Lab 2: Deep Q-Network (DQN)

Lab Objective:
In this lab, you will learn and implement the Deep Q-Network algorithm by solving MsPacman-v5.

Important Date:
- Submission deadline: 10/15 (Sun) 23:59

Turn in:
1. Experiment report (.pdf)
2. Source code [NOT including model weights]

Notice: zip all files with name “RL_LAB2_StudentId_Name.zip”,

e.g.: 「RL_LAB2_311551059_陳昱丞.zip」

```
RL_LAB2_311551059_陳昱丞
├── src
│   ├── base_agent.py
│   …
└── report.pdf
```

(Put all your source code in **src** directory)

(Wrong format deduction: -5pts; Multiple deductions may apply.)

Lab Description:
- Understand the mechanism of both behavior network and target network.
- Understand the mechanism of experience replay buffer.
- Learn to construct and design neural networks.

Requirements:
- Implement DQN
    - Construct the neural network.
    - Select action according to epsilon-greedy.
    - Construct Q-values and target Q-values.
    - Calculate loss function.
    - Update behavior and target network.
    - Understand Deep Q-learning mechanisms.
    - Understand common techniques in Atari environment like frame stack, grayscale…

Game Environment – MsPacman-v5:
- Introduction: Your goal is to collect all of the pellets on the screen while avoiding the ghosts.
- Observation: By default, the environment returns the RGB image that is displayed to human players as an observation.

- Action [9]:

Num	Action
0	NOOP
1	UP
2	RIGHT
3	LEFT
4	DOWN
5	UPRIGHT
6	UPLEFT
7	DOWNRIGHT
8	DOWNLEFT

Algorithm – Deep Q-learning with experience replay:

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Implementation Details:

Network Architecture

- Deepmind DQN network. Please refer to the sample code for details.

Training Hyper-Parameters

- Memory capacity (experience buffer size): 100000
- Batch size: 32
- Warmup steps: 20000
- Optimizer: Adam
- Learning rate: 0.0000625
- Epsilon: 1 → 0.1
- Gamma (discount factor): 0.99
- Update network every 4 steps
- Update target network every 10000 steps

You can tune the hyperparameter yourself.

Bonus:

- Solve Enduro-v5 using DQN.
- Add three improvements in your DQN implementation.
  1. Double DQN.
  2. Dueling DQN.
  3. DQN with parallelized rollout.

Scoring Criteria:

Show your results, otherwise no credit will be granted.

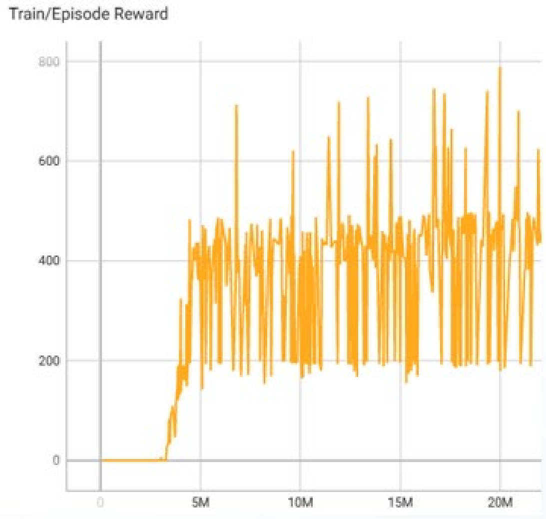
Your Score = report (30%) + report bonus (20%) + demo performance (50%) + demo questions (20%)

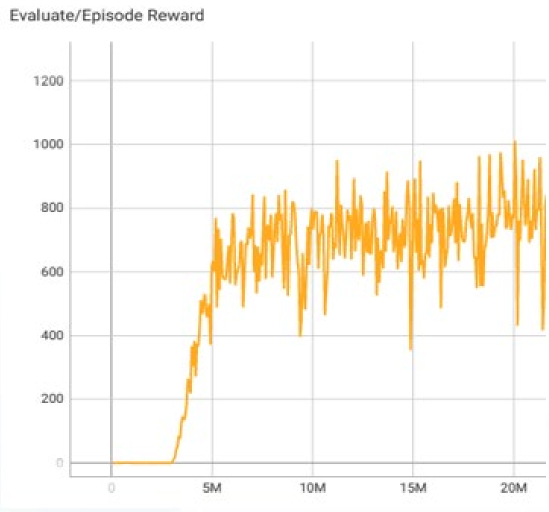
- **Report contains two parts:**
  - ■ **Experimental Results (30%)**
    - **(1)** Screenshot of Tensorboard training curve and testing results on DQN.
  - ■ **Experimental Results and Discussion of bonus parts (bonus) (20%)**
    - **(1)** Screenshot of Tensorboard training curve and testing results on Enduro-v5 (10%).
    - **(2)** Screenshot of Tensorboard training curve and testing results on DDQN, and discuss the difference between DQN and DDQN (3%).
    - **(3)** Screenshot of Tensorboard training curve and testing results on Dueling DQN, and discuss the difference between DQN and Dueling DQN (3%).
    - **(4)** Screenshot of Tensorboard training curve and testing results on DQN with parallelized rollout, and discuss the difference between DQN and DQN with parallelized rollout (4%).
- **Demo Performance (50%):**
  - ■ Test your best model for one game.
  - ■ You have to show the video while testing. You can use env.render() or save video function to achieve this.
  - ■ You can use a fixed random seed to reproduce your best game score.
  - ■ Demo performance – Score table:

Reward (Game score)	Points (50%)
0 ~ 500	0
500 ~ 1000	10
1000 ~ 1500	20
1500 ~ 2000	30
2000 ~ 2500	40
2500 ~ 3000	45
> 3000	50

Examples of Tensorboard training curve and testing results:

- Training curve:

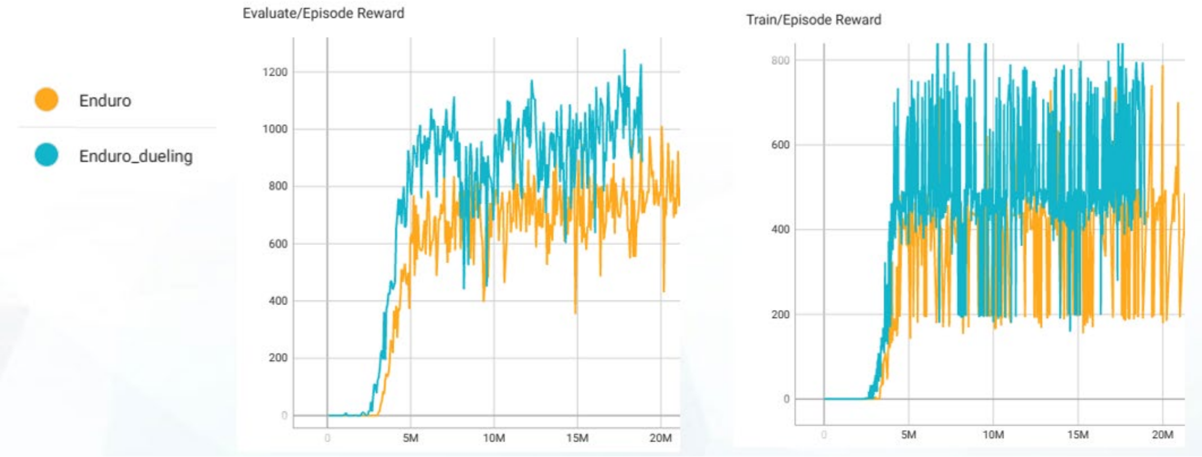




- Testing results (5 games):

```
Episode: 1       Length: 13301   Total reward: 1040.00
Episode: 2       Length: 13307   Total reward: 1066.00
Episode: 3       Length: 9969    Total reward: 705.00
Episode: 4       Length: 13286   Total reward: 1059.00
Episode: 5       Length: 16630   Total reward: 1318.00
average score: 1037.6
```

- Training curve (comparison):



References:

[1] Mnih, Volodymyr et al. "Playing Atari with Deep Reinforcement Learning." ArXiv abs/1312.5602 (2013).
[2] Mnih, Volodymyr et al. "Human-level control through deep reinforcement learning." Nature 518 (2015): 529-533.
[3] Van Hasselt, Hado, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning." AAAI. 2016.
[4] Lillicrap, Timothy P. et al. "Continuous control with deep reinforcement learning." CoRR abs/1509.02971 (2015).
[5] Silver, David et al. "Deterministic Policy Gradient Algorithms." ICML (2014).
[6] OpenAI. "OpenAI Gym Documentation." Retrieved from Getting Started with Gym: https://gym.openai.com/docs/.
[7] PyTorch. "Reinforcement Learning (DQN) Tutorial." Retrieved from PyTorch Tutorials: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.