LAB 2

TA 陳昱丞

2023/10/3

yucheng.cs11@nycu.edu.tw

Deadline: 2023/10/15 (Sun) 23:59

Demo: 2023/10/16 (Mon) 18:00

In this lab,

Must use sample code,
otherwise no credit.

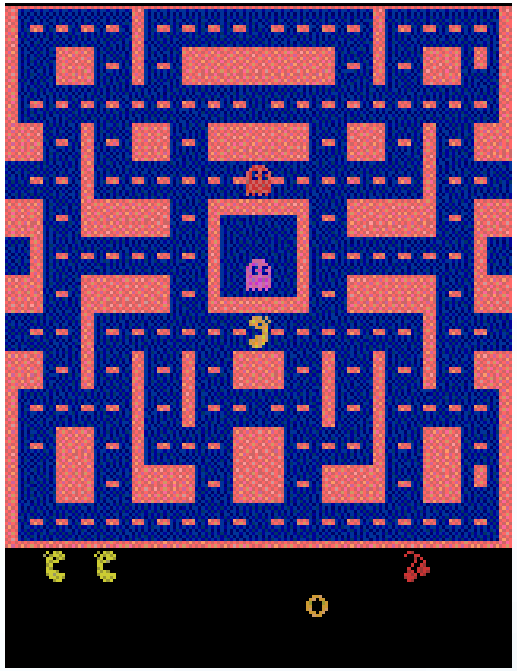
Atari Games

- The Atari 2600 is a video game console released in September 1977 by Atari Inc. The 2600 was typically bundled with two joystick controllers, a conjoined pair of paddle controllers, and a cartridge game — initially Combat and later Pac-Man. The Atari 2600 was wildly successful during the early 1980s.
- Play Atari games online: https://www.free80sarcade.com/all2600games.php

MsPacman-v5

- Introduction:
  - Your goal is to collect all of the pellets on the screen while avoiding the ghosts.
- Observation space:
  - The whole image
- Action space:

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

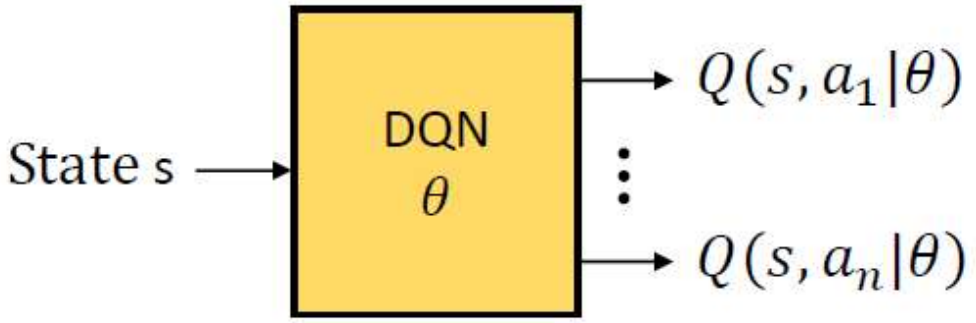


https://www.gymlibrary.dev/environments/atari/ms_pacman/

Deep Q-Network (DQN)

Target Q:

$$Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta)$$



Loss function:

$$L_Q(s_t, a_t|\theta) = (Y_t^Q - Q(s_t, a_t|\theta))^2$$

Gradient descent:

$$\nabla_\theta L_Q(s_t, a_t|\theta) = (Y_t^Q - Q(s_t, a_t|\theta))\nabla_\theta Q(s_t, a_t|\theta)$$

Deep Q-Network (DQN)

Algorithm 1 – 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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Behavior and target network

ϵ-greedy based on behavior network

Experience replay

Update behavior and target network

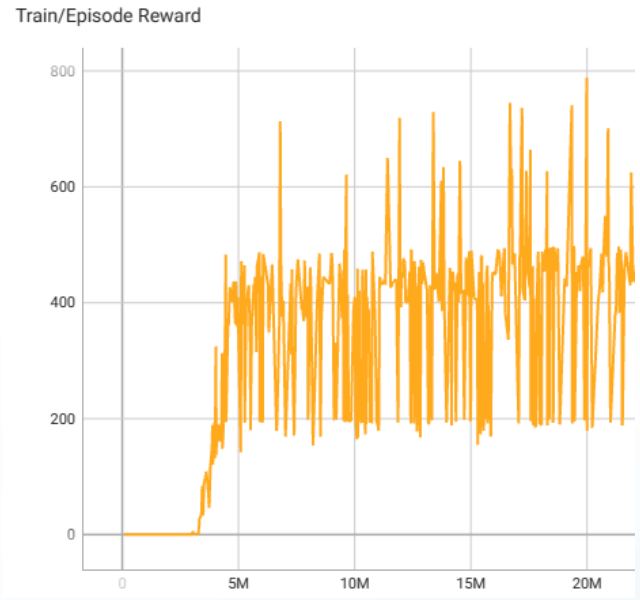
TODO

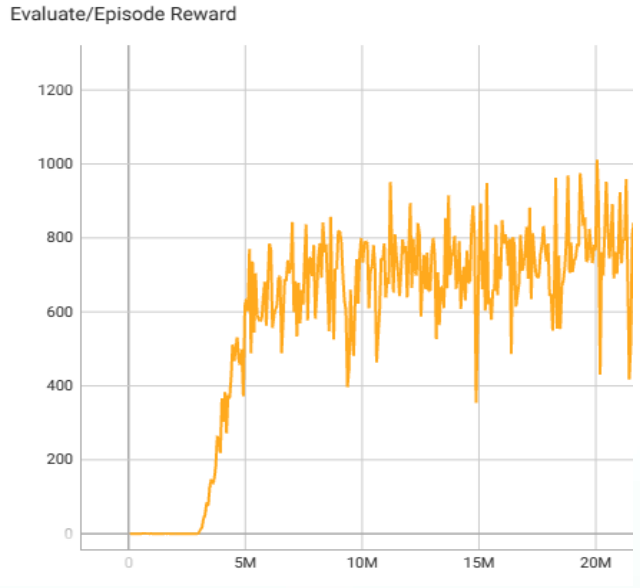
- Solve MsPacman-v5 using DQN.
- (Bonus) Add Double DQN in your implementation.
- (Bonus) Add Dueling DQN in your implementation.
- (Bonus) Add parallelized rollout in your implementation.
- (Bonus) Solve Enduro-v5 using DQN.
- Find the #TODO comments and hints, remove the raise NotImplementedError.
- Inherit from the "DQNBaseAgent" and override the "decide_agent_actions" and "update_behavior_network" functions.
- Screenshot your Tensorboard training curve and testing results and put it on the report.

TODO

- Screenshot of Tensorboard training curve and testing results and put it on the report.

Training curve:



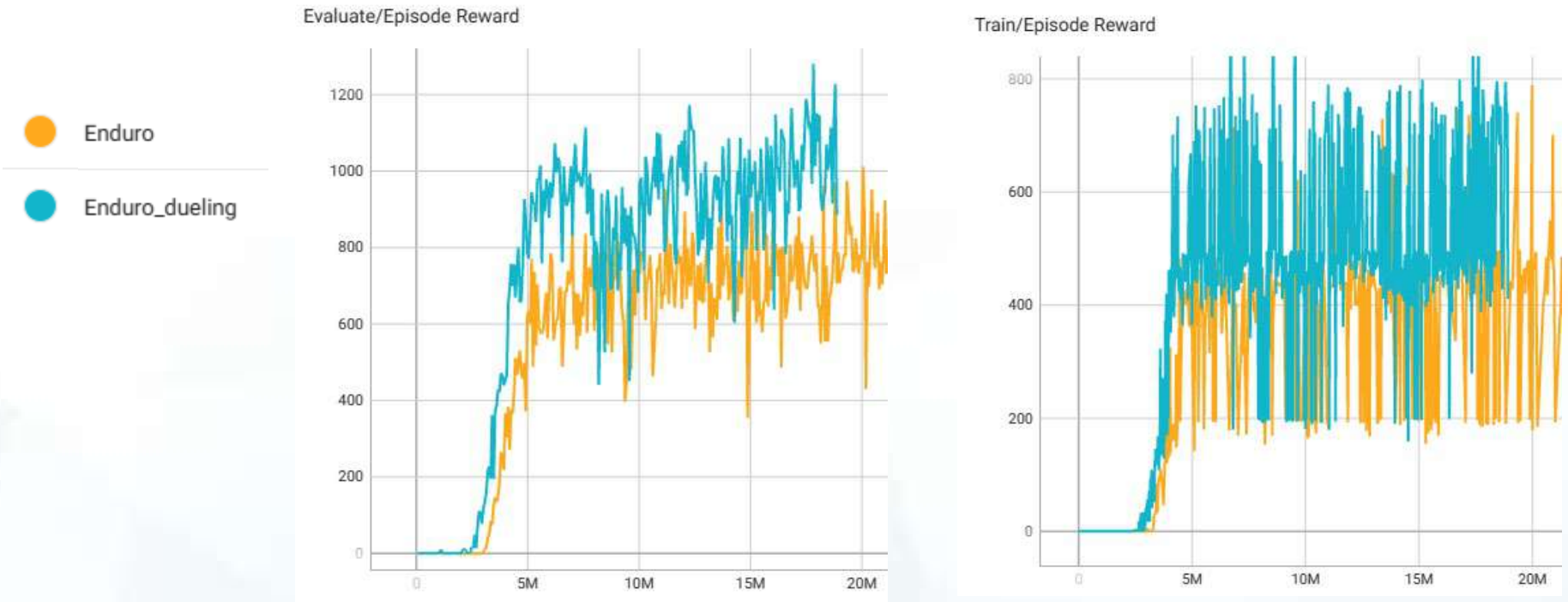


Testing results (5 games):

```
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
```

TODO

- Screenshot of Tensorboard training curve and testing results and put it on the report.



TODO

- Inherit from the "DQNBaseAgent" and override the "decide_agent_actions" and "update_behavior_network" functions.

base_agent.py

```
class DQNBaseAgent(ABC):
    ...

    @abstractmethod
    def decide_agent_actions(...):
        ...

    def update(...):
        ...

    @abstractmethod
    def update_behavior_network(...):
        ...

    def update_target_network(...):
        ...

    def train(...):
        ...

    def evaluate(...):
        ...
```

dqn_agent_atari.py

```
class AtariDQNAgent(DQNBaseAgent):
    ...

    def decide_agent_actions(...):
        ...


    def update_behavior_network(...):
        ...
```

Bonus: Double DQN

- DQN suffers from over-estimation.
- Behavior and Target network.
- Reduce the over-estimation problem.

$$Y_t^{Q} = r_{t+1} + \gamma \max_a Q(S_{t+1}, a|\theta^-)$$

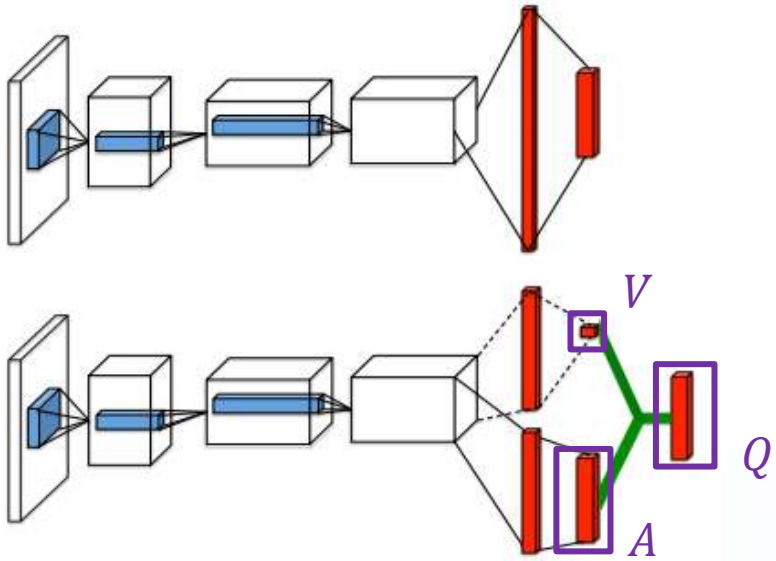
$$Y_t^{DoubleQ} = r_{t+1} + \gamma Q\left(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a|\theta)|\theta^-\right)$$

Bonus: Dueling Networks

- $A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$

  $\rightarrow Q(s_t, a_t) = A(s_t, a_t) + V(s)$

- Constrain the value of A:

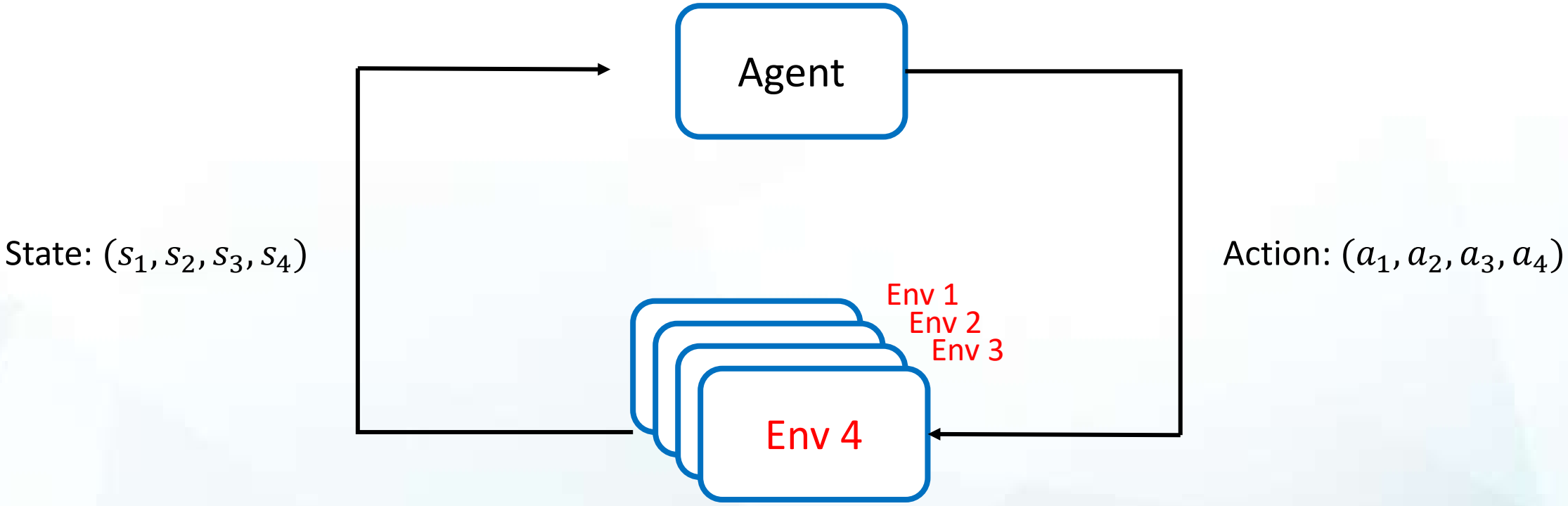


$$Q(s_t, a_t) = V(s) + A(s_t, a_t) - \frac{1}{|A|}\sum_{a_t'} A(s_t, a_t')$$

- Due to the relatively small numerical range of the value A, it is more sensitive to model updates, making it easier for the model to consider the relative changes in relation to other actions.

Bonus: Parallelized Rollout

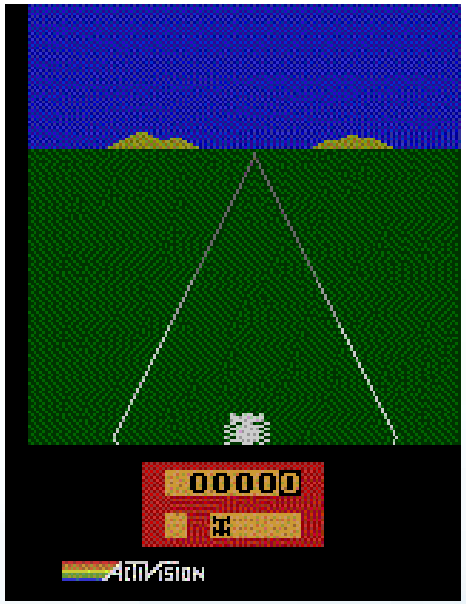
- Parallelly running multiple environments for data collection can help accelerate the training of RL.



Bonus: Enduro-v5

- Introduction:
  - You are a racer in the National Enduro, a long-distance endurance race. You must overtake a certain amount of cars each day to stay on the race. The first day you need to pass 200 cars, and 300 for each following day. The game ends if you do not meet your overtake quota for the day.
- Observation space:
  - The whole image.
- Action space:

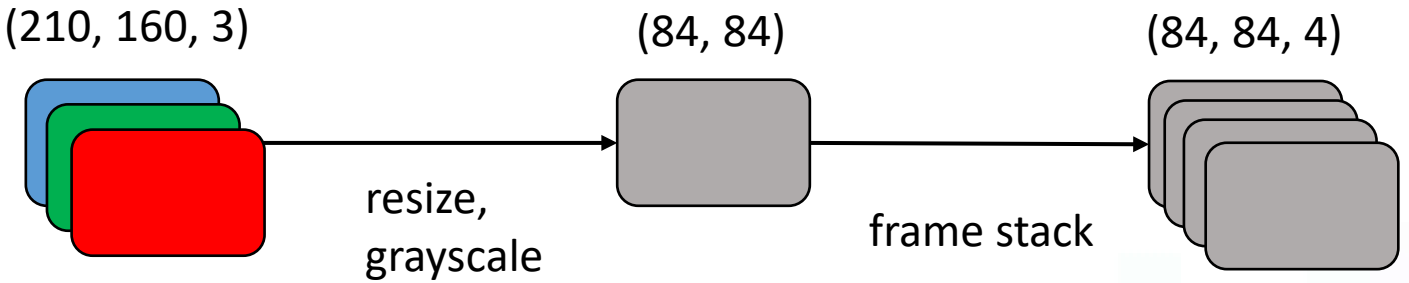
Num	Action
0	NOOP
1	UP
2	RIGHT
3	LEFT
4	DOWN
5	UPRIGHT
6	UPLEFT
7	DOWNRIGHT
8	DOWNLEFT



https://www.gymlibrary.dev/environments/atari/enduro/

Hint

- You can add any trick you want. E.g.:
  - Frame stack
  - Clip reward
  - Grayscale observation
  - ……



- You can use OpenAI Gym Wrapper.
  - https://www.gymlibrary.dev/api/wrappers/

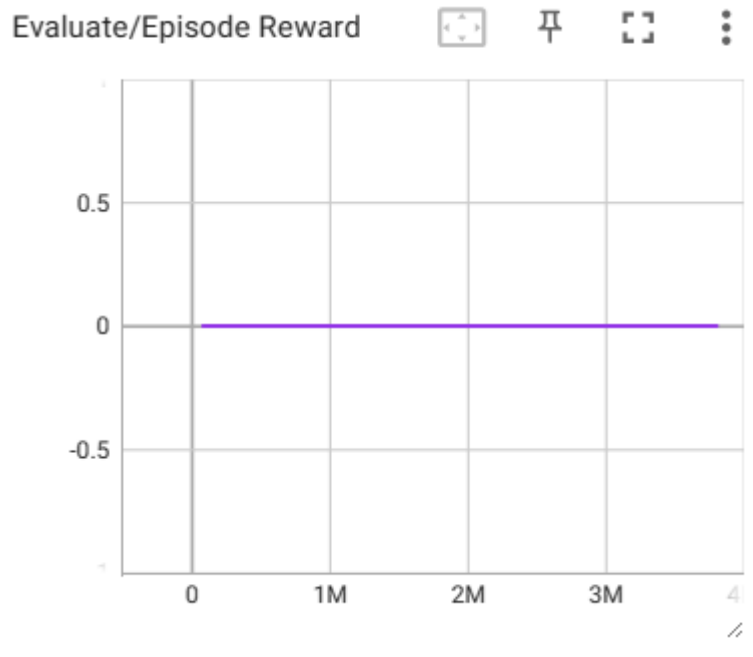
Scoring Criteria

- Your Score = report (30%) + report bonus (20%) + demo performance (50%) + demo questions (20%)
- Report contains two parts:
  - Experimental Results (30%)
    - Screenshot of Tensorboard training curve and testing results on DQN.
  - Experimental Results and Discussion of bonus parts (bonus) (20%).
    - Screenshot of Tensorboard training curve and testing results on Enduro-v5 (10%).
    - Screenshot of Tensorboard training curve and testing results on DDQN, and discuss the difference between DQN and DDQN (3%).
    - Screenshot of Tensorboard training curve and testing results on Dueling DQN, and discuss the difference between DQN and Dueling DQN (3%).
    - 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%).

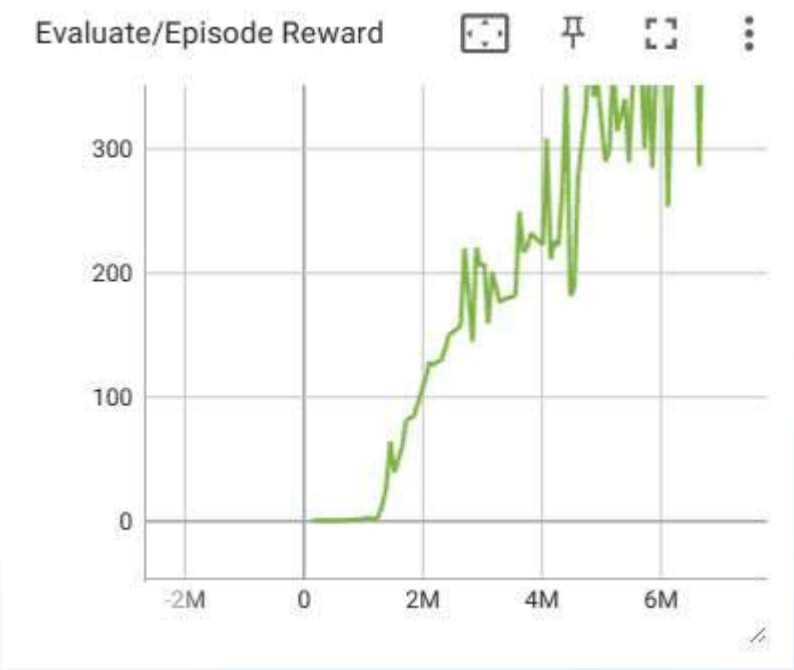
Scoring Criteria

- Screenshot of Tensorboard training curve and testing results and put it on the report.

✖ No score

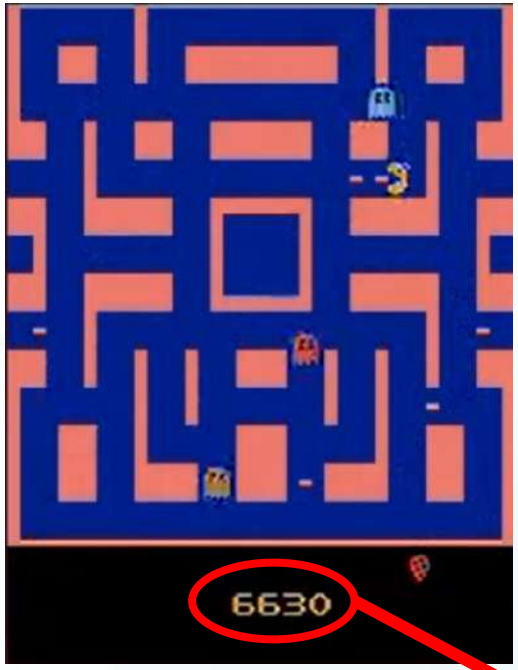


○ Get score (30%)



Scoring Criteria - Demo Performance

- 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.



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

Next Lab (PPO) Scoring Criteria - Demo Performance

- Demo performance – Score table:

Terminated in day 9

Day	Cars	Reward	Points (50%)
1	200	0~200	0
2	300	200~500	10
3	300	500~800	20
4	300	800~1100	25
5	300	1100~1400	30
6	300	1400~1700	35
7	300	1700~2000	40
8	300	2000~2300	45
9	300	2300~2600	50

Tensorboard Remote Server

- ssh -p [your port] -L 6006:localhost:6006 pp037@140.113.215.196
- tensorboard --logdir log/dqn
- Open your browser locally and input 127.0.0.1:6006

Recommended Package Version

- gym 0.26.2
- numpy 1.25.2
- pytorch 2.0.1
- tensorboard 2.14.0
- opencv-python 4.8.0.76
- moviepy 1.0.3

Reminders

- Your network architecture and hyper-parameters can differ from the defaults.
- Ensure the shape of tensors all the time especially when calculating the loss.
- with no_grad() : scope is the same as xxx.detach()
- Be aware of the indentation of hints.

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 DoubleQ-Learning.” AAAI. 2016.
4. OpenAI. “OpenAI Gym Documentation.” Retrieved from Getting Started with Gym: https://gym.openai.com/docs/ .
5. PyTorch. “Reinforcement Learning (DQN) Tutorial.” Retrieved from PyTorch Tutorials: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html .