1. (Implementation of the DQN and Double DQN algorithms) You are required to implement the DQN and Double DQN algorithms for the gym pendulum-v0 environment. This environment is part of the classic control environments. The system in this environment consists of a pendulum attached at one end to a fixed point and the other end being free. The pendulum begins in a random position and the objective is to apply torque to the free end to swing it into an upright position. When the pendulum is upright upwards, a reward of 0 is given, while being in other positions yields a negative reward. The environment itself has no termination state but the episode truncates at 200 steps.

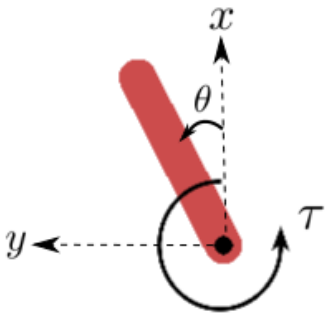

Figure 1: The pendulum environment.

In this environment, the observation is a `ndarray` with shape `(3,)` representing the x-y coordinates of the pendulum's free end and its angular velocity.

Num	Observation	Min	Max
0	x = cos(theta)	-1.0	1.0
1	y = sin(theta)	-1.0	1.0
2	Angular Velocity	-8.0	8.0

Figure 2: Observation space.

The vanilla action space in the environment is continuous, where each action is a `ndarray` with shape `(1,)`, representing the torque applied to free end of the pendulum. To apply value-based methods in this environment, we further discretize the action space into a discrete one, where there are 11 actions and each discrete action $a \in \{0, 1, \ldots, 10\}$ represents torque $0.4 \times a - 2$. You can refer to the website for more detailed information about this environment.

Num	Action	Min	Max
0	Torque	-2.0	2.0

Figure 3: Vanilla action space (continuous).

You are required to first implement

(a) the computation of the Q values of next-states for Double DQN;
(b) the computation of the Q values of next-states for vanilla DQN;
(c) the computation of the expected Q values of current states; and
(d) the computation of the TD loss in function `learn()` of `main.py`.

And then implement

(a) the push operation of one sample in function `push()` of `buffer.py`; and

(b) the sample operation of a batch of samples in function `sample()` of `buffer.py`.

You may refer to [1] and [2] for the algorithmic details of DQN and Double DQN, respectively. Run your codes to show the performances of vanilla DQN, DQN without using the target network, DQN without using the replay buffer, and Double DQN algorithms in the pendulum-v0 environment. Answer the following questions.

(a) Compare the performance difference between vanilla DQN and DQN without using the target network. Discuss the reason for this difference.

(b) Compare the performance difference between vanilla DQN and DQN without using the replay buffer. Discuss the reason for this difference.

(c) Compare the performance difference and estimated maximum Q-value between vanilla DQN and Double DQN. Discuss the reason for this difference.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. URL `https://doi.org/10.1038/nature14236`.

[2] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2094–2100. AAAI Press, 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389`.