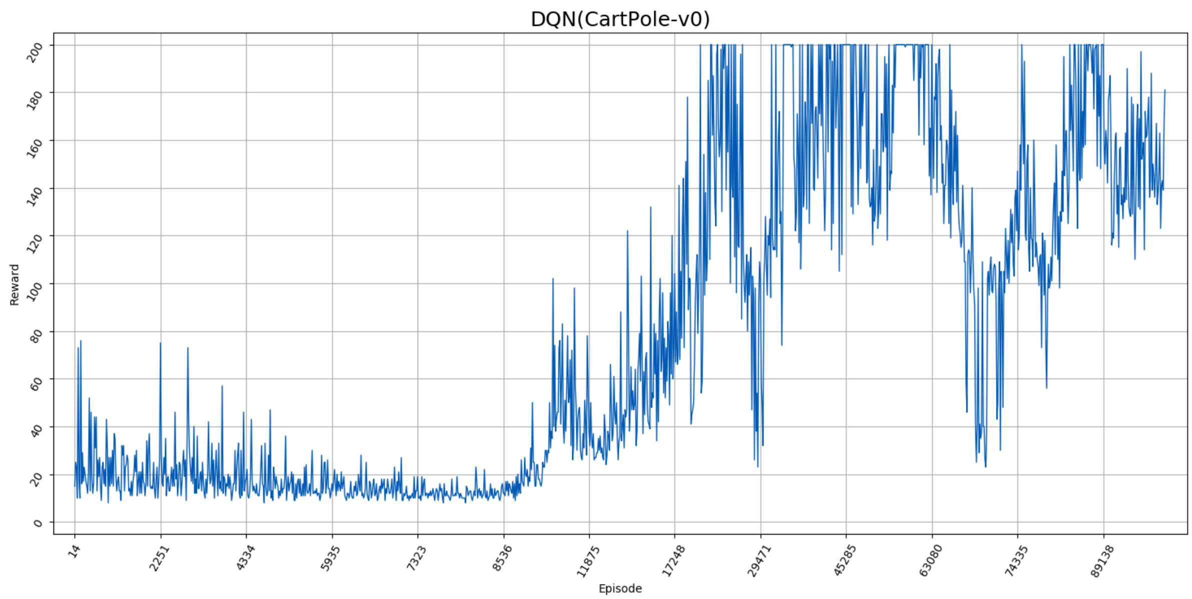
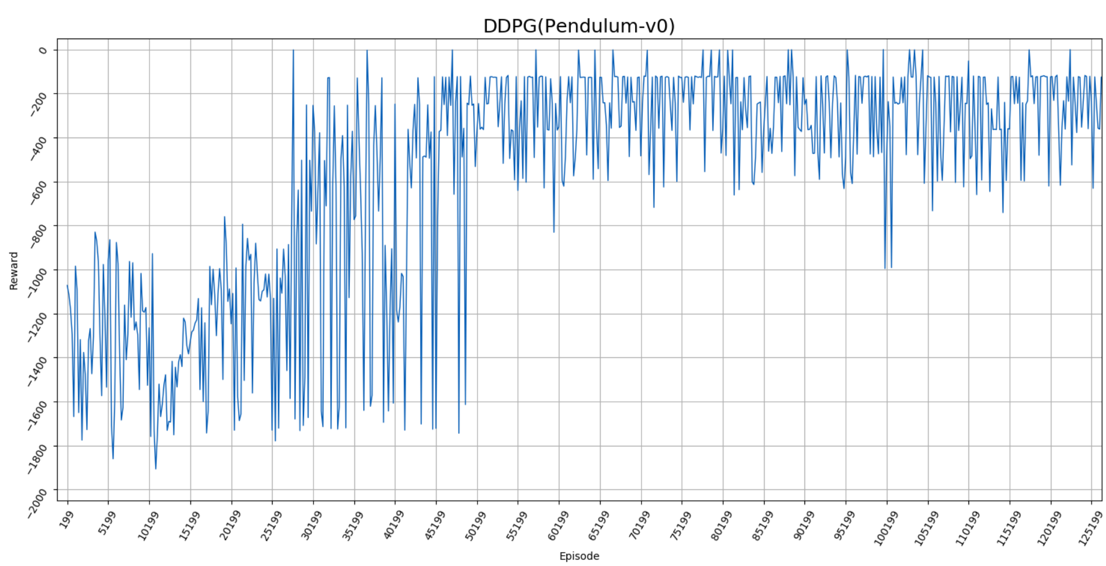
Lab8 Deep Q-Learning and Deep Deterministic Policy Gradient

0516054 劉雨恩

1. A plot shows episode scores of at least 1000 training episodes in the game CartPole-v0



2. A plot shows episode rewards of at least 10000 training episodes in the game Pendulum



3. Describe your implement/adjustment of the network structure & each loss function

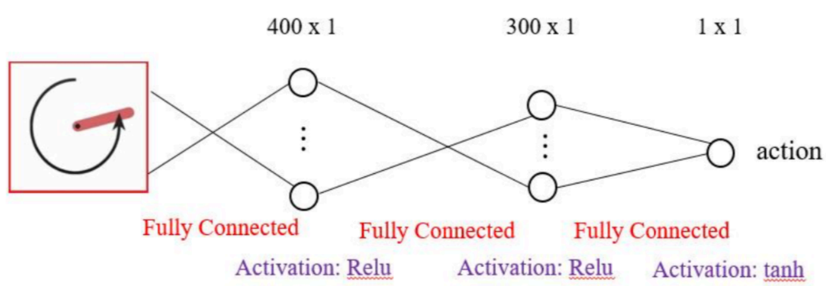
(1) Network structure

i. DQN

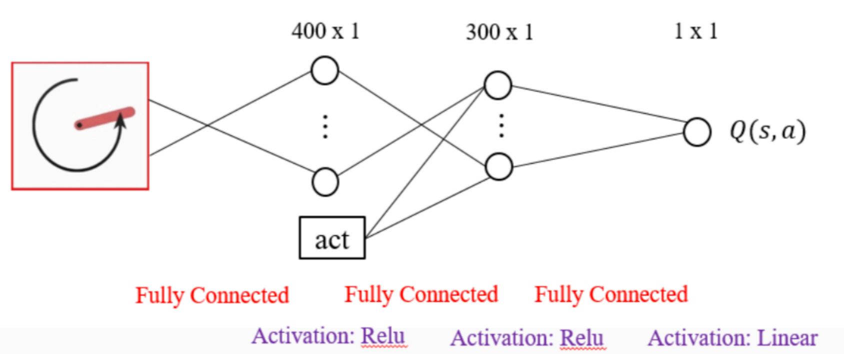
- Input: Observation (4 elements, not images)
- First layer: fully connected layer (ReLU)
  - input: 4, output: 32
- Second layer: fully connected layer
  - input: 32, output: 2

ii. DDPG

- Actor



- Critic



(2) Loss function

真實值(下一個state採取最好action的Q值)和估計值(當前state和action的Q值)取mean square error便為loss。

i. DQN： $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$

ii. DDPG： $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$

4. Describe how you implement the training process of deep Q-learning

(1) 初始Q network

(2) 開始重複執行每個episode

(3) 每episode都需初始一開始的state

(4) 而後開始依照將state放進Q network得到的action(使用epsilon-greedy)，得到reward和next state

(5) 利用reward、真實值和估計值的loss(如3.(2)所述)更新Q network，並將state更新為next state

5. Describe the way you implement of epsilon-greedy action select method

在epsilon的機率會選到最優的action，1-epsilon的機率選擇random action。

```
if self._total_steps < config.exploration_steps \
        or np.random.rand() < config.random_action_prob():
    action = np.random.randint(0, len(q_values))
else:
    action = np.argmax(q_values)
```

並且使epsilon的機率在總共的training step中均勻的從1 decay到0.1。

```
config.random_action_prob = LinearSchedule(1.0, 0.1, 1e5)
```

6. Explain the mechanism of critic updating

$$L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$

藉由minimize真實值和估計值之間的mean square error(如3.(2)所述)，來update critic。

7. Explain the mechanism of actor updating

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

藉由將Q function對policy θ做gradient(如上述式子，由chain rule而來)，來update actor。

8. Describe how to calculate the gradients

```
phi = self.network.feature(states)
action = self.network.actor(phi)
policy_loss = -self.network.critic(phi.detach(), action).mean()
```

計算sample出來走過state和action的Q value，取mean值再加負號當作policy的loss。update policy使loss最小化，也就是使policy的Q value能夠達到最大化。

9. Describe how the code work (the whole code)

(1) DQN

i. Parameters setting

- Optimizer: RMSprop
- Learning Rate: 0.0005
- Epsilon: 1 → 0.1 (decay evenly in 1e5 steps)
- Batch Size: 128

- Experience buffer size (Memory capacity): 5000
- Gamma (Discount Factor): 0.95
- Training Episode: 100000
- Update target network every 50 iterations
- Game environment: CartPole-v0

ii. Algorithm

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 $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ 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

(2) DDPG

i. Parameters setting

- Optimizer: Adam
- Learning Rate (Actor): 0.0001
- Learning Rate (Critic): 0.001
- Tau: 0.001

- Batch Size: 64
- Experience buffer size = 10000
- Gamma (Discount Factor): 0.99
- Total training episode: 150000
- Game environment: Pendulum-v0

ii. Algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ 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 $a_t = \mu(s_t|\theta^\mu) + N_t$ 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: $L = \frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
    Update the actor policy using the sampled gradient:
$$\nabla_{\theta^\mu}\mu|s_i \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|s_i$$
    Update the target networks:
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$$
  end for
end for

10. Other study or improvement for the project

可以增進DDPG表現的其他方法：

(1) DDPG from Demonstrations (DDPGfD)

這個方法對DDPG做了以下五種改變：

i. Transitions from a human demonstrator are added to the replay buffer

ii. Prioritized replay

iii. A mix of 1-step return and N-step return losses

iv. Learning multiple times per environment steps

v. L2 regularization losses on the weights of the critic and the actor

(2) Distributed Distributional DDPG (D4PG)

這個方法對DDPG做了以下四種改變：

i. Distributed parallel actors：利用threading大大節省training時間。

ii. Distributional critic：相對於只給予一個mean，給予分佈更能得到更多資訊。(比如選擇雖然mean較小，但variance較小，得到很差的reward風險會較小)

iii. N-step returns

iv. Prioritization of the experience replay

(3) Reference

i. DDPGfD：https://arxiv.org/pdf/1707.08817.pdf

ii. D4PG：https://github.com/msinto93/D4PG

11. Performance

(1) [CartPole-v0] Average reward during 100 testing episodes

```
2019-06-05 18:36:51,418 - root - INFO: steps 60000, episodic_return_test 200.00(0.00)
```

(2) [Pendulum-v0] Highest episode reward during training

```
2019-06-04 00:32:03,783 - root - INFO: steps 568599, episodic_return_train -0.17162123767493354
```