1. Learning algorithm

1.1. Overview of the technique

The agent is implemented using the Twin-delayed deep deterministic policy gradient (TD3 for short) method. The paper for this algorithm can be found here:
[Addressing Function Approximation Error in Actor-Critic Methods (arxiv.org)]
This algorithm is an upgraded version of the Deep deterministic policy gradient (DDPG). A quick recap, DDPG is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. DDPG is an off-policy algorithm and can only be used in continuous action space.
TD3 added these following extensions to DDPG:

- Use Clipped Double Q-learning technique. Concretely, TD3 use 2 critics network to calculate Q-value and the one with smaller value will be used to perform loss update
- Use "Delayed" Policy Update. TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates
- Apply Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action

These changes resolve DDPG problem of overestimating Q-value as well as hyperparameters tuning

The algorithm in detail:

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      if $j \mod$ `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

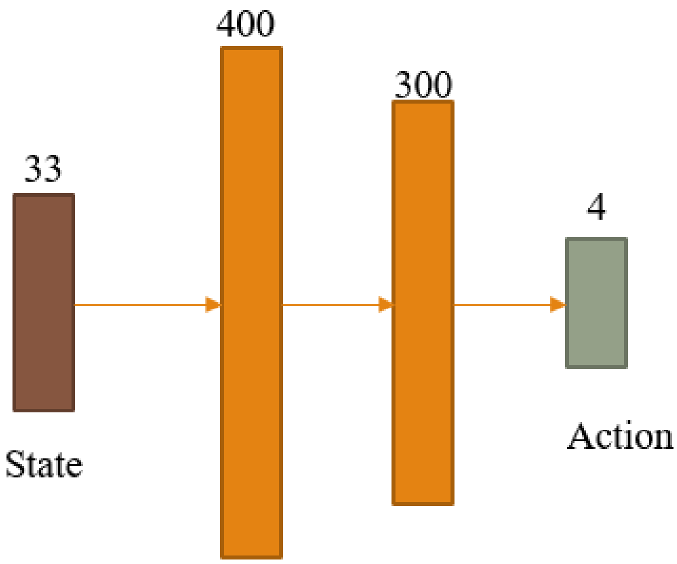
        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta$$

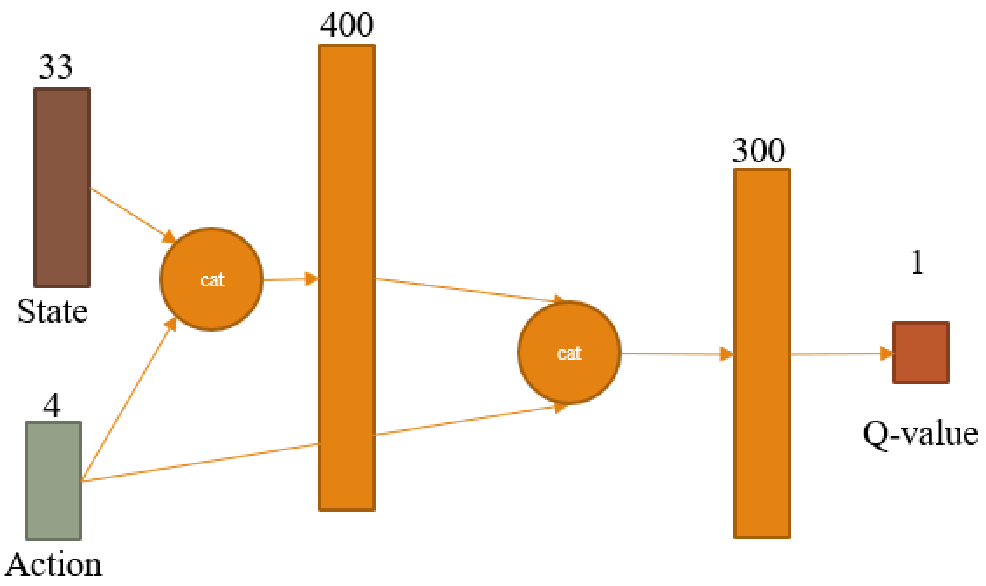
      end if
    end for
  end if
until convergence

1.2. Network architecture

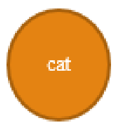
The network architecture for actor and critic are described below (the number is # neurons at each hidden layers)



Actor network structure



Critic network structure

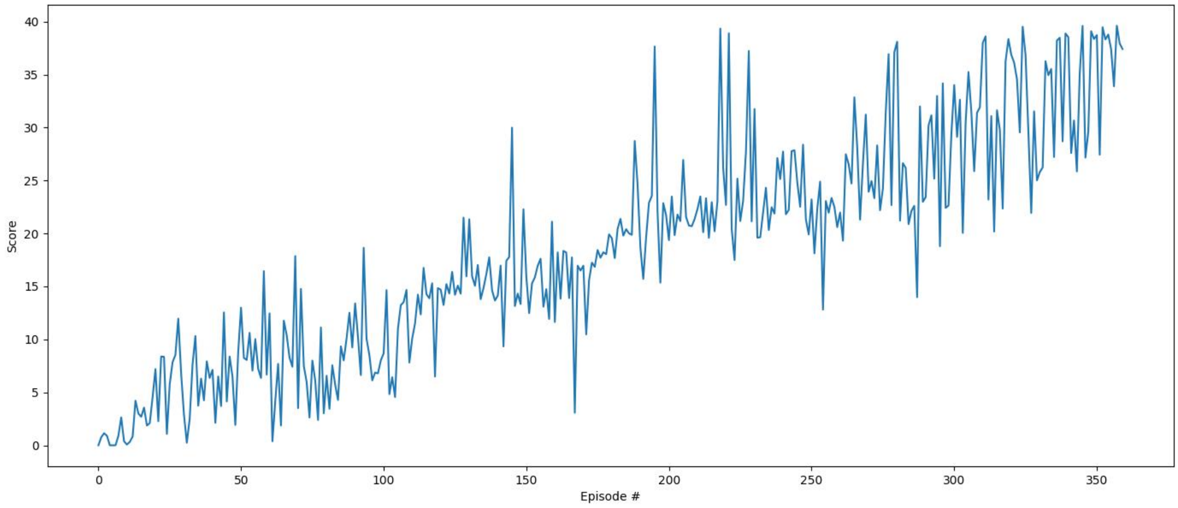


: Concatnate

1.3. Hyperparameters select

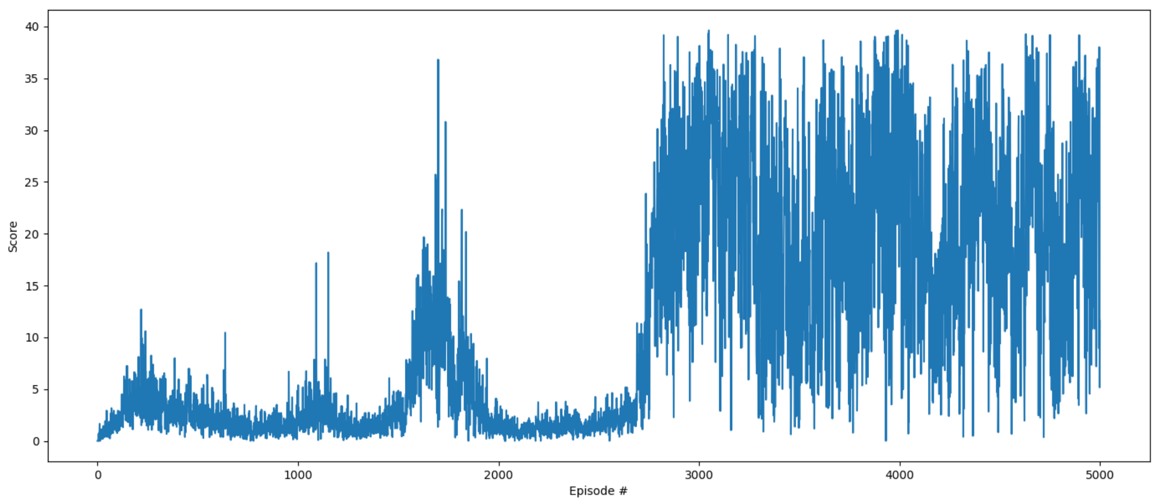
Name	Value	Meaning
seed	73	Random seed for stable result
actor_lr	0.0001	Learning rate for actor's Adam optimizer
critic_lr	0.0001	Learning rate for critic's Adam optimizer
gamma	0.99	Discount factor
tau	0.005	Soft update target network hyperparameter
buffer_size	1000000	Number of transitions stored in replay buffer
batch_size	100	Batch size for learning phase of the agent

2. Plot of rewards



3. Ideas for future works

The result I have shown is quite good, which solve the problem in around 200 episodes. To be honest, I have played around with DDPG before TD3 and the result is terrible. I cannot solve the problem with 5000 episodes. Here is the plot of DDPG:



Although the result I got from TD3 is good enough, I think some modifications will make the result even better:

- Tuning hyperparameters such as learning rate and the architecture of the actor and critic network probably enhance the learning ability of the agent
- Moreover, many other algorithms can be used in order to help the agent learn better such as Distributed Distributional DDPG (D4PG), Proximal Policy Optimisation (PPO) or Soft Actor Critic (SAC)