Homework 3

4771 Reinforcement Learning (Spring 2026)

Submission deadline: 11:59pm, March 22

The latex template can be found here.

1 Deep Q-Learning for CartPole

In this problem, we will implement DQN to solve the classic CartPole environment.

You can definitely find many good implementations of DQN 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 may 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. In CartPole-v1, the learner can choose either *left* or *right* action in each step, which specifies the direction of the force imposed on the cart. The goal is to balance the pole on the cart as long as possible. The state is a four-dimensional vector that represents *cart position*, *cart velocity*, *pole angle*, and *pole angular velocity*, respectively. An episode terminates whenever 1) the cart position is outside the range of $[-2.4, 2.4]$, 2) the pole angle is outside the range of $[-12°, 12°]$, or 3) the episode length reaches 500. At each step before termination, the learner obtains a reward of $+1$. More information about the environment can be found in `https://gymnasium.farama.org/environments/classic_control/cart_pole/`.

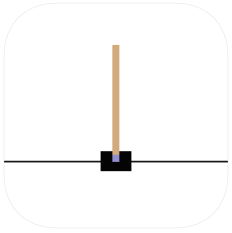

Figure 1: CartPole

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

```
HW3.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`, and `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:

```
HW3.py --seed 0 --algorithm RAND --figure output_RAND.png
```

1.1 Play CartPole Yourself

To get familiar with the environment, download this file and run

```
python cartpole_interactive.py
```

Use the left and right keys to control the cart and balance the pole. The line

```
step_duration = 0.5
```

sets the duration of each frame—smaller values make the game run faster. You may adjust it as you like.

The environment requires either a left or right action at every step. If you do not provide a new action at a step, the code will reuse the action from the previous step. Thus, repeatedly pressing the same key has the same effect as just pressing it once.

(a) (5%) Play CartPole yourself. Take a screenshot of the final game-over view from your favorite run and paste it here.

1.2 DQN

We will begin by using the standard DQN algorithm to play CartPole. The algorithm is shown in Algorithm 1. Download the starter code here. You're welcome to modify the code structure if you find a better design.

In the starter code, the exploration rate ϵ is time-varying and gradually decays. Its scheduling is controlled by the parameters `EPS_START`, `EPS_END`, and `EPS_DECAY` in the starter code in the following way:

$$\epsilon_j = \texttt{EPS\_END} + (\texttt{EPS\_START} - \texttt{EPS\_END}) \times \exp\left(-j/\texttt{EXP\_DECAY}\right) \quad \text{in episode } j.$$

In words, the exploration rate starts from `EPS_START` and then gradually decreases to `EPS_END`. The rate of the decrease is controlled by `EXP_DECAY`—the smaller the faster.

(b) (20%) Read the starter code and implement Algorithm 1 to play CartPole. The parts that you need to implement are tagged with `TODO 1, 2.1, 2.2, 2.3` in Algorithm 1 and explained more in the starter code. 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 2000 episodes there exists a window of 100 episodes where the running average return exceeds 475, you will receive full credit.

There is no need to average results over multiple random seeds—just choose a seed that yields good performance.

(c) (10%) List the hyperparameters you use in (b):

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 in the above table any additional hyperparameters you introduced.

Algorithm 1 DQN (with ϵ-greedy exploration)

Parameters: N, M, B, α (learning rate), τ (target network update rate), ϵ (exploration probability).

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

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 $a_i = \begin{cases} \operatorname{argmax}_a Q_\theta(s_i, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$
    Observe reward r_i, next state s'_i, and $t_i \in \{0, 1\}$ that indicates whether the episode has terminated (1 if terminated).
    If $t_i = 1$, sample $s_{i+1} \sim \rho$; otherwise, set $s_{i+1} = s'_i$.
  $s_{\text{tmp}} \leftarrow s_{N+1}$
  $D \leftarrow D \cup \{(s_1, a_1, r_1, s'_1, t_1), (s_2, a_2, r_2, s'_2, t_2), \ldots, (s_N, a_N, r_N, s'_N, t_N)\}$. // TODO 1
  for $j = 1, \ldots, M$ **do**
    Randomly sample a minibatch $\mathcal{B} \subseteq \mathcal{D}$ with size $|\mathcal{B}| = B$. // TODO 2.1
    Update the main network: // TODO 2.2

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a,r,s',t) \in \mathcal{B}} \left(Q_\theta(s, a) - r - \gamma \mathbb{I}[t = 0] \max_{a'} Q_{\overline{\theta}}(s', a') \right)^2 \tag{2}$$

    Update the target network: // TODO 2.3

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

1.3 Double DQN

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

$$\theta \leftarrow \theta - \alpha \nabla_\theta \frac{1}{B} \sum_{(s,a,r,s',t) \in \mathcal{B}} \left(Q_\theta(s, a) - r - \gamma \mathbb{I}[t = 0] Q_{\overline{\theta}}(s', a') \right)^2 \qquad \text{with } a' \triangleq \operatorname*{argmax}_a Q_\theta(s', a),$$

which yields Double DQN.

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

2 Survey

(5%) Leave any feedback for the course.