(λ') = TD(1) for value learning (see their Section 5).

Algorithm 2 PPO

```
1  Parameters: $N, M, B, \alpha$ (policy learning rate), $\alpha'$ (value learning rate), $\lambda$ (GAE parameter), $\beta$ (KL penalty coefficient), $\epsilon$ (clipping parameter).
2  Initialize the policy network $\pi_\theta$ with weights $\theta$
3  Initialize the value network $V_\phi$ with weights $\phi$.

4  Sample $s_{\text{tmp}} \sim \rho$, where $\rho$ is the initial state distribution.
5  for $k = 1, \ldots$ do
6      $s_1 \leftarrow s_{\text{tmp}}$
7      for $i = 1, 2, \ldots, N$ do
8          Select action $a_i \sim \pi_\theta(\cdot|s_i)$.
9          Observe reward $r_i$ and the next state $s_i'$.
10         If $s_i'$ is a terminal state, sample $s_{i+1} \sim \rho$; otherwise, set $s_{i+1} = s_i'$.
11     $s_{\text{tmp}} \leftarrow s_{N+1}$
12     Compute $\hat{A}(s_i, a_i)$ for all $i = 1, \ldots, N$ by calling GAE$_\lambda(\{s_i, a_i, r_i, s_i'\}_{i=1}^N, V_\phi)$ (Algorithm 3).
13     Define $\hat{V}(s_i) = \hat{A}(s_i, a_i) + V_\phi(s_i)$ for all $i = 1, \ldots, N$.
14     Define $\hat{\pi}(a_i|s_i) = \pi_\theta(a_i|s_i)$ for all $i = 1, \ldots, N$.
15     $\hat{A}(s_i, a_i), \hat{V}(s_i), \hat{\pi}(a_i|s_i)$ should all be viewed as constant below—call tensor.detach() for them.

16     for $j = 1, \ldots, M$ do
17         Randomly sample a minibatch $\mathcal{B} \subset \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ with size $|\mathcal{B}| = B$.
18         Update the policy network:
               $$\theta \leftarrow \theta + \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \Big( \min\big\{ \varrho_\theta(s,a)\hat{A}(s,a), \varrho_\theta^{\text{clip}}(s,a)\hat{A}(s,a) \big\} - \beta\big(\varrho_\theta(s,a) - 1 - \log \varrho_\theta(s,a)\big) \Big),$$
19         where $\varrho_\theta(s,a) \triangleq \frac{\pi_\theta(a|s)}{\hat{\pi}(a|s)}$ and $\varrho_\theta^{\text{clip}}(s,a) \triangleq \max\{1-\epsilon, \min\{1+\epsilon, \varrho_\theta(s,a)\}\}$.
20         Update the value network:
               $$\phi \leftarrow \phi - \alpha' \nabla_\phi \frac{1}{B} \sum_{(s,a,r,s') \in \mathcal{B}} \Big( V_\phi(s) - \hat{V}(s) \Big)^2.$$
```

Algorithm 3 GAE$_\lambda$

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

3 Survey

(5%) Leave any feedback for the course.