535507 Spring 2025: Deep Learning **(Due: 2025/04/29, Tuesday, 23:59)**

Lab 5: Value-Based Reinforcement Learning

Instructor: Ping-Chun Hsieh
TAs: Wei Hung, Po-Hsuan Wen

Intended Learning Outcomes:

- Understand the core components of the Deep Q-Network algorithm.
- Gain hands-on experience implementing DQNs in PyTorch.
- Learn how to use DQNs in both low-dimensional and high-dimensional (visual) environments.
- Explore several practical enhancements to the DQN algorithm: Double DQN, Prioritized Experience Replay, and multi-step return.

1 Background

In this assignment, you will implement and experiment with Deep Q-Networks (DQNs) to solve classic control and visual reinforcement learning (RL) tasks using PyTorch and OpenAI Gym environments. The homework consists of three main parts, progressing from basic DQN implementation to incorporating advanced techniques to improve learning efficiency.

Vanilla DQN. As described in Lecture 8, DQN [3] combines Q-learning with a neural function approximator and leverages two practical techniques, namely *experience replay* and *target network*. Specifically, experience replay refers to the use of a replay memory that stores transitions (s, a, r, s') collected from the environment, and target network refers to maintaining a second Q network which is updated only periodically every N updates from the main Q network. The training of DQN centers around the minimization of the following loss function

$$L_{\text{DQN}}(\theta) := \frac{1}{2} \sum_{(s,a,r,s') \in D} \Big(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \bar{\theta}) - Q(s, a; \theta) \Big)^2, \tag{1}$$

where D denotes the sampled batch of transitions from the replay memory, and θ and $\bar{\theta}$ denote the network parameters of the main Q network and the target Q network, respectively.

Enhancements to vanilla DQN. To improve the sample efficiency of DQN, various techniques have been proposed [2, 1], such as enhancements for the loss function, the replay memory sampling strategy, and the exploration strategy. In this lab, we focus on the following three techniques:

- **Double DQN (DDQN)**: To mitigate the overestimation bias in DQN, one can apply the double Q-learning technique, which decouples the maximization step in the loss function of DQN into two sub-steps, i.e., using the main Q network for selecting the greedy action while using the target Q network for taking the Q values. Accordingly, the loss function of DDQN is

$$L_{\text{DDQN}}(\theta) := \frac{1}{2} \sum_{(s,a,r,s') \sim D} \Big(r + \gamma Q\big(s', \arg\max_{a' \in A} Q(s, a; \theta); \bar{\theta}\big) - Q(s, a; \theta) \Big)^2 \tag{2}$$

- **Prioritized Experience Replay (PER)**: PER [4] improves learning by sampling transitions with higher learning potential more frequently. Each transition i in the replay buffer is assigned a priority p_i, often based on the magnitude of its Bellman error:

$$p_i = |\delta_i| + \epsilon$$

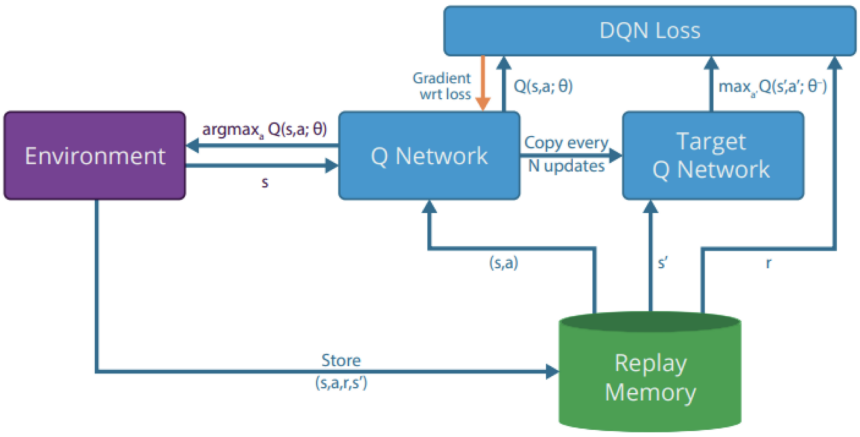

Figure 1: The architecture of vanilla DQN.

where $\delta_i = r_i + \gamma \max_{a'} Q(s'_i, a') - Q(s_i, a_i)$ is the Bellman error, and $\epsilon > 0$ is a small constant to ensure all transitions are sampled with non-zero probability. The probability of sampling transition i is:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where $\alpha \in [0, 1]$ controls the degree of prioritization. To correct the introduced bias, importance-sampling (IS) weights are used during training:

$$w_i = \left(\frac{1}{N \cdot P(i)} \right)^\beta$$

where N is the replay buffer size and $\beta \in [0, 1]$ controls the degree of correction. These weights are typically normalized to stabilize learning.

- **Multi-Step Return**: Multi-step learning in Deep DQN enhances learning efficiency by propagating rewards over multiple time steps, allowing the agent to learn from longer-term outcomes. Instead of using the immediate reward for a single transition, multi-step DQN uses the accumulated return over n steps. The n-step return $R_t^{(n)}$ is defined as:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n \max_{a'} Q(s_{t+n}, a')$$

Here, r_{t+k} is the reward at time $t + k$, γ is the discount factor, and $Q(s_{t+n}, a')$ is the estimated future return from the state s_{t+n}. This return replaces the target in the loss function, helping the agent learn more effectively from delayed rewards and reducing the bias-variance tradeoff in value estimation.

2 Lab Description

2.1 Task 1: Vanilla DQN in a toy environment

Goal: Implement a vanilla DQN using PyTorch to solve the `CartPole-v1` (https://gymnasium.farama.org/environments/classic_control/cart_pole/) from OpenAI Gym.

Requirements:

- Use `dqn.py` as the starting point for your implementation.

- Use a simple fully connected neural network to approximate the Q-function. The model size can be configured freely by yourself.
- Implement an epsilon-greedy policy for action selection.
- Use experience replay with uniform sampling and a target network.
- Evaluate and plot the total episodic rewards versus environment steps (preferably via Weight and Bias).

(a) CartPole (b) Pong

Figure 2: The evaluation domains used in Lab 5.

2.2 Task 2: Vanilla DQN with Visual Observations on Atari

Goal: Extend your DQN implementation to work on high-dimensional visual input using the `Pong-v5` environment from the Arcade Learning Environment suite, also known as Atari. You can find the detailed definitions of the states, actions, and rewards of each environment at the official website https://ale.farama.org/index.html.

Requirements:

- Preprocess the input frames (grayscale, resize, and stack frames)
- Use a convolutional neural network (CNN) as the Q-function approximator
- Evaluate and plot the total episodic rewards versus environment steps (preferably via Weight and Bias).

```python
import gymnasium as gym
import ale_py

gym.register_envs(ale_py)

# Initialise the environment
env = gym.make("ALE/Breakout-v5", render_mode="human")

# Reset the environment to generate the first observation
observation, info = env.reset(seed=42)
for _ in range(1000):
    # this is where you would insert your policy
    action = env.action_space.sample()

    # step (transition) through the environment with the action
    # receiving the next observation, reward and if the episode has terminated or truncated
    observation, reward, terminated, truncated, info = env.step(action)

    # If the episode has ended then we can reset to start a new episode
    if terminated or truncated:
        observation, info = env.reset()

env.close()
```

Figure 3: The typical usage of the Atari environments in RL.

2.3 Task 3: Enhanced DQN

Goal: Improve the learning efficiency of your DQN agent by incorporating the following enhancements:

- Double DQN
- Prioritized Experience Replay (PER)
- Multi-Step Return

Requirements:

- Integrate the enhancements into your DQN code (you may start from either CartPole or Pong implementation).
- Justify the integration choices.
- Compare training performance (e.g., learning speed, final reward) against vanilla DQN using the `Pong-v5` environment.

3 Model and Package Configurations

- All the modules or classes (listed below) are already provided in the file `dqn.py`. Please do not change the structure of the modules.
  - DQN
  - PrioritizedReplayBuffer
  - AtariPreprocessor
  - DQNAgent
- The model sizes can be freely configured by yourself. That being said, it is expected that Q networks with only a few hidden layers (less than 8) would be sufficient for successful training.
- Regarding the GPU memory usage, you may need to pay attention to the size of your replay memory, especially for the Atari game. Typically, the memory size is set to around 50k to 500k, depending on your GPU memory.
- Training in Pong takes much longer than that in CartPole. As a rule of thumb, Pong typically requires about up to 1 million environmental steps before converging to a dominant score of around 21 points. This could convert to up to about 20 hours of wall clock time (e.g., under a RTX3090 GPU). **Therefore, please do start early so that you have sufficient time for training**.
- The recommended package versions are as follows:
  - `Python` >= 3.8
  - `gymnasium` 1.1.1
  - `ale-py` >= 0.10.0
  - `opencv-python`
  - `torch`
  - `wandb`

4 Grading Policy

4.1 Report (50%)

- Introduction (5%): Please provide a high-level introduction to your report. You can mention the most important findings and the overall organization of this report.
- Your implementation (20%): Please briefly explain your implementation for Tasks 1-3. Specifically, please describe
  - How do you obtain the Bellman error for DQN?
  - How do you modify DQN to Double DQN?
  - How do you implement the memory buffer for PER?
  - How do you modify the 1-step return to multi-step return?
  - Explain how you use Weight & Bias to track the model performance.
- Analysis and discussions (25%)
  - Plot the training curves (evaluation score versus environment steps) for Task 1, Task 2, and Task 3 separately (10%).
  - Analyze the sample efficiency with and without the DQN enhancements. If possible, perform an ablation study on each technique separately (15%).
- Additional analysis on other training strategies (Bonus up to 10%)

4.2 Attained Model Performance: Snapshots and Demo Video (50%)

4.2.1 Demo Video

Please record a demo video that showcases two things via screen recording:

- Your source code: Please use about 2 minute to describe your implementation.
- Your model performance: Please use about 3 minute to demo your obtained models for Task 1, Task 2, and Task 3.

The demo video shall be between 5-minute to 6-minute long. **Please prepare your demo video in English** (unless there is any special circumstances discussed beforehand with the TAs). **Notably, your model snapshots will NOT be graded if you do not provide a valid demo video.**

4.2.2 Model Snapshots

- Task 1 (15%): The grading of Task 1 would depend on the evaluation score of your submitted snapshot. **Please use the best snapshot that you have obtained during the training process.** The score will be determined by the following equation

$$\text{Score percentage} = \frac{\min\{\text{average evaluation score}, 480\}}{480} \times 15\% \tag{3}$$

  Accordingly, to get full score for this part, your vanilla DQN shall constantly achieve an average score above 480 (over 20 evaluation episodes).

- Task 2 (20%): Similarly, the grading of Task 2 would depend on the evaluation score of your submitted snapshot. **Please use the best snapshot that you have obtained during the training process.** The score will be determined by the following equation

$$\text{Score percentage} = \frac{\min\{\text{average evaluation score}, 19\} + 21}{40} \times 20\% \tag{4}$$

Accordingly, to get full score for this part, your vanilla DQN shall steadily achieve an average score above 19 (over 20 evaluation episodes).

- Task 3 (15%): The grading of Task 3 would depend on the **sample efficiency of your enhanced DQN**. Please submit 5 model snapshots that are trained for 200k, 400k, 600k, 800k, and 1M environment steps.

Environment steps needed (Reaching score 19 on Pong)	200k	400k	600k	800k	1M	>1M
Score Percentage	15%	12%	10%	8%	6%	3%

5 Format of Submitted Files

You should submit a single ZIP file that contains all your deliverables. Please strictly follow the naming and structure below:

- **ZIP file**: Compress all files into one archive.
- **Filename**: `LAB5_{StudentID}_{YourName}.zip`

The ZIP file should contain the following structure:

Directory Structure

```
LAB5_StudentID_YourName.zip
|-- LAB5_StudentID_YourName_Code/         <- Source code folder
|   |-- dqn.py                            <- Your code files
|   |-- (any other .py files)
|-- LAB5_StudentID_YourName.pdf           <- Technical report (single PDF)
|-- LAB5_StudentID_YourName.mp4           <- Demo video (5 - 6 minutes)
|-- LAB5_StudentID_task1_cartpole.pt      <- Task 1 model snapshot
|-- LAB5_StudentID_task2_pong.pt          <- Task 2 model snapshot
|-- LAB5_StudentID_task3_pong200000.pt    <- Task 3 snapshot (step = 200k)
|-- LAB5_StudentID_task3_pong400000.pt    <- Task 3 snapshot (step = 400k)
|-- ...
|-- LAB5_StudentID_task3_pong1000000.pt   <- Task 3 snapshot (step = 1M)
```

Note:

- Only source code files should go into the `Code` folder.
- All other items (report, video, models) must be placed directly in the ZIP root.
- A 5-point penalty will be applied for any incorrect filenames or folder structure.

References

[1] J. S. O. Ceron and P. S. Castro. Revisiting Rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning*, pages 1373–1383, 2021.

[2] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.