Lab 4: Twin Delayed DDPG (TD3)

Lab Objective:

In this lab, you will learn and implement the Twin Delayed DDPG (TD3) algorithm by solving CarRacing-v2.

Important Date:

- Submission deadline: 11/24 (Sun) 23:59

Turn in:

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

Notice: zip all files with name "RL_LAB4_StudentId_Name.zip",
e.g.: 「RL_LAB4_312551033_邱恆毅.zip」

```
RL_LAB4_312551033_邱恆毅
├── 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 the behavior network and target network.
- Understand the mechanism of the experience replay buffer.
- Learn to construct and design neural networks.
- Understand "soft" target updates.
- Understand the difference between DDPG and TD3.

Requirements:

- Implement TD3
  - Construct neural networks of both Actor and Critic.
  - Select action according to the actor and the exploration noise.
  - Update critic by minimizing the loss.
  - Update actor using the sampled policy gradient.
  - Update target network softly.
  - Understand the mechanism of actor-critic.
  - Understand the effect of three tricks added to DDPG.

Game Environment – CarRacing-v2:

- Introduction: The easiest control task to learn from pixels - a top-down racing environment. The generated track is random every episode. Some indicators are shown at the bottom of the window along with the state RGB buffer. From left to right: true speed, four ABS sensors, steering wheel position, and gyroscope.
- Observation: State consists of 96x96 pixels.
- Action [3]: steering (-1 is full left, +1 is full right), gas (0~1), and breaking (0~1).

Algorithm – DDPG algorithm:

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' 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 $a_t = \mu(\theta^\mu) + N_t$ 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 $y_i = r_i + \gamma Q'(\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i \left(y_i - Q(\theta^Q)\right)^2$
    Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|s_i \approx \frac{1}{N}\sum_i \nabla_a Q(\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(\theta^\mu)|s_i$$

    Update the target networks:

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

  end for
end for

Algorithm – TD3 algorithm:

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta_1' \leftarrow \theta_1, \theta_2' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

2. Delayed Policy Updates

3. Target Policy Smoothing Regularization

Implementation Details – CarRacing-v2:

Network Architecture

- Please refer to the sample code for details.

Training Hyper-Parameters

- Memory capacity (experience buffer size): 5000
- Batch size: 32
- Warmup steps: 1000
- Optimizer: Adam
- Learning rate: 4.5e-5
- Gamma (discount factor): 0.99
- Tau: 0.005
- Target and actor update frequency: 2

You can tune the hyperparameter yourself.

Bonus – Ablation Study:

1. Impact of Twin Q-Networks:
   - Compare the performance of using twin Q-networks and single Q-networks in TD3, and explain.
2. Target Policy Smoothing:
   - Compare the impact of enabling and disabling target policy smoothing in TD3. Attempt to disable the component of target policy smoothing in TD3 and observe whether performance decreases, and explain.
3. Delayed Policy Update Mechanism:
   - Study the effects of using a single delayed update or adding more delayed update steps in TD3. Increase or decrease the number of delayed update steps and compare the results, and explain.
4. Action Noise Injection:
   - Compare the effects of adding different levels of action noise (exploration noise) in TD3. Try increasing or decreasing the magnitude of action noise in TD3 and observe how it affects algorithm convergence speed and stability, and explain.
5. Reward Function Design:
   - Design a different reward function that works better than the original one, and explain why it's effective.

Scoring Criteria:

Show your results, otherwise no credit will be granted.

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

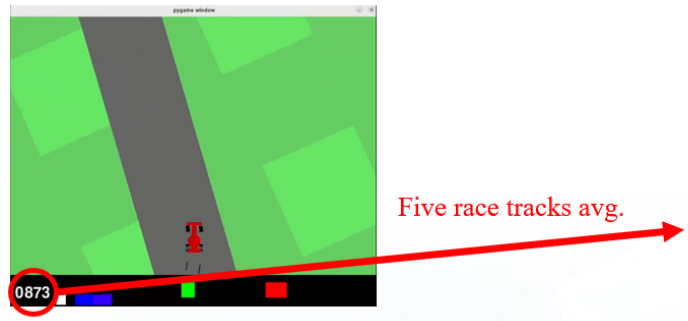
- **Report contains two parts:**
  - **Experimental Results (30%)**
    - **(1)** Screenshot of Tensorboard training curve and testing results on TD3.
  - **Experimental Results and Discussion of bonus parts (Impact of Twin Q-Networks, Target Policy Smoothing, Delayed Policy Update Mechanism, Action Noise Injection) (bonus) (30%)**
    - **(1)** Screenshot of Tensorboard training curve and compare the performance of using twin Q-networks and single Q-networks in TD3, and explain (5%).
    - **(2)** Screenshot of Tensorboard training curve and compare the impact of enabling and disabling target policy smoothing in TD3, and explain (5%).

(3) Screenshot of Tensorboard training curve and compare the impact of delayed update steps and compare the results, and explain (5%).

(4) Screenshot of Tensorboard training curve and compare the effects of adding different levels of action noise (exploration noise) in TD3, and explain (5%).

(5) Screenshot of Tensorboard training curve and compare your reward function with the original one and explain why your reward function works better (10%).

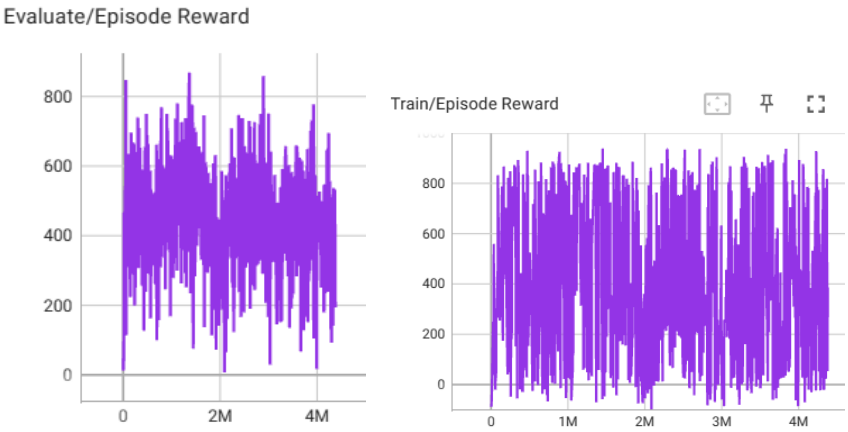
- **Demo Performance (50%):**
  - Test your best model for five race tracks. Seeds of five tracks will be given on demo day.
  - You have to show the video while testing. You can use env.render() or save video function to achieve this.
  - Demo performance – Score table (50%):



Reward	Points (50%)
0~100	0
100~199	10
200~299	20
300~399	25
400~499	30
500~599	35
600~699	40
700~799	45
800~	50

Examples of Tensorboard training curve and testing results:

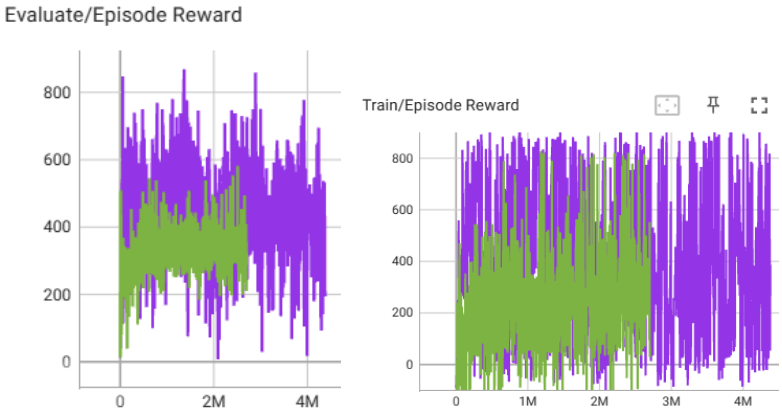
- Training curve:



- Testing results (10 games):

```
Episode: 1      Length: 999     Total reward: 874.44
Episode: 2      Length: 999     Total reward: 883.05
Episode: 3      Length: 999     Total reward: 797.44
Episode: 4      Length: 999     Total reward: 679.18
Episode: 5      Length: 999     Total reward: 866.78
Episode: 6      Length: 999     Total reward: 888.97
Episode: 7      Length: 751     Total reward: 924.80
Episode: 8      Length: 999     Total reward: 883.33
Episode: 9      Length: 999     Total reward: 614.81
Episode: 10     Length: 999     Total reward: 878.34
average score: 829.1142945389436
```

- Training curve (comparison):



References:

[1] Lillicrap, Timothy P. et al. “Continuous control with deep reinforcement learning.” CoRR abs/1509.02971 (2015).
[2] Silver, David et al. “Deterministic Policy Gradient Algorithms.” ICML (2014).
[3] OpenAI. “OpenAI Gym Documentation.” Retrieved from Getting Started with Gym: https://gym.openai.com/docs/.
[4] PyTorch. “Reinforcement Learning (DQN) Tutorial.” Retrieved from PyTorch Tutorials: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html.
[5] Dankwa, Stephen, and Wenfeng Zheng. "Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent." Proceedings of the 3rd international conference on vision, image and signal processing. 2019.