# Homework 3

6501 Reinforcement Learning (Spring 2024)

Submission deadline: 11:59pm, April 7

**Rules**

- Collaboration is allowed but must be stated in precision per sub-problem. For example, comment in the sub-problem which idea is a result of a discussion with whom.
- Leveraging large language models is allowed but must be stated. Describe how you use them or provide precise prompts that you use.
- Please put all proofs (typed in latex) and figures for experiments in a single pdf file. Submit the pdf to the Question 1 in Gradescope. Put all codes in a single .py file **(not .ipynb file)**, and submit it to Question 2 in Gradescope.
- For theoretical problems, rigorous proofs are required.

## 1 Bellman Completeness

In this problem and the next, we will test your understanding about Bellman completeness. It naturally arises in the theoretical analysis of approximate value iteration with function approximation, but it is quite strong so in real world we seldom can claim it holds.

Let $M$ be an MDP, and let $R(s,a)$ and $P(s'|s,a)$ be the reward function and transition probabilities associated with it.

**Definition 1** (Bellman Completeness)**.** *We say a set of functions $\mathcal{F}$ is Bellman complete for $M$ if the following holds: for any $f' \in \mathcal{F}$, there exists an $f \in \mathcal{F}$ such that*

$$f(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[ \max_{a'} f'(s',a') \right] \qquad \textit{for all } s,a.$$

Next, we define a specific class of MDPs.

**Definition 2** (Linear MDP)**.** *Suppose that the learner is given some known feature mapping $\phi(s,a) \in \mathbb{R}^d$ for all $s,a$. We say $M$ is a linear MDP with respect to $\phi(s,a)$ if there exists $w \in \mathbb{R}^d$ and $\psi(s') \in \mathbb{R}^d$ for all $s'$ such that*

$$R(s,a) = \phi(s,a)^\top w,$$
$$P(s'|s,a) = \phi(s,a)^\top \psi(s')$$

*for all $s,a,s'$.*

(a) (10%) Prove that if $M$ is a linear MDP with respect to $\phi(s,a)$, then the function set $\mathcal{F} = \left\{ \phi(s,a)^\top \theta : \ \theta \in \mathbb{R}^d \right\}$ is Bellman complete for $M$.

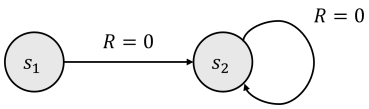


Figure 1: MDP Example

## 2 Bellman Completeness

In this problem, we would like to argue that a simple instance is NOT Bellman complete.

Consider the MDP $M$ depicted in Figure 1 that has two states $s_1, s_2$ and one action $a$. The transitions are deterministic as indicated by the black arrows: $P(s_2|s_1, a) = 1$, $P(s_2|s_2, a) = 1$. The reward function is all-zero: $R(s_1, a) = R(s_2, a) = 0$. Let the discount factor be $\gamma = 0.9$. Suppose that the state-action pairs are equipped with 1-dimensional features $\phi(s_1, a) = 1$ and $\phi(s_2, a) = 2$ that induce the following function set:

$$\mathcal{F} = \{\phi(s, a)\theta : \theta \in \Theta\}$$

for some bounded interval $\Theta = [-\beta, \beta]$ with $\beta > 0$.

(a) (8%) Prove that $\mathcal{F}$ is NOT Bellman complete for $M$. (Hint: by the definition of Bellman completeness in Definition 1, you will need to show that there exists an $f' \in \mathcal{F}$ such that for all $f \in \mathcal{F}$, $f(s, a) \neq R(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)}[\max_{a'} f'(s', a')]$ for some $s, a$).

(b) (4%) Argue that $\theta^\star = 0$ is the only solution that can realize $Q^\star$. In other words, show that $\theta^\star = 0$ is the unique solution that satisfies $Q^\star(s, a) = \phi(s, a)\theta^\star$ for all $s, a$.

(c) (8%) Argue that if the samples $(s, a, r, s')$ are drawn uniformly from the two states, i.e.,

$$(s, a, r, s') \sim \mathcal{D} := \text{Uniform}\{(s_1, a, 0, s_2), (s_2, a, 0, s_2)\},$$

then the LSVI update

$$\theta^{(k+1)} = \operatorname*{argmin}_{\theta \in \Theta} \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[ \left( \phi(s, a)\theta - r - \gamma \max_{a'} \phi(s', a')\theta^{(k)} \right)^2 \right]$$

does not converge to $\theta^\star$ if the initialization $\theta^{(1)} \neq \theta^\star$.

**Remark.** This example is a minor modification from the $w$-to-$2w$ example in Sutton and Barto's book [1] (Example 11.1). The book describes this as an example of "deadly triad" (explained in their Chapter 11.3). We, on the other hand, attribute its failure more specifically to "not satisfying Bellman completeness", since we have shown in the lecture that when Bellman completeness holds, LSVI can succeed and thus does not suffer from deadly triad.

## 3 Implementing Deep Q-learning

In this problem, we will implement deep Q-learning along with several techniques introduced in the class. Download the starter code here.

### 3.1 Environment

We focus on the Lunar Lander environment, which is a game provided by the OpenAI Gym library. In this environment, the agent controls a lander, with the objective of safely landing it on a designated landing pad. A brief introduction

is given below, and more information about the environment can be found in `https://www.gymlibrary.dev/environments/box2d/lunar_lander/`.

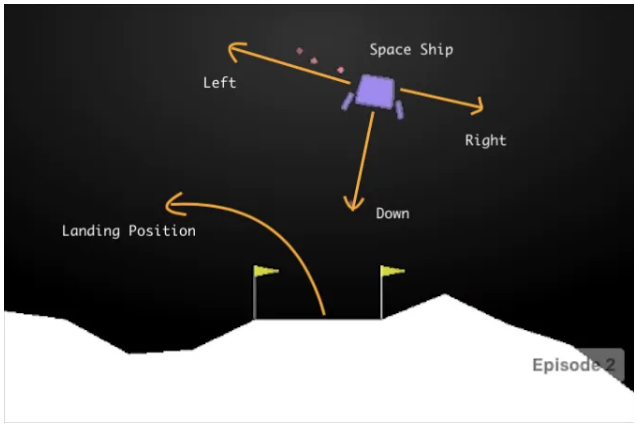


Figure 2: Lunar Lander

**Objective:** The agent's goal is to land the lunar module on the landing pad, which is always located at coordinates (0,0). Successful landing is marked by coming to a rest on the landing pad, maintaining a vertical orientation (no tilting), and achieving minimal velocity upon touchdown.

**Observations:** The state space includes continuous values representing the lander's horizontal and vertical position, horizontal and vertical velocity, angle, angular velocity, and two booleans indicating whether each leg is in contact with the ground. In the start code, the dimension of states is given by `env.observation_space.shape[0]`.

**Actions:** The agent can take discrete actions. In the simplest version, these actions typically include firing the left orientation engine, firing the main engine, firing the right orientation engine, and doing nothing. In the start code, the number of actions is given by `env.action_space.n`.

**Rewards:** The agent receives positive rewards for moving closer to the landing pad and making a soft landing and negative rewards for moving away, crashing, or using too much fuel (i.e., firing engines). The problem is regarded as solved if more than 200 points are earned.

**Episode Termination:** An episode ends when the lander crashes or comes to a rest, potentially with additional conditions for running out of time or fuel.

To configure the environment, you can simply run `pip install gymnasium[box2d]`. If you encounter an error related to "building wheels", you may run `pip install swig` first.

To get familiar with the environment. You can run function `generate_videos("random")` in the start code, which generates a video for the game playing by the random policy.

## 3.2 Vanilla DQN

We will first use vanilla DQN [2] to play the Lunar Lander game. Standard DQN consists of a Q-network along with a target network and a replay buffer. The target network slows down the move of the regression target, and the "replay buffer + sampling" mechanism decorrelates the samples. Both are important to stabilize the Q-network update. In addition, the use of replay buffer allows the algorithm to reuse previous data. The algorithm is summarized in Algorithm 1.

**Algorithm 1** Vanilla DQN (with $\epsilon$-greedy exploration)

1 Initialize the experience replay buffer $D$ to capacity $N$
2 Initialize the action-value network (policy network) $Q$ with weights $\theta$
3 Initialize the target network $\hat{Q}$ with weights $\theta_{tar} = \theta$
4 **for** episode $= 1, \ldots, M$ **do**
5  Initialize the environment with state $s_1$.
6  **for** $t = 1, \ldots$ **do**
7  With probability $\epsilon$, select a random action $a_t$. Otherwise, select action $a_t = \text{argmax}_a \, Q(s_t, a; \theta)$
8  Execute action $a_t$, observe reward $r_t$, and observe the next state $s_{t+1}$ if the episode hasn't terminate.
9  Let $s'_t$ be the terminal state if the episode terminates after $t$; otherwise, let $s'_t = s_{t+1}$.
10  Store transition $(s_t, a_t, r_t, s'_t)$ in replay buffer $D$.
11  Sample a random minibatch of transitions $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{M}$ with size $M$ from $D$.
12  Set

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is the terminal state} \\ r_j + \gamma \max_{a'} \hat{Q}(s'_j, a'; \theta_{tar}) & \text{otherwise} \end{cases}$$

13  Perform a gradient descent step on MSE loss $\frac{1}{M} \sum_{j=1}^{M} \left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to network parameter $\theta$.
14  Update $\theta_{tar} = \tau\theta + (1 - \tau)\theta_{tar}$ for some $\tau$ close to zero.
15  If the episode is terminated, let $s_{t+1}$ be the initial state of the next episode.

Algorithm 1 will run $M$ episodes and update parameters after playing one step in every episode. Besides the two components mentioned above, we also use $\epsilon$-greedy exploration to collect samples. The $\epsilon$ is time-varying and is given in the start code. You are free to change the exploration mechanism to other ones discussed in the class.

#### 3.2.1 Tasks

(a) (40%) Implement the vanilla DQN to play Lunar Lander in function `optimize_model_DQN()` and plot rewards for all episodes.

(b) (5%) Mean Square Error (MSE) loss is usually regarded as a good performance metric for supervised learning, but in reinforcement learning, we usually do not evaluate performance based on it (e.g., we do not measure how good a Q-network is by seeing how small its MSE loss is). Explain why.

### 3.3 Double DQN

Although DQN achieves great success, it suffers an "overestimation" issue which limits the further improvement of its performance. The overestimation issue means that the Q-values estimated by the network tend to be overly large. From Line 12/13 of Algorithm 1, since we minimize the MSE loss, we find $Q(s_j, a_j; \theta) \approx r_j + \max_{a'} \hat{Q}\left(s_{j+1}, a'; \theta_{tar}\right)$. The overestimation issue arises because, in practice, the max operator used in $\max_{a'} \hat{Q}\left(s_{j+1}, a'; \theta_{tar}\right)$ tends to select overestimated Q-values. When the max operator selects these overestimated values for updates, it propagates the overestimation bias through the learning process. This overestimation can lead to suboptimal policies, as the agent may prefer actions that lead to overestimated future rewards rather than the optimal actions.

To tackle this issue, Double DQN (DDQN) is proposed [3]. Compared with Vanilla DQN in Algorithm 1, DDQN only changes $y_j$ by redefining

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is the terminal state} \\ r_j + \gamma \hat{Q}(s'_j, \text{argmax}_{a'} \, Q(s'_j, a'; \theta); \theta_{tar}) & \text{otherwise} \end{cases}$$

All the remaining procedures of DDQN are the same as Algorithm 1.

### 3.3.1 Tasks

(a) (20%) Implement the DDQN to play Lunar Lander in function `optimize_model_DDQN()` and plot rewards for all episodes.

## 3.4 Dueling Networks (Optional)

*Although this technique is not covered in the lecture and this is an optional problem, you are encouraged to implement this since it only requires three additional lines from the previous DDQN (based on the TA's implementation).*

Dueling networks [4] provide a more sophisticated way to learn optimal Q-values. Unlike DQN and DDQN, which directly learn the optimal Q-values, dueling networks decompose Q-values into two components and learn them separately. Specifically, we first define advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ for any policy $\pi$, which measures the relative importance of each action given a state. This implies for optimal policy $\pi^*$, we have $Q^\star(s,a) = A^\star(s,a) + V^\star(s)$. Since we want to learn $Q^\star(s,a)$, the idea of dueling network is to use one network $A(s,a;\theta_1)$ to approximate $A^\star(s,a)$ and another network $V(s;\theta_2)$ to approximate $V^\star(s)$. Thus, one way to estimate $Q^\star(s,a)$ is $Q(s,a;\theta) = A(s,a;\theta_1) + V(s,\theta_2)$ where $\theta = (\theta_1,\theta_2)$. However, in practice, people never use this equation to learn. Instead, given the observation that $\max_a A^\star(s,a) = \max_a Q^\star(s,a) - V^\star(s) = 0$, we have $Q^\star(s,a) = A^\star(s,a) + V^\star(s) - \max_{a'} A^\star(s,a')$. This leads to the estimation $Q(s,a;\theta) = A(s,a;\theta_1) + V(s;\theta_2) - \max_{a'} A(s,a';\theta_1)$. Empirical evidence shows that define $Q(s,a;\theta) = A(s,a;\theta_1) + V(s;\theta_2) - \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} A(s,a;\theta_1)$, where $\mathcal{A}$ is the action set and $|\mathcal{A}|$ is the number of actions, usually results in better performance but this trick has no solid theoretical support.

Dueling networks only change the network structure to learn Q-values, and the following procedures are the same as DDQN. Although we have two networks and two parameters $(\theta_1,\theta_2)$, they are jointly optimized when we try to minimize the MSE loss of Q-values. In this problem, you will implement a dueling network algorithm given in Algorithm 2.

**Algorithm 2** Dueling Network with DDQN

Initialize the experience replay buffer $D$ to capacity $N$

Initialize the advantage network $A$ with weights $\theta_1$, and initialize the value network $V$ with weights $\theta_2$.

Initialize the target advantage network $\hat{A}$ with weights $\theta^1_{tar} = \theta_1$ and the target value network $\hat{V}$ with weights $\theta^2_{tar} = \theta_2$.

In the following procedure, define

$$Q(s, a; \theta) = A(s, a; \theta_1) + V(s; \theta_2) - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} A(s, a; \theta_1) \quad \text{where } \theta = (\theta_1, \theta_2)$$

$$\hat{Q}(s, a; \theta_{tar}) = \hat{A}(s, a; \theta^1_{tar}) + V(s; \theta^2_{tar}) - \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \hat{A}(s, a; \theta^1_{tar}) \quad \text{where } \theta_{tar} = (\theta^1_{tar}, \theta^2_{tar})$$

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

  Initialize the environment with state $s_1$.

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

    With probability $\epsilon$, select a random action $a_t$. Otherwise, select action $a_t = \text{argmax}_a \, Q(s_t, a; \theta)$

    Execute action $a_t$, observe reward $r_t$, and observe the next state $s_{t+1}$ if the episode hasn't terminate.

    Let $s'_t$ be the terminal state if the episode terminates after $t$; otherwise, let $s'_t = s_{t+1}$.

    Store transition $(s_t, a_t, r_t, s'_t)$ in replay buffer $D$.

    Sample a random minibatch of transitions $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{M}$ with size $M$ from $D$.

    Set

$$y_j = \begin{cases} r_j & \text{if } s'_j \text{ is the terminal state} \\ r_j + \gamma \hat{Q}(s'_j, \text{argmax}_{a'} \, Q(s'_j, a'; \theta); \theta_{tar}) & \text{otherwise} \end{cases}$$

    Perform a gradient descent step on MSE loss $\frac{1}{M} \sum_{j=1}^{M} \left(y_j - Q(s_j, a_j; \theta)\right)^2$ with respect to network parameter $\theta$.

    Update $\theta_{tar} = \tau\theta + (1 - \tau)\theta_{tar}$ for some $\tau$ close to zero.

    If the episode is terminated, let $s_{t+1}$ be the initial state of the next episode.

#### 3.4.1 Tasks

(a) (5%) Explain why people prefer to utilize $Q(s, a; \theta) = A(s, a; \theta_1) + V(s; \theta_2) - \max_{a'} A(s, a'; \theta_1)$ to estimate the optimal Q-values rather than using $Q(s, a; \theta) = A(s, a; \theta_1) + V(s; \theta_2)$.

(b) (15%) Implement the dueling networks with DDQN to play Lunar Lander in function `optimize_model_DN()` and plot rewards for all episodes.

### 3.5 Prioritized Experience Replay Buffer (Optional)

In class, we briefly mentioned the prioritized experience Replay buffer. In the standard replay buffer, experiences are stored and sampled uniformly at random, treating each experience equally regardless of its significance in learning. In contrast, Prioritized Experience Replay assigns a priority level to each experience based on the magnitude of its temporal difference (TD) error, which reflects how surprising or unexpected the experience is. Experiences with higher TD errors are considered more informative or valuable because they indicate a significant discrepancy between the predicted and actual outcomes, suggesting that the agent has more to learn from these experiences.

#### 3.5.1 Tasks

(a) (20%) Read the paper for prioritized experience replay buffer [5]. Implement their Algorithm 1 together with the dueling network technique to play Lunar Lander (i.e. implement prioritized experience replay + DDQN + dueling networks) and plot rewards for all episodes. Note that on page 4 of [5], two methods are given to determine priority. You

can choose anyone you like to implement. There is no start code for the prioritized experience replay buffer, and you need to define a new class to implement it from scratch.

After finishing these problems, you may get familiar with several tricks for deep Q-learning and find that all of them can be combined together. The combination of different tricks and detailed ablation studies are exhaustively discussed in [6].

# 4 Survey

(a) (5%) How much time did you spend on each part of the homework? What do you think about the course so far?

# References


[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[3] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[4] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[5] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[6] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.