

LAB 5

TA 陳昱丞

yucheng.cs11@nycu.edu.tw

Deadline: 2023/8/24 (Thu) 12:00

No Demo

In this lab,

**Must use sample code,
otherwise no credit.**

Outline

Part 1: Low-dimensional Observation

Solve **LunarLander-v2** using **DQN** (30%)

Solve **LunarLanderContinuous-v2** using **DDPG** (30%)

Bonus: Implement **DDQN** (5%)

Bonus: Implement **TD3** (10%)

Part 2: High-dimensional Observation

Solve **BreakoutNoFrameskip-v4** using **DQN** (40%)

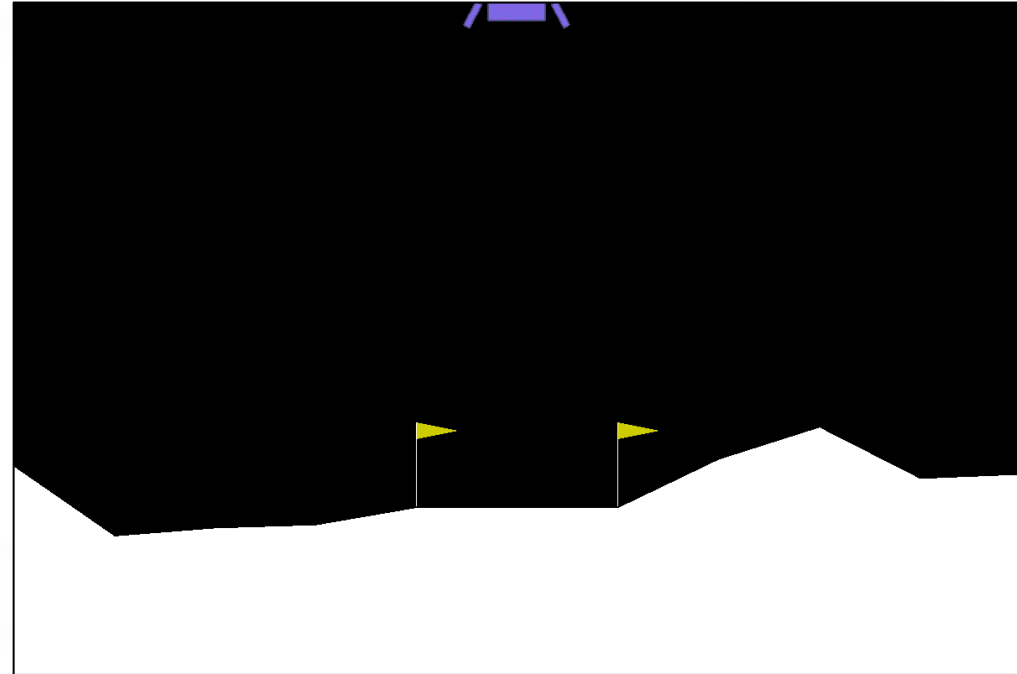
Report:

Result (0% **but necessary**)

Bonus: Questions (10%)

LunarLander-v2

- Observation [8]
 1. Horizontal Coordinate
 2. Vertical Coordinate
 3. Horizontal Speed
 4. Vertical Speed
 5. Angle
 6. Angle Speed
 7. If first leg has contact
 8. If second leg has contact
- Action [4]
 1. No-op
 2. Fire left engine
 3. Fire main engine
 4. Fire right engine



- Action [2] (Continuous)
 - Main engine: -1 to 0 off, 0 to +1 throttle from 50% to 100% power. Engine can't work with less than 50% power
 - Left-right: -1.0 to -0.5 fire left engine, +0.5 to +1.0 fire right engine, -0.5 to 0.5 off

Deep Q-Network (DQN)

Target Q:

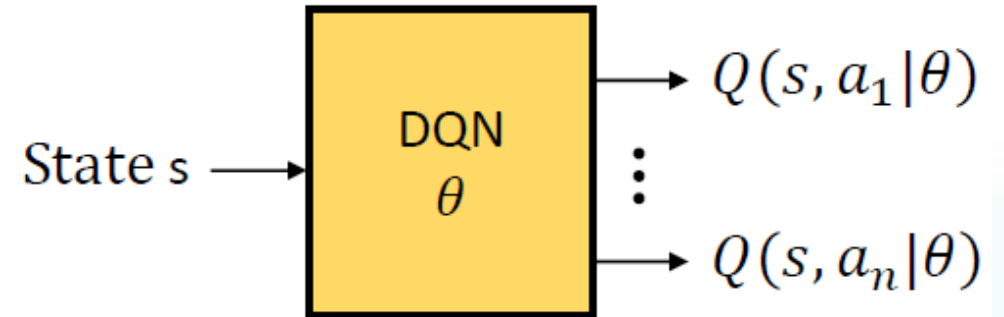
$$Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a' | \theta)$$

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 $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ 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

Deep Deterministic Policy Gradient (DDPG)

Consider **continuous actions** and deterministic policy: $a = \pi_\theta(s)$

Deterministic Policy Gradient Theorem:

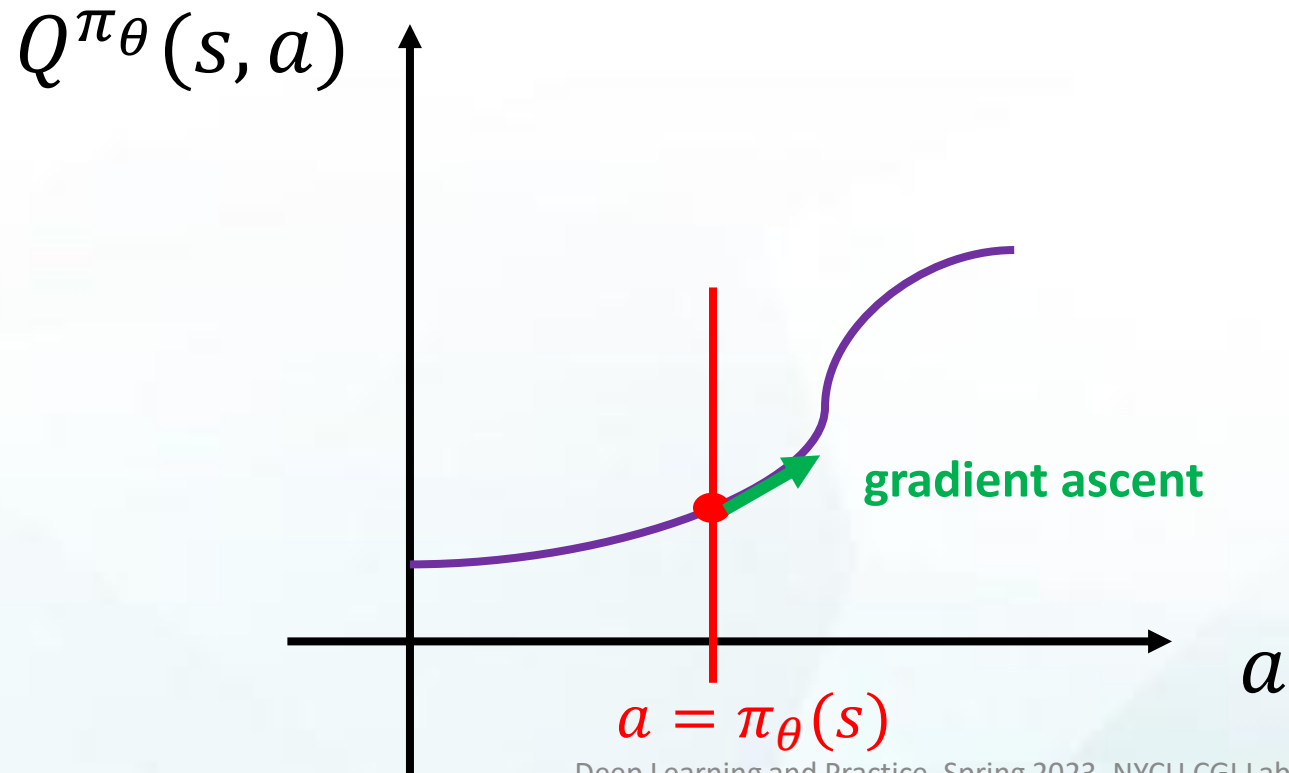
$$\nabla_\theta V^{\pi_\theta}(\mu) = \frac{1}{1-\gamma} E_{s \sim d_\mu^{\pi_\theta}} [\nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s, a) |_{a=\pi_\theta(s)}]$$

Off-policy Deterministic Policy Gradient:

$$\nabla_\theta J_\beta^{\pi_\theta} \approx E_{\underline{s \sim d_\mu^\beta}} [\nabla_\theta \pi_\theta(s) \nabla_a Q^{\pi_\theta}(s, a) |_{a=\pi_\theta(s)}]$$

Deep Deterministic Policy Gradient (DDPG)

$$\nabla_{\theta} J_{\beta}^{\pi_{\theta}} \approx E_{s \sim d_{\mu}^{\beta}} [\nabla_{\theta} \pi_{\theta}(s) \nabla_a Q^{\pi_{\theta}}(s, a) |_{a=\pi_{\theta}(s)}]$$



Deep Deterministic Policy Gradient (DDPG)

Algorithm 1 – DDPG:

```
Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ 
Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$ 
Initialize replay buffer  $R$ 
for  $episode = 1, M$  do
    Initialize a random process  $N$  for action exploration
    Receive initial observation state  $s_1$ 
    for  $t = 1, T$  do
        Select action  $a_t = \mu(s_t|\theta^\mu) + N_t$  according to the current policy and exploration noise
        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$ 
        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$ 
        Sample random minibatch of  $N$  transitions  $(s_j, a_j, r_j, s_{j+1})$  from  $R$ 
        Set  $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$ 
        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$ 
        Update the actor policy using the sampled gradient:
            
$$\nabla_{\theta^\mu} \mu|s_i \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|s_i$$

        Update the target networks:
            
$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

            
$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

    end for
end for
```

2 Behavior and 2 target networks

Action drawn from deterministic policy with exploration

Experience replay

Update actor and critic

Update target networks (soft update)

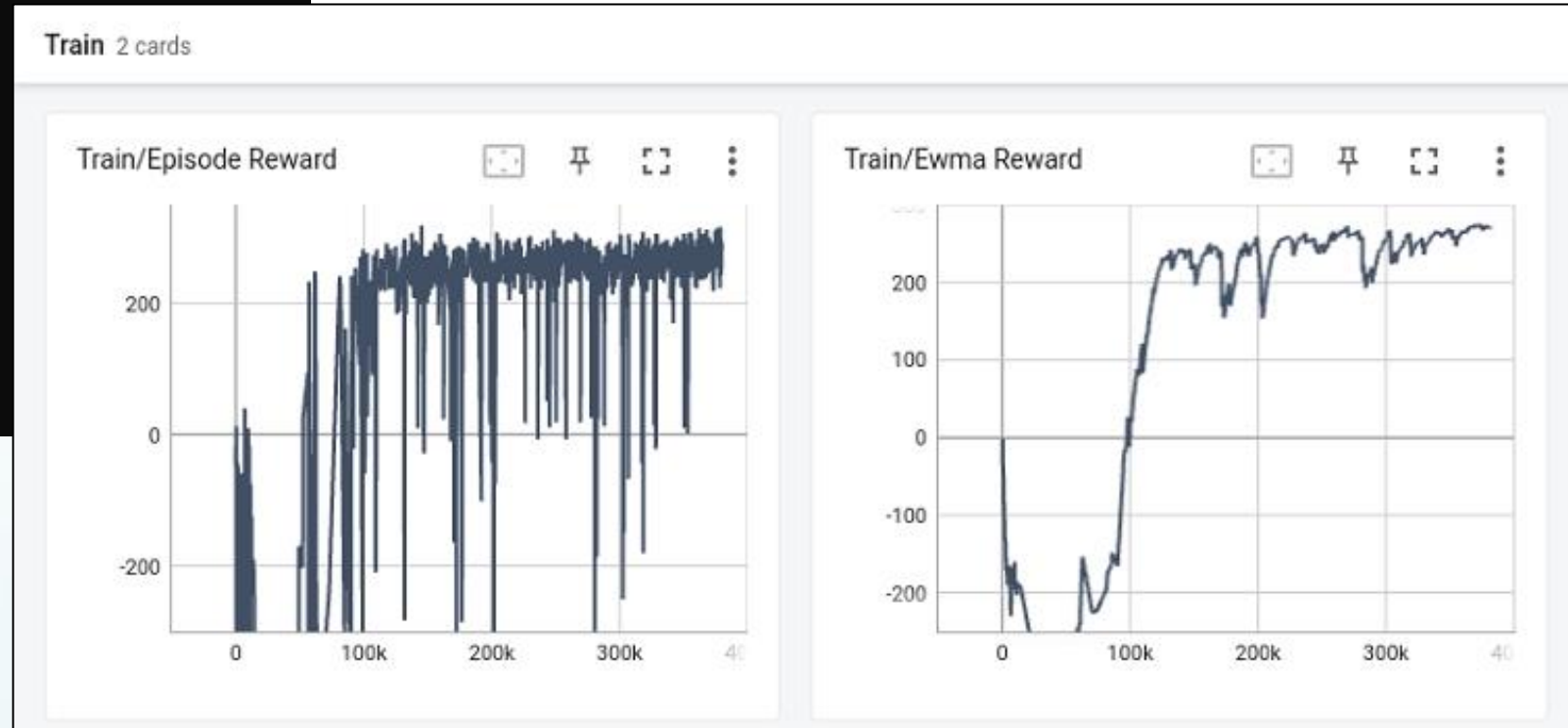
TODO

- Solve LunarLander-v2 using DQN.
- Solve LunarLanderContinuous-v2 using DDPG.
- Find the #TODO comments and hints, remove the raise NotImplementedError.
- Screenshot your tensorboard and testing results and put it on the report.

TODO

- Screenshot your tensorboard and testing results and put it on the report.

```
~/DLP/lab6$ python3 dqn.py --test_only
Start Testing
episode 1: 264.02
episode 2: 237.68
episode 3: 257.86
episode 4: 298.76
episode 5: 267.87
episode 6: 315.03
episode 7: 302.27
episode 8: 282.32
episode 9: 277.29
episode 10: 262.62
Average Reward 276.57379065836386
```



Scoring Criteria

- DQN performance (30%)
 - Bonus: implement DDQN (5%)
- DDPG performance (30%)
 - Bonus: implement TD3 (10%)
- Run test 10 times, average score:

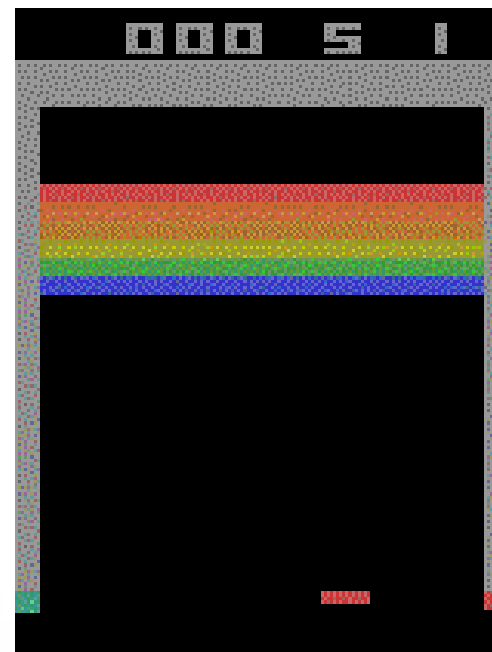
```
~/DLP/lab6$ python3 dqn.py --test_only
Start Testing
episode 1: 264.02
episode 2: 237.68
episode 3: 257.86
episode 4: 298.76
episode 5: 267.87
episode 6: 315.03
episode 7: 302.27
episode 8: 282.32
episode 9: 277.29
episode 10: 262.62
Average Reward 276.57379065836386
```

average score	points
≤ 0	0
0 ~ 100	5
100 ~ 150	10
150 ~ 200	20
≥ 200	30

BreakoutNoFrameskip-v4

- Observation space:
 - The whole image
- Action space:

Num	Action
0	NOOP
1	FIRE
2	RIGHT
3	LEFT

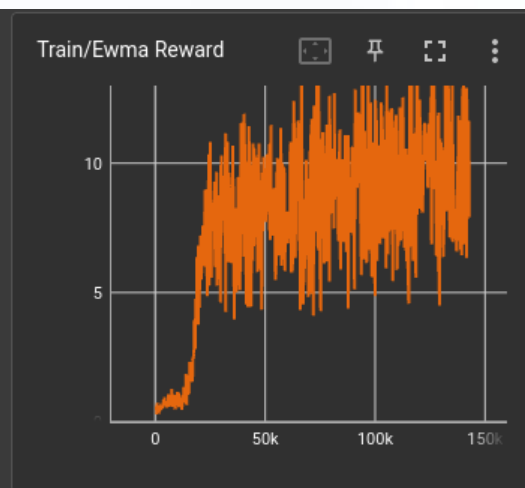
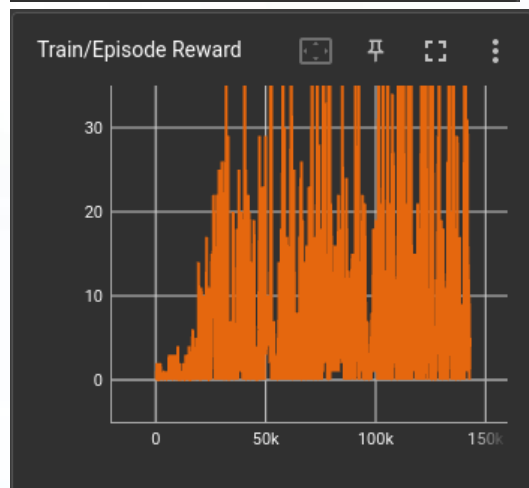
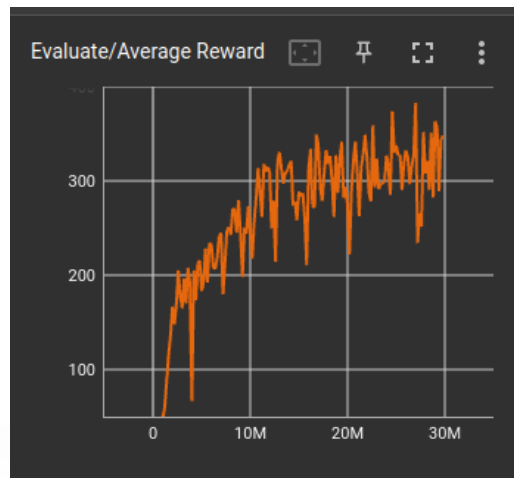


TODO

- Solve BreakoutNoFrameskip-v4 using DQN.
- You can use any trick you want.
- Screenshot your tensorboard and testing results and put it on the report.

TODO

- Screenshot your tensorboard and testing results and put it on the report.



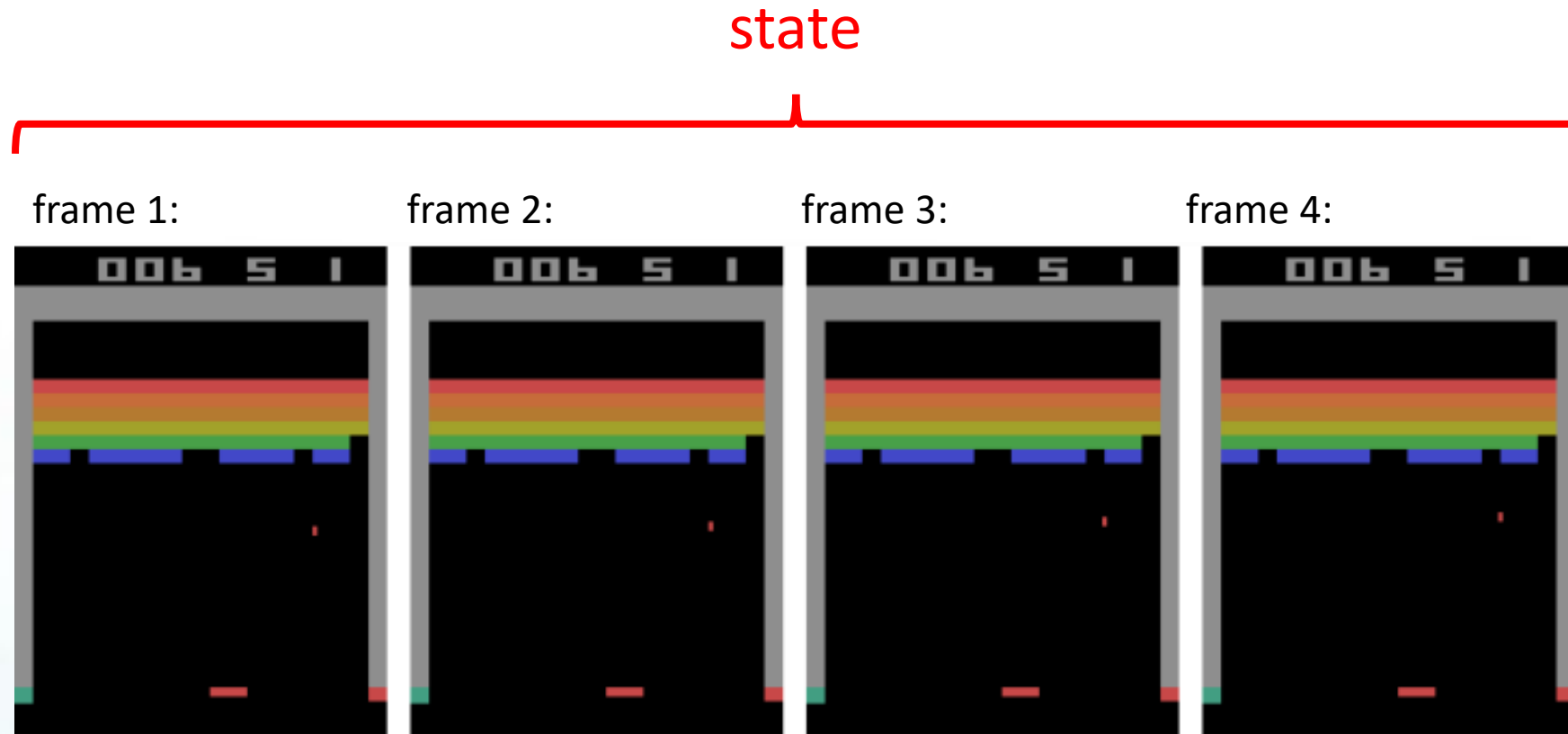
```
Start Testing
episode 1: 421.00
episode 2: 414.00
episode 3: 828.00
episode 4: 427.00
episode 5: 396.00
episode 6: 430.00
episode 7: 424.00
episode 8: 798.00
episode 9: 433.00
episode 10: 427.00
Average Reward: 499.80
```


Hint: Trick 1

- Use `make_Atari()` and `wrap_deepmind()` provided by OpenAI baselines.
 - `atari_wrappers.py`
- Remember to set `episode_life=False, clip_rewards=False` while testing.

Hint: Trick 2

- Stack a sequence of four frames together.



Scoring Criteria

- Performance (40%)
- Run test 10 times, $\min(5 * \sqrt{\text{average score}}, 100)$.
- For example:

```
Start Testing
episode 1: 421.00
episode 2: 414.00
episode 3: 828.00
episode 4: 427.00
episode 5: 396.00
episode 6: 430.00
episode 7: 424.00
episode 8: 798.00
episode 9: 433.00
episode 10: 427.00
Average Reward: 499.80
```

$$\sqrt{499.8} = 22.35$$

$$5 * 22.35 = 111.75$$

$$\min(111.75, 100) = 100$$

$$40 * 100\% = 40 \rightarrow \text{points you get}$$

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`

Package Version

- gym 0.15.7
- numpy 1.22.3
- pytorch 1.7.1
- tensorboard 2.10.0

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.
- When testing DDPG, action selection need **NOT** include the noise.

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. Lillicrap, Timothy P. et al. “Continuous control with deep reinforcement learning.” CoRRabs/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. OpenAI. “OpenAI Wiki for Pendulum v0.” Retrieved from Github: <https://github.com/openai/gym/wiki/Pendulum-v0> .
8. PyTorch. “Reinforcement Learning (DQN) Tutorial.” Retrieved from PyTorch Tutorials: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html .