Twin-Delayed DDPG (TD3) with Bipedal Walker

Chien-Te Lee

2020/8/20

Outline

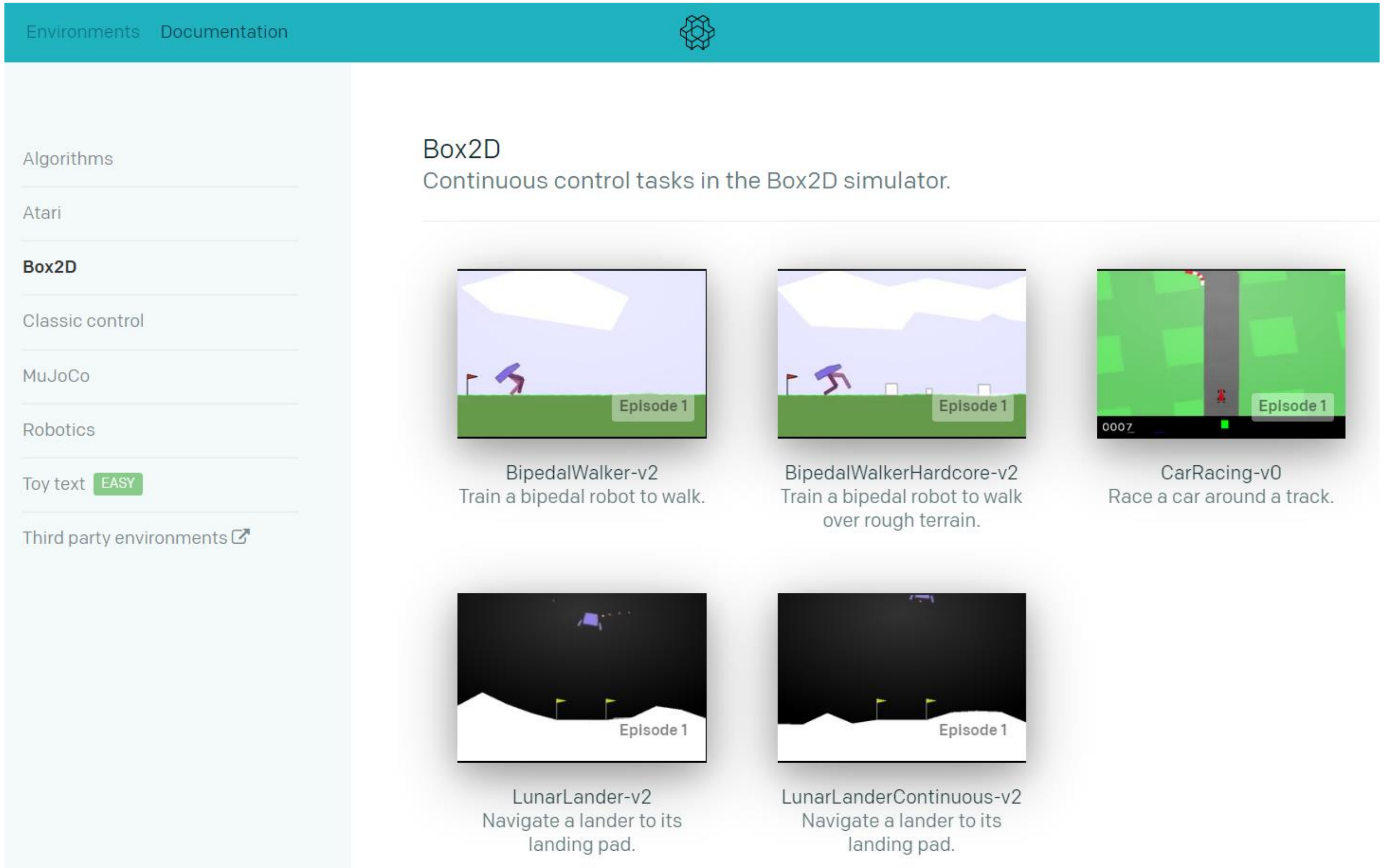
- **OpenAI Gym environment**
- **Model Architecture**
- **Algorithm**
- **Result**
- **Reference**

Outline

- **OpenAI Gym environment**
- **Model Architecture**
- **Algorithm**
- **Result**
- **Reference**

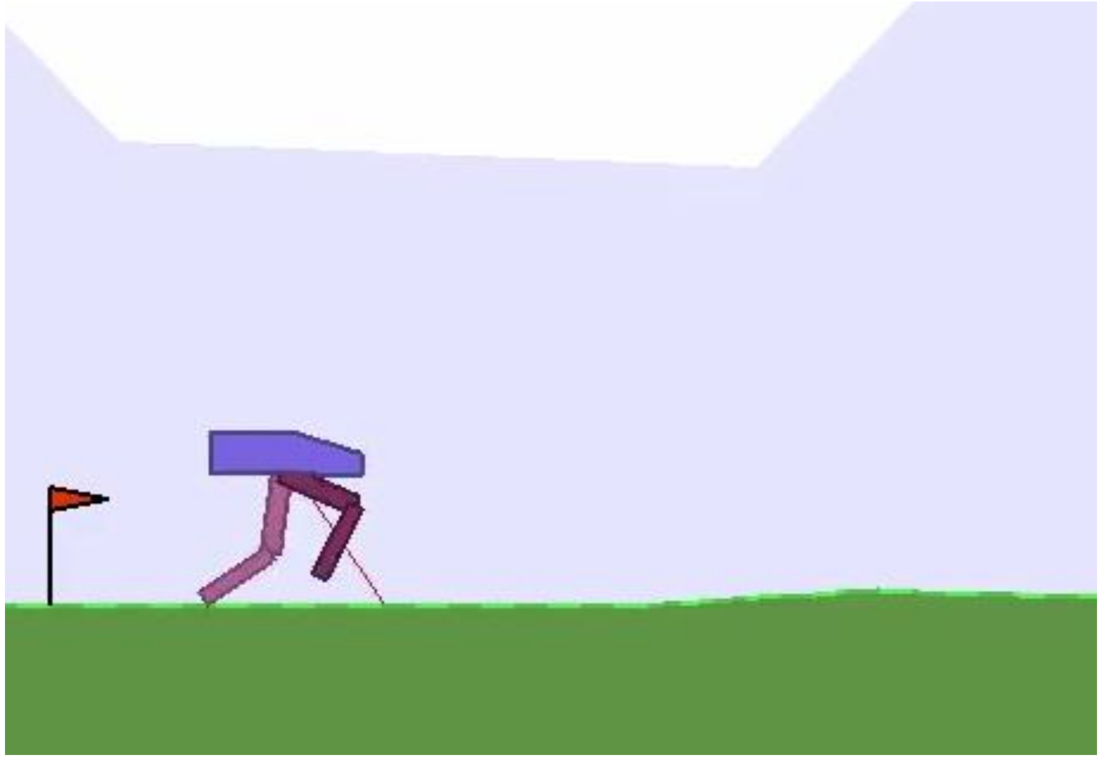
OpenAI Gym environment

- OpenAI Gym is a toolkit for developing and comparing reinforcement learning (RL) algorithms.



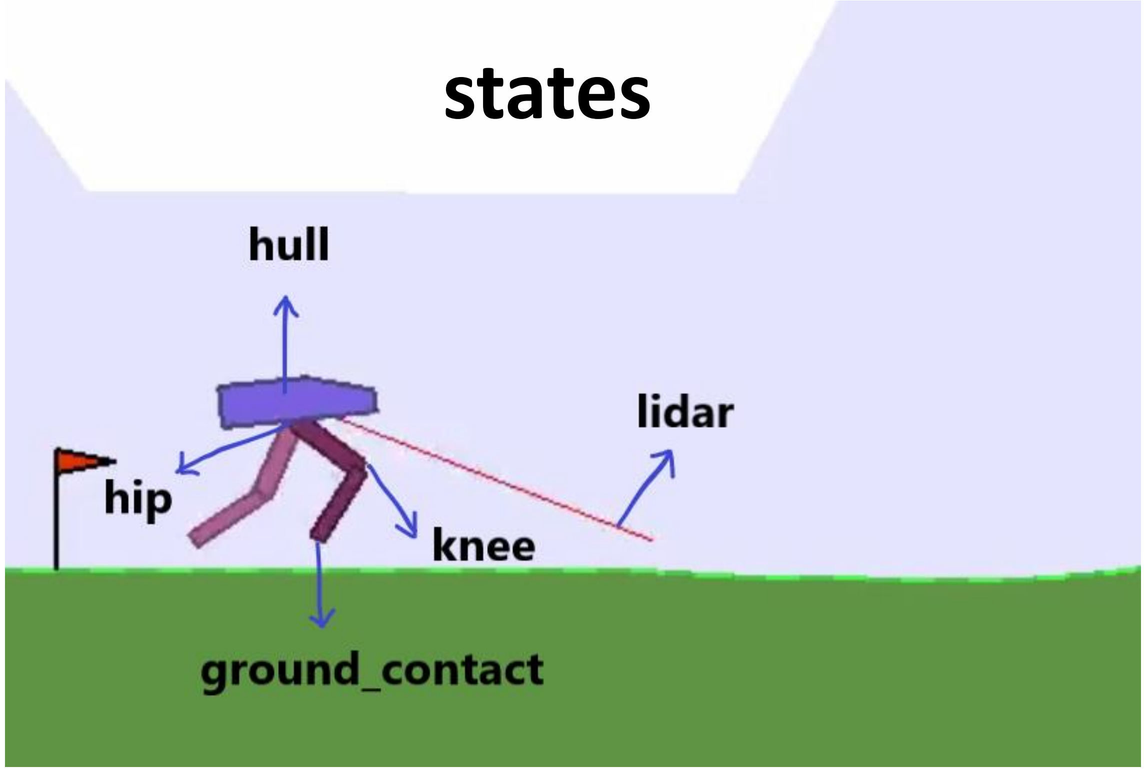
OpenAI Gym environment

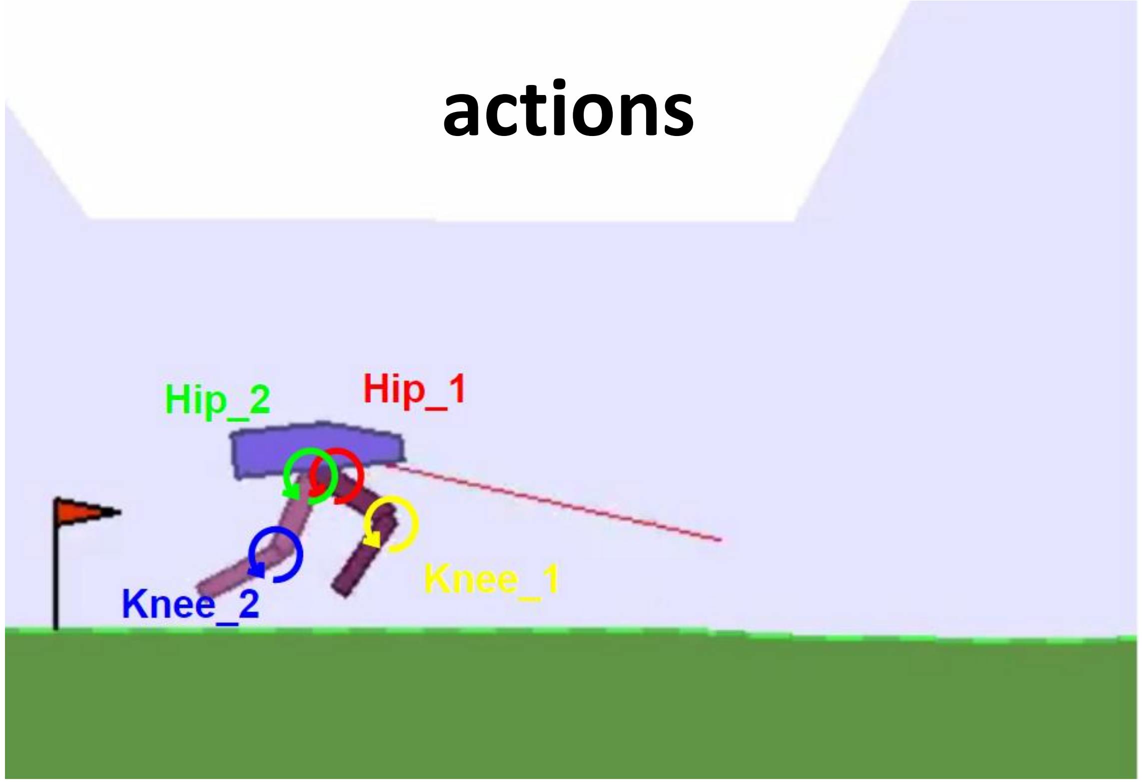
- We are going to use BipedalWalker-v3 environment.
- The environment simulates a random and uneven terrain.
- The goal is to develop an RL algorithm which can drive the Bipedal Walker, to walk to the end of the terrain without falling down.



OpenAI Gym environment

- The 24 states represent the x and y velocities, the joint angle and angular velocities, ground contact of legs, and lidar readings for ground distance.
- The 4 actions represent the torque control of 2 hips and 2 knees.



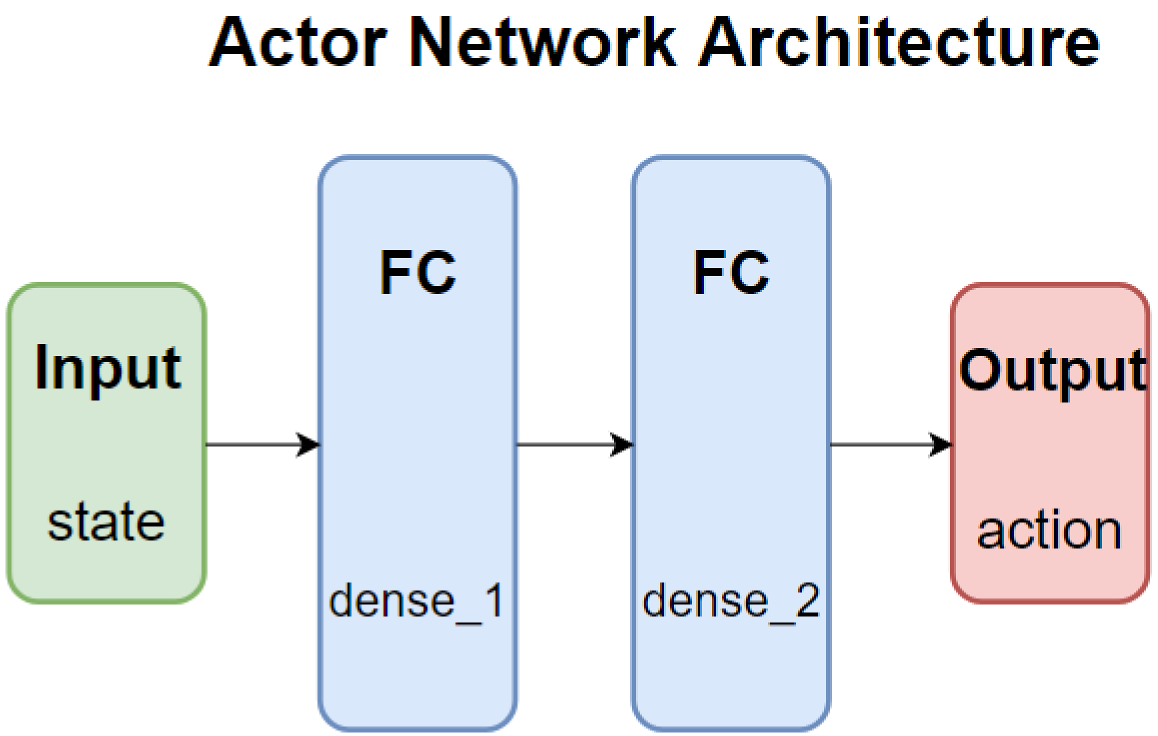


Outline

- **OpenAI Gym environment**
- **Model Architecture**
- **Algorithm**
- **Result**
- **Reference**

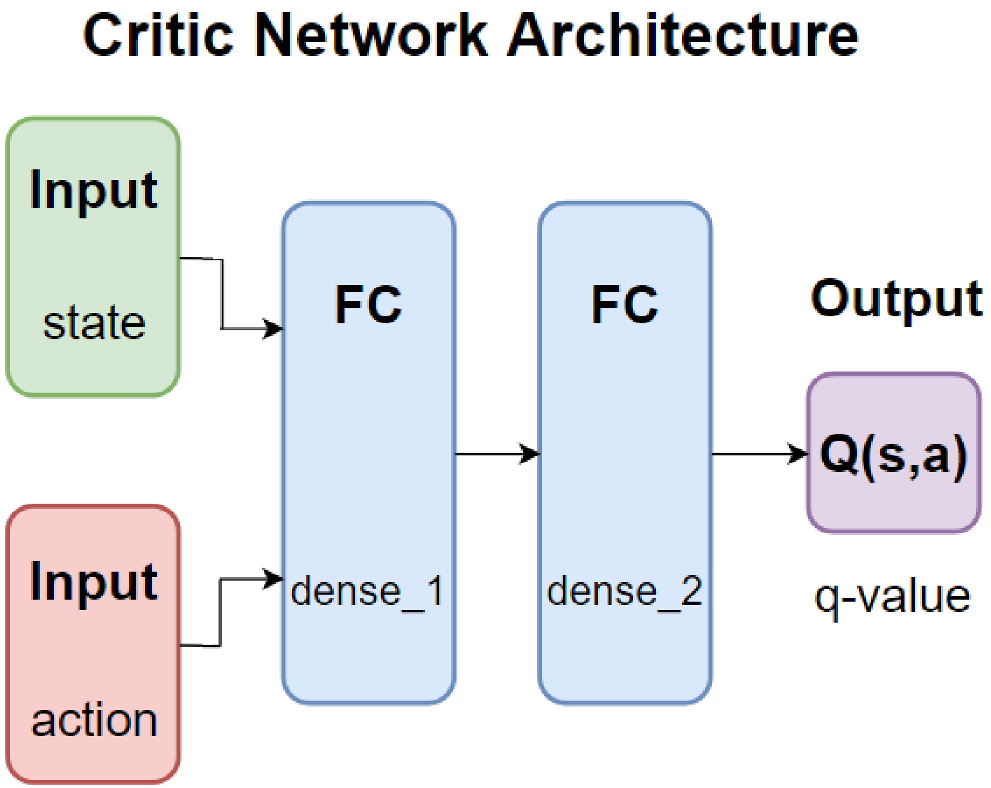
Model Architecture

- Twin-Delayed DDPG contains 2 actor (policy) networks: actor-evaluate and actor-target. They have same architecture and different random weights.
- The actor (policy) takes the states and predicts what actions to perform.
- Notation: $a = \pi(s)$



Model Architecture

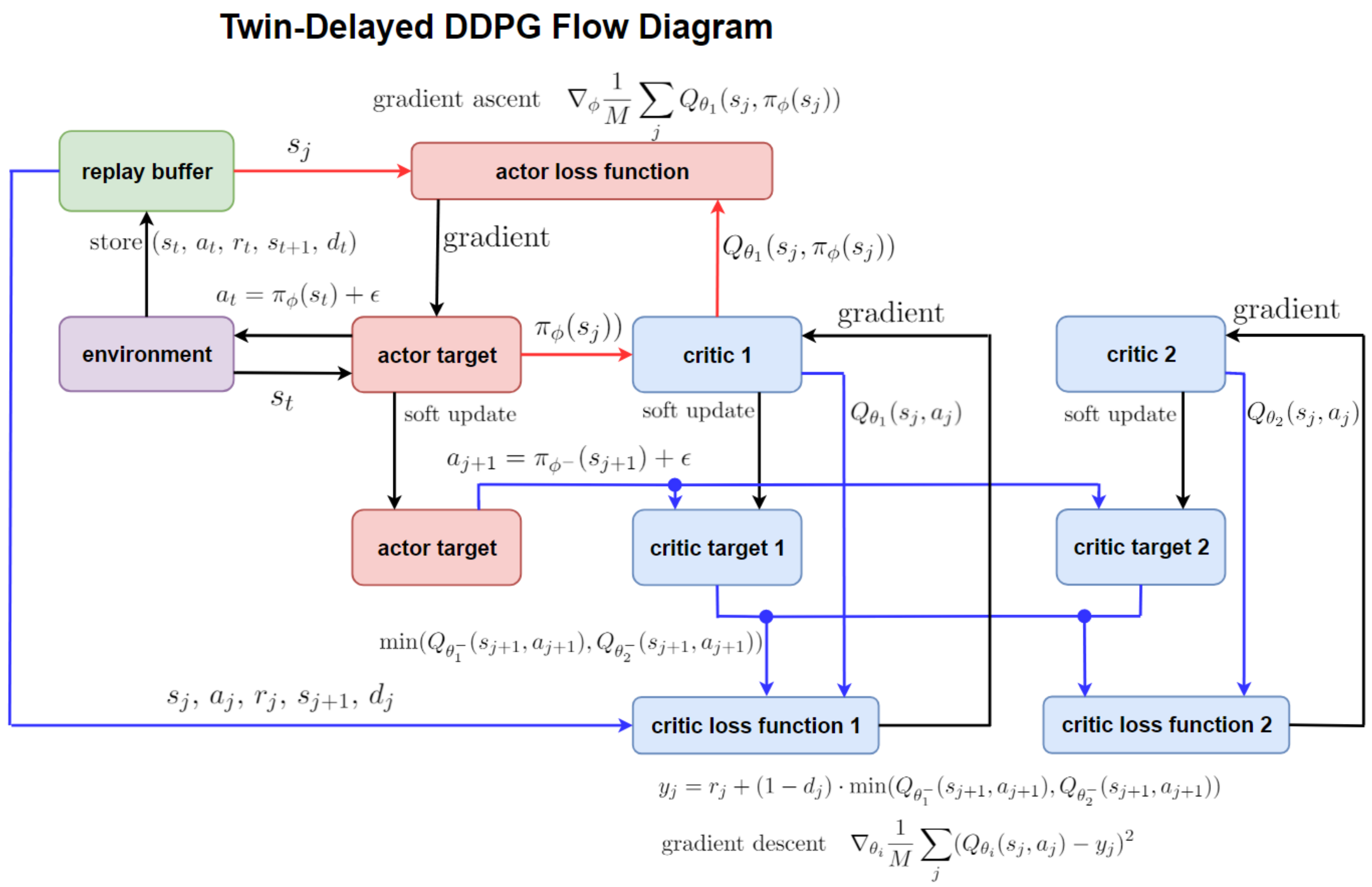
- Twin-Delayed DDPG contains 4 critic networks: 2 critic-evaluate and 2 critic-target. They have same architecture and different random weights.
- Two critics take the states and actions to predict estimated q-value.
- Notation: q-value $= Q(s, a)$



Outline

- **OpenAI Gym environment**
- **Model Architecture**
- **Algorithm Implementation**
- **Result**
- **Reference**

Algorithm



Twin-Delayed DDPG Flow Diagram
gradient ascent $\nabla_\phi \frac{1}{M}\sum_j Q_{\theta_1}(s_j, \pi_\phi(s_j))$
replay buffer
s_j
actor loss function
store $(s_t,\ a_t,\ r_t,\ s_{t+1},\ d_t)$
gradient
$Q_{\theta_1}(s_j, \pi_\phi(s_j))$
$a_t = \pi_\phi(s_t) + \epsilon$
environment
actor target
$\pi_\phi(s_j))$
critic 1
gradient
critic 2
gradient
s_t
soft update
soft update
$Q_{\theta_1}(s_j, a_j)$
soft update
$Q_{\theta_2}(s_j, a_j)$
$a_{j+1} = \pi_{\phi^-}(s_{j+1}) + \epsilon$
actor target
critic target 1
critic target 2
$\min(Q_{\theta_1^-}(s_{j+1}, a_{j+1}), Q_{\theta_2^-}(s_{j+1}, a_{j+1}))$
$s_j,\ a_j,\ r_j,\ s_{j+1},\ d_j$
critic loss function 1
critic loss function 2
$y_j = r_j + (1 - d_j) \cdot \min(Q_{\theta_1^-}(s_{j+1}, a_{j+1}), Q_{\theta_2^-}(s_{j+1}, a_{j+1}))$
gradient descent $\nabla_{\theta_i} \frac{1}{M}\sum_j (Q_{\theta_i}(s_j, a_j) - y_j)^2$

Algorithm

Algorithm 1 Twin Delayed DDPG

Initialize two critic nets Q with random weights θ_1, θ_2, one actor net π with random weights ϕ

Set target net weights $\theta_1^- \leftarrow \theta_1$, $\theta_2^- \leftarrow \theta_2$, $\phi^- \leftarrow \phi$

Initialize learning rate η, reward decay γ, soft update parameter τ

Initialize noise standard deviation σ, noise boundary c, delay interval K

Initialize replay memory R to capacity N and minibatch size M

Initialize episode E with play time T

for $episode = 1$ to E **do**

  Reset environment and agent to random initial state s_0

  Set done $d = 0$

  for $t = 1$ to T **do**

    Select action with exploration noise $a_t = \pi_\phi(s_t) + \epsilon$, and $\epsilon \sim clip(\mathcal{N}(0,\sigma), -c, c)$

    Executes action a_t at state s_t and observes reward r_t, next state s_{t+1}, and done d_t

    Store transition (s_t, a_t, r_t, s_{t+1}, d_t) in replay memory R

    Sample minibatch with M transitions (s_j, a_j, r_j, s_{j+1}, d_j) from replay memory R

    Calculate next actions for minibatch $a_{j+1} = \pi_{\phi^-}(s_{j+1}) + \epsilon$, and $\epsilon \sim clip(\mathcal{N}(0,\sigma), -c, c)$

    Calculate target $y_j = r_j + (1 - d_j) \cdot \gamma \cdot \min(Q_{\theta_1^-}(s_{j+1}, a_{j+1}), Q_{\theta_2^-}(s_{j+1}, a_{j+1}))$

    Perform two gradient descents to update two critic nets respectively:

$$\nabla_{\theta_i} \frac{1}{M} \sum_j (Q_{\theta_i}(s_j, a_j) - y_j)^2 \qquad \text{for i} \in \{1,2\}\text{, j in minibatch}$$

    if $t \mod K = 0$ **then**

      Perform gradient ascent to update actor net using critic net 1:

$$\nabla_\phi \frac{1}{M} \sum_j Q_{\theta_1}(s_j, \pi_\phi(s_j)) \qquad \text{for j in minibatch}$$

      Soft update target nets:

$$\begin{aligned} \theta_i^- &\leftarrow \tau \cdot \theta_i + (1-\tau) \cdot \theta_i^- \qquad \text{for i} \in \{1,2\} \\ \phi^- &\leftarrow \tau \cdot \phi + (1-\tau) \cdot \phi^- \end{aligned}.$$

    end if

    if done $d = 1$ **then**

      Break

    end if

    Move on to next state $s_t \leftarrow s_{t+1}$

  end for

end for

Algorithm

Algorithm 1 Twin Delayed DDPG

Initialize two critic nets Q with random weights θ_1, θ_2, one actor net π with random weights ϕ
Set target net weights $\theta_1^- \leftarrow \theta_1$, $\theta_2^- \leftarrow \theta_2$, $\phi^- \leftarrow \phi$
Initialize learning rate η, reward decay γ, soft update parameter τ
Initialize noise standard deviation σ, noise boundary c, delay interval K
Initialize replay memory R to capacity N and minibatch size M
Initialize episode E with play time T
for $episode = 1$ to E **do**
    Reset environment and agent to random initial state s_0
    Set done $d = 0$
    for $t = 1$ to T **do**
        Select action with exploration noise $a_t = \pi_\phi(s_t) + \epsilon$, and $\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$
        Executes action a_t at state s_t and observes reward r_t, next state s_{t+1}, and done d_t
        Store transition $(s_t,\ a_t,\ r_t,\ s_{t+1},\ d_t)$ in replay memory R

1. Initialize random weights for 2 actor networks and 4 critic networks.
2. Set parameters such as learning rate, reward decay, or soft update parameter.

Algorithm

Algorithm 1 Twin Delayed DDPG

```
Initialize two critic nets $Q$ with random weights $\theta_1$, $\theta_2$, one actor net $\pi$ with random weights $\phi$
Set target net weights $\theta_1^- \leftarrow \theta_1$, $\theta_2^- \leftarrow \theta_2$, $\phi^- \leftarrow \phi$
Initialize learning rate $\eta$, reward decay $\gamma$, soft update parameter $\tau$
Initialize noise standard deviation $\sigma$, noise boundary $c$, delay interval $K$
Initialize replay memory $R$ to capacity $N$ and minibatch size $M$
Initialize episode $E$ with play time $T$
for $episode = 1$ to $E$ do
    Reset environment and agent to random initial state $s_0$
    Set done $d = 0$
    for $t = 1$ to $T$ do
        Select action with exploration noise $a_t = \pi_\phi(s_t) + \epsilon$, and $\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$
        Executes action $a_t$ at state $s_t$ and observes reward $r_t$, next state $s_{t+1}$, and done $d_t$
        Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in replay memory $R$
```

1. Actor-evaluate selects action with bounded random noise.
2. Execute action at state in gym.
3. Observe reward and next state.
4. Store transition in replay memory.

Algorithm

Sample minibatch with M transitions $(s_j,\ a_j,\ r_j,\ s_{j+1},\ d_j)$ from replay memory R

Calculate next actions for minibatch $a_{j+1} = \pi_{\phi^-}(s_{j+1}) + \epsilon$, and $\epsilon \sim clip(\mathcal{N}(0,\sigma), -c, c)$

Calculate target $y_j = r_j + (1 - d_j) \cdot \gamma \cdot \min(Q_{\theta_1^-}(s_{j+1}, a_{j+1}), Q_{\theta_2^-}(s_{j+1}, a_{j+1}))$

Perform two gradient descents to update two critic nets respectively:

$$\nabla_{\theta_i} \frac{1}{M} \sum_j (Q_{\theta_i}(s_j, a_j) - y_j)^2 \qquad \text{for i} \in \{1,2\}\text{, j in minibatch}$$

1. Sample minibatch transitions for experience replay.
2. Use actor-target to calculate next actions with bounded random noise for minibatch.
3. Calculate 2 next q-values with 2 critic-targets. Select the smaller next q-value of the two, and combine with the reward to calculate target for each transition in the minibatch.

Algorithm

Sample minibatch with M transitions $(s_j,\ a_j,\ r_j,\ s_{j+1},\ d_j)$ from replay memory R

Calculate next actions for minibatch $a_{j+1} = \pi_{\phi^-}(s_{j+1}) + \epsilon$, and $\epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$

Calculate target $y_j = r_j + (1 - d_j) \cdot \gamma \cdot \min(Q_{\theta_1^-}(s_{j+1}, a_{j+1}), Q_{\theta_2^-}(s_{j+1}, a_{j+1}))$

Perform two gradient descents to update two critic nets respectively:

$$\nabla_{\theta_i} \frac{1}{M} \sum_j (Q_{\theta_i}(s_j, a_j) - y_j)^2 \qquad \text{for i} \in \{1, 2\}\text{, j in minibatch}$$

if $t \mod K = 0$ **then**

  Perform gradient ascent to update actor net using critic net 1:

$$\nabla_{\phi} \frac{1}{M} \sum_j Q_{\theta_1}(s_j, \pi_\phi(s_j)) \qquad \text{for j in minibatch}$$

1. For every experience replay, perform gradient descend and update two critic-evaluate with the targets calculated in previous slide.
2. For every K experience replay, perform gradient ascend and update actor-evaluate using the “first” critic-evaluate.
3. The actor update is “delayed” for stabilizing training process.

Algorithm

if $t \mod K = 0$ **then**

Perform gradient ascent to update actor net using critic net 1:

$$\nabla_\phi \frac{1}{M} \sum_j Q_{\theta_1}(s_j, \pi_\phi(s_j)) \qquad \text{for j in minibatch}$$

Soft update target nets:

$$\begin{aligned} \theta_i^- &\leftarrow \tau \cdot \theta_i + (1-\tau)\cdot \theta_i^- \qquad \text{for i} \in \{1,2\} \\ \phi^- &\leftarrow \tau \cdot \phi + (1-\tau)\cdot \phi^- \end{aligned}$$

end if

if done $d = 1$ **then**

Break

end if

Move on to next state $s_t \leftarrow s_{t+1}$

end for

end for

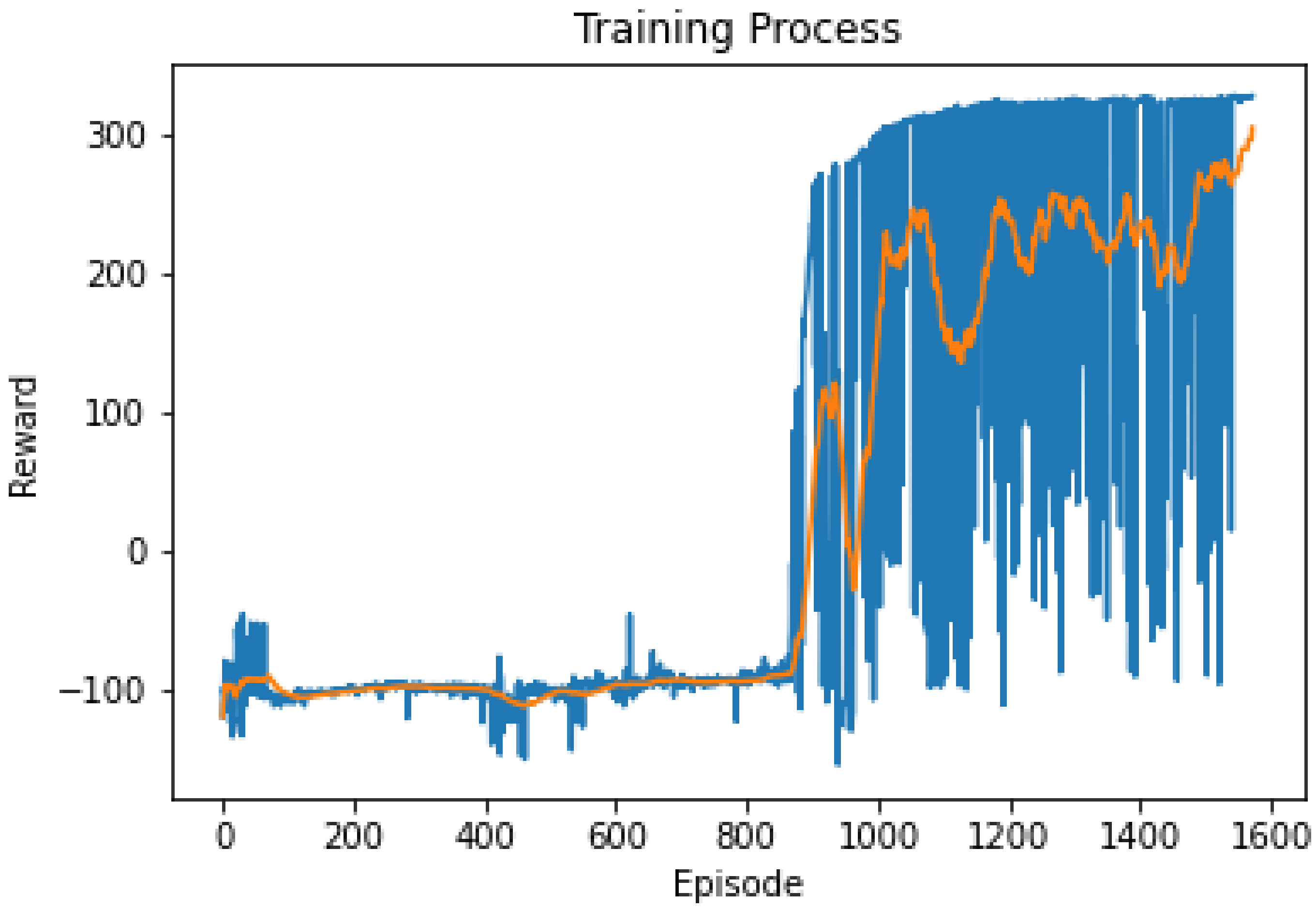
1. Every K experience replay, when we update actor-evaluate, we also do soft update for actor and critic target networks.
2. This time-step finishes, this state move on to next state.

Outline

- **OpenAI Gym environment**
- **Model Architecture**
- **Algorithm Implementation**
- **Result**
- **Reference**

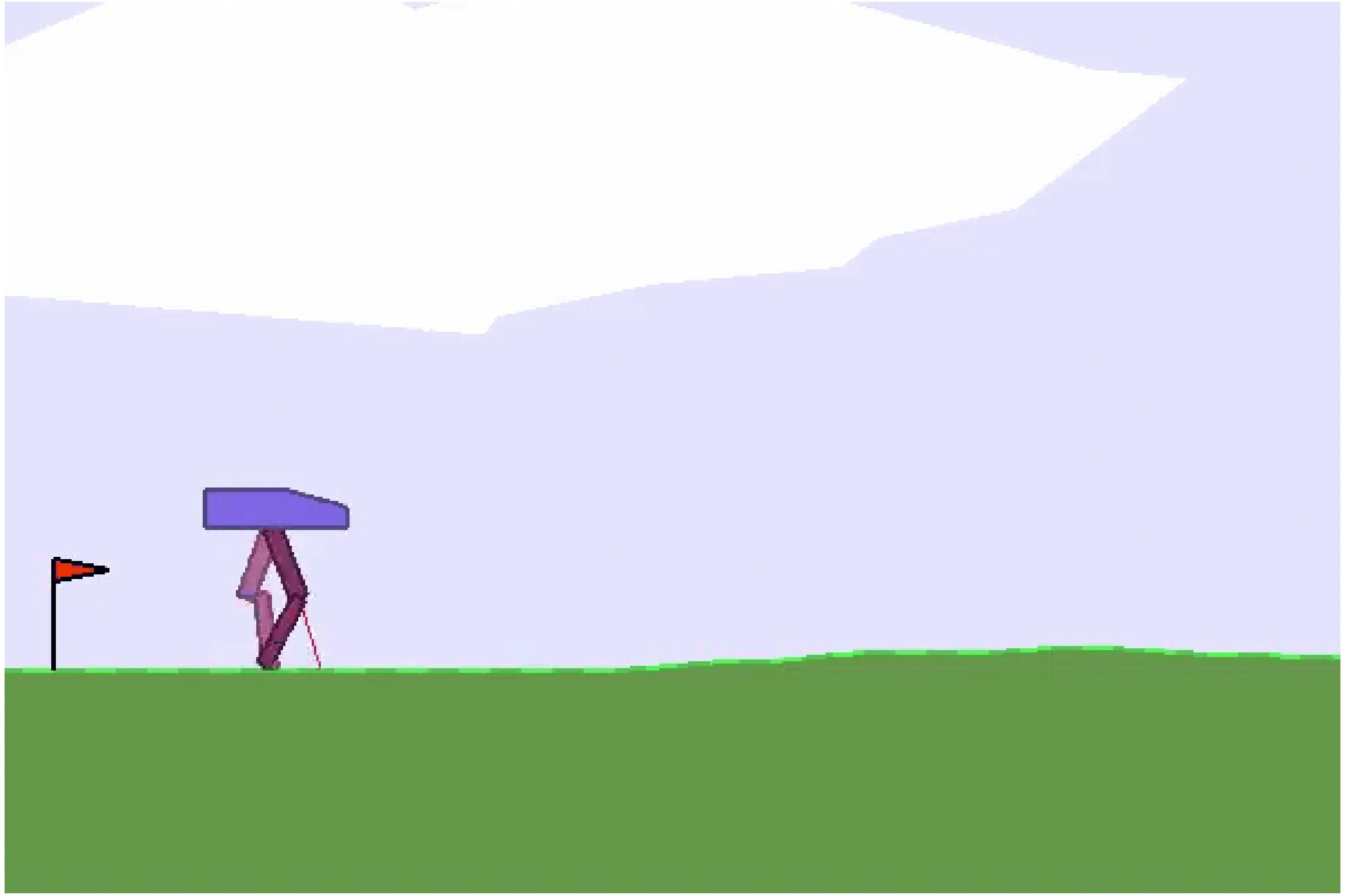
Result

- The training process is shown below. After training, the average score for greedy agent (action without noise) is around 328, which is considered solved for this environment.



Result

- We can see that the Bipedal Walker can successfully walk to the end of the terrain.



Outline

- **OpenAI Gym environment**
- **Model Architecture**
- **Algorithm Implementation**
- **Result**
- **Reference**

Reference

- Continuous control with deep reinforcement learning
- Addressing Function Approximation Error in Actor-Critic Methods
- Deep Deterministic Policy Gradient
- Deep Reinforcement Learning. Deep Deterministic Policy Gradient (DDPG) algorithm
- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed DDPG
- TD3: Learning To Run With AI

Thank you for your attention!!