Off-Policy Algorithms for Continuous Control (DDPG/TD3/SAC)

Khoa. Tran[1] Luc. Truong[2] Vu. Huynh[3]

[1]20520222 ‘ [2]20520241 [3]20520864

Class : CS106.M21.KHTN

Supervisor: Dr. Luong Ngoc Hoang

July 08, 2022

Table of Contents

1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison

Table of Contents

1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison

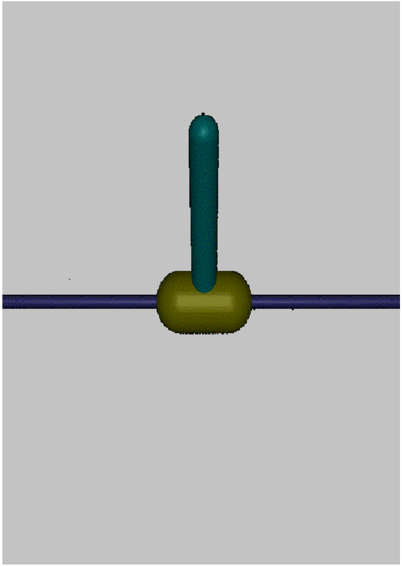
Continuous Control Problem

INPUT

Num	Observation	Min	Max
0	position of the cart along the linear surface	-Inf	Inf
1	vertical angle of the pole on the cart	-Inf	Inf
2	linear velocity of the cart	-Inf	Inf
3	angular velocity of the pole on the cart	-Inf	Inf

OUTPUT

Num	Action	Control Min	Control Max
0	Force applied on the cart	-3	3



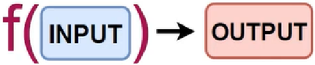
Continuous Control Problem

INPUT

Num	Observation	Min	Max
0	position of the cart along the linear surface	-Inf	Inf
1	vertical angle of the pole on the cart	-Inf	Inf
2	linear velocity of the cart	-Inf	Inf
3	angular velocity of the pole on the cart	-Inf	Inf

OUTPUT

Num	Action	Control Min	Control Max
0	Force applied on the cart	-3	3



CONTROLLER / POLICY FUNCTION

$$\pi(s) = a$$

OBJECTIVE FUNCTION

$$J(\pi) = \mathrm{E}_{\tau \sim \pi}[R(\tau)]$$

Terminology:

$$\tau = \{(s_0, a_0), (s_H, a_H)\}$$

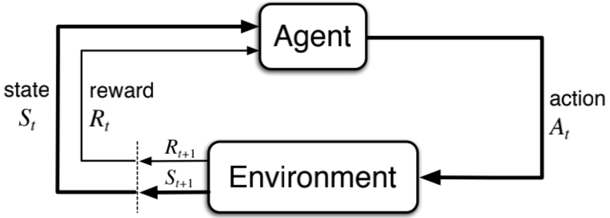
$$R(\tau) = \sum_{t=0}^{H} \gamma^t r(s_t, a_t)$$

(reward of decision seq.)

Discretized time axis:
0, 1, 2, ..., H (Time Horizon)

Table of Contents

1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison



π' RL considers solving Markov Decision Process (MDP)
A policy a $\sim \pi(.|\theta)$
Sample a trajectory from $\pi : s_t \rightarrow a_t \sim \pi(.|s_t) \rightarrow r_t \rightarrow st + 1 \rightarrow ...$
the object :

$$\pi^* = \underset{\pi}{\text{argmax}} J^\gamma(\pi) = \underset{\pi}{\text{argmax}}\, \mathbb{E}(\sum_{n=1}^{N-1} r_n \gamma^n)$$

Value Based Method

Many methods in reinforcement learning approach rely on learning a value function like Q-value to find the optimal policy. Where

$$Q(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})]$$

Q-learning:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[R_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Once we have Q-value, an optimal policy can be easily found

$$\pi(s) = argmax_a Q(s, a)$$

This approach is Value-Based Method.

Policy Gradient

Another approach is Policy-Based Method. Instead of learn a value function, this approach directly learn a approximate policy function with paremeter θ based on gradient of object function.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \mathcal{J}(\pi_{\theta_t})$$

where $\nabla \mathcal{J}(\pi_\theta)$ can be computed by using *policy gradient theorem*:

$$\begin{aligned} \nabla_\theta \mathcal{J}_{\pi_\theta} &= \mathbb{E}_{\pi_\theta}[V^{\pi_\theta}(s_0)] \\ &= \mathsf{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)] \end{aligned}$$

Table of Contents

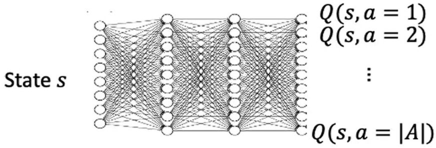
1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison

DDPG - Main Idea

The main idea of DDPG is combining DPG and DQN.

DQN is an algorithm built to solve large-dimensional tasks in state space by combining basic Q-learning algorithm and neural network.

Specifically, DQN learns an approximation of the function Q based on the basic Q-learning algorithm.

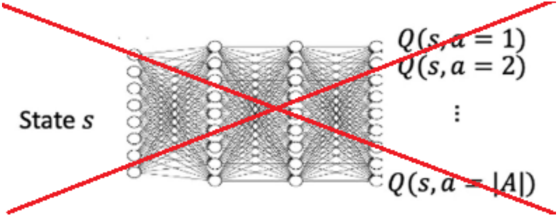


With the policy being a greedy policy

$$\pi(s) = argmax_a Q(s, a)$$

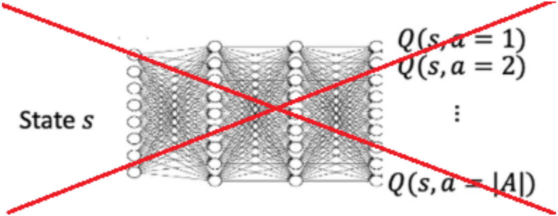
DDPG - Main Idea

DQN **only** works in an environment with a discrete action space. This is because in a task with a continuous action space, it is not possible to find Q-value for all actions and apply a greedy policy to choose the action that has a maximum Q-value



DDPG - Main Idea

DQN **only** works in an environment with a discrete action space. This is because in a task with a continuous action space, it is not possible to find Q-value for all actions and apply a greedy policy to choose the action that has a maximum Q-value



To solve this problem, DDPG algorithm uses an actor-critic architecture based on the DPG algorithm. And to extend the DPG algorithm to solve continuous control tasks, DPG uses a neural network as an approximation function like the DQN algorithm.

DDPG - Main Idead

The DPG algorithm maintains :

- Actor $\mu(s|\theta^\mu)$ with a parameter θ^μ representing a deterministic policy of the agent, which deterministic mapping a state to a specific action. Update using deterministic policy gradient theorem

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t \sim p^\beta}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}]$$

$$= \mathbb{E}_{s_t \sim p^\beta}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta_\mu)|_{s_t}]$$

DDPG - Main Idead

The DPG algorithm maintains :

- Actor $\mu(s|\theta^{\mu})$ with a parameter θ^{μ} representing a deterministic policy of the agent, which deterministic mapping a state to a specific action. Update using deterministic policy gradient theorem

$$\nabla_{\theta^{\mu}} J \approx \mathbb{E}_{s_t \sim p^{\beta}}[\nabla_{\theta^{\mu}} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t|\theta^{\mu})}]$$
$$= \mathbb{E}_{s_t \sim p^{\beta}}[\nabla_{a} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta_{\mu})|_{s_t}]$$

- Critic $Q(s, a|\theta^{Q})$ with a parameter θ^{Q} use to learn Q function, showing how well the action is chosen by the actor. Update by minimizing loss:

$$L = \mathbb{E}[(y_t - Q(s_t, a_t|\theta_Q))^2]$$

where

$$y_t = r(s_t, a_t) + (s_{t+1}, \mu(s_{t+1}|\theta^{Q})$$

DDPG-Replay buffer and Target Network

DDPG algorithm borrows the idea of using *replay buffer* and *target networks* from DQN algorithm.

DDPG-Replay buffer and Target Network

DDPG algorithm borrows the idea of using *replay buffer* and *target networks* from DQN algorithm.

Replay buffer use to store experience when the agent interacts with the environment, this makes for better learning because when randomly taking experience from the replay buffer to break the correlation between consecutive samples.

DDPG-Replay buffer and Target Network

DDPG algorithm borrows the idea of using *replay buffer* and *target networks* from DQN algorithm.

Replay buffer use to store experience when the agent interacts with the environment, this makes for better learning because when randomly taking experience from the replay buffer to break the correlation between consecutive samples.

Target networks use to keep learning stable by frozen the target value. Update by using *soft update*: $\theta' = \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$.

DDPG-Replay buffer and Target Network

DDPG algorithm borrows the idea of using *replay buffer* and *target networks* from DQN algorithm.

Replay buffer use to store experience when the agent interacts with the environment, this makes for better learning because when randomly taking experience from the replay buffer to break the correlation between consecutive samples.

Target networks use to keep learning stable by frozen the target value. Update by using *soft update*: $\theta' = \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$.Changing to soft-update causes the target network to change slowly, greatly improving the stability of learning.

Exploration

One problem with deterministic policies is that it often quickly converges to produce the same actions while not exploring enough, thereby ignoring states that might lead to a higher total expected reward. To address this problem, the DDPG algorithm constructs an exploration policy μ' by adding noise to the actor policy

$$\mu'(s_t) = \mu(s_t) + \mathcal{N}$$

$\mathcal{N}$ can be anything, DDPG's author choice is to use Ornstein-Uhlenbeck process.

DDPG - Pseudocode

Algorithm 1 DDPG 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 $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ 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 a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    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 policy gradient:

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

    Update the target networks:

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

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

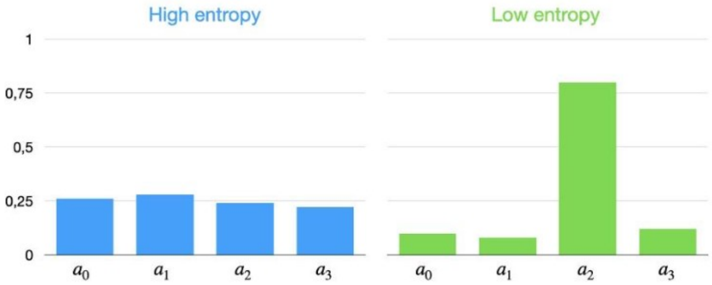
  end for
end for

Table of Contents

1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison

Entropy

In RL, entropy refers to the predictability of the actions of an agent. This is closely related to the certainty of its policy about what action will yield the highest cumulative reward in the long run: if certainty is high, entropy is low and vice versa



Entropy

With x is a random variable with probability mass function P(X)
Etropy can be calculated by

$$H(X) = -\sum_{x \in X} P(x) log P(x)$$

In RL we want to calcutate the entropy of policy π
The equation now become

$$H(\pi(.|s_t)) = -\sum_{a \in A} \pi(a|s_t) log \pi(a|s_t)$$

To do that we need the policy $\pi isaProbabilitydistribution$
So that the SAC actor need to be a Stochastic Actor

Stochastic Actor

Unlike DDPG and TD3 what give directly the action vector given a state vector
A Stochastic Actor will give mean and covariance
Then use the Gaussian distribution with mean and coveriance to make a action vector

SAC is an off-policy algorithm that is built on the actor-critic framework. The algorithm push a entropy of policy in RL objective function. The RL objective function now become

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t, a_t) \sim \pi}[r(s_t, a_t) + \alpha H(\pi(.|s_t))]$$

The policy now need to learn how to maximize the reward and the entropy at the same time

soft actor-critic Pseudocode

Data: $\lambda_\psi, \lambda_{\theta_1}, \lambda_{\theta_2}, \lambda_\phi, \tau$
Initialize parameter vectors $\psi, \hat{\psi}, \theta_1.\theta_2, \phi$
for *each iteration* **do**
    for *each step in environment* **do**
        $a_t \sim \pi_\phi(a_t, s_t)$
        $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $D \leftarrow D \cup (s_t, a_t, r, s_{t+1})$
        end
    for *each gradient step* **do**
        $\psi \leftarrow \psi - \lambda_\psi \nabla_\psi J_V(\psi)$
        for *i in 1,2* **do**
            $\theta_i \leftarrow \theta_i - \lambda_{\theta i} \nabla_{\theta i} J_Q(\theta_i)$
            end
            $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi J_\pi(\phi)$
            $\psi \leftarrow \tau\psi + (1 - \tau)\hat{\psi}$
            end

soft actor-critic

We will consider some neural network to describe:
state value function: $V_\psi(s_t)$
soft Q function: $Q_\theta(s_t, a_t)$
policy: $\pi_\phi(a_t|s_t)$
The parameters of these network are ψ, θ, ϕ.

soft actor-critic

The soft value function is trained to minimize the squared residual error

$$J_v(\psi) = \mathbb{E}_{s_t \sim D}[1/2(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - log\pi_\phi(a_t|s_t)])^2]$$

The gradient it can be estimated by

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t, a_t) + log\pi_\phi(a_t, s_t))$$

soft actor-critic

The soft Q-function parameters can be trained to minimize the soft Bellman residual

$$J_Q(\theta) = \mathbb{E}_{(s_t,a_t)\sim D}[1/2(Q_\theta(s_t,a_t) - \hat{Q}(s_t,s_t))^2]$$

with

$$\hat{Q}(s_t,s_t) = r(s_t,s_t) + \gamma\,\mathbb{E}_{s_t+1\sim p}[V_{\hat{\psi}(s_t,a_t)}]$$

can be optimized with stochastic gradients

$$\nabla_\theta J_Q(\theta) = \nabla_\theta Q_\theta(s_t,a_t) - r(s_t,a_t) - \gamma V_{\hat{\psi}(s_{t+1})})$$

Finally, the policy parameters can be learned by directly minimizing the objective function

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim D, \epsilon \sim N}[log \pi_\phi(f_\phi(\epsilon, s_t)|s_t) - Q_\phi(s_t, f_\phi(\epsilon_t; s_t))]$$

We can approximate the gradient with

$$\begin{gathered} {}_\phi J_\pi(\phi) = \nabla_\phi log \pi_\phi(a_t, s_t) \\ + (\nabla_{a_t} log \pi_\phi(a_t, s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\phi f_\phi(\epsilon; s_t) \end{gathered}$$

Table of Contents

1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison

TD3 - Overestimation Bias

- TD3 addresses overestimation bias.

TD3 - Overestimation Bias

- TD3 addresses overestimation bias.
- Overestimation bias occurs when estimated values $Q(s, a)$ are in general greater than true values $Q(s, a)$.

TD3 - Overestimation Bias

- TD3 addresses overestimation bias.
- Overestimation bias occurs when estimated values $Q(s, a)$ are in general greater than true values $Q(s, a)$.
- In Q-Learning we get bias from the max over actions.

$$y = r + \gamma \max_{a'} Q\left(s', a'\right)$$

TD3 - Overestimation Bias

- Overestimation also comes from approximation errors.

TD3 - Overestimation Bias

- Overestimation also comes from approximation errors.

- AC methods are bootstrapped so errors accummulate.

$$
\begin{aligned}
& Q_\theta\left(s_t, a_t\right)=r_t+\gamma \mathbb{E}\left[Q_\theta\left(s_{t+1}, a_{t+1}\right)\right]-\delta_t \\
& =r_t+\gamma \mathbb{E}\left[r_{t+1}+\gamma \mathbb{E}\left[Q_\theta\left(s_{t+2}, a_{t+2}\right)-\delta_{t+1}\right]\right]-\delta_t \\
& =\mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}\left[\sum_{i=t}^T \gamma^{i-t}\left(r_i-\delta_i\right)\right]
\end{aligned}
$$

TD3 - Overestimation Bias

- Overestimation also comes from approximation errors.

- AC methods are bootstrapped so errors accummulate.

$$
\begin{aligned}
& Q_\theta\left(s_t, a_t\right)=r_t+\gamma \mathbb{E}\left[Q_\theta\left(s_{t+1}, a_{t+1}\right)\right]-\delta_t \\
& =r_t+\gamma \mathbb{E}\left[r_{t+1}+\gamma \mathbb{E}\left[Q_\theta\left(s_{t+2}, a_{t+2}\right)-\delta_{t+1}\right]\right]-\delta_t \\
& =\mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}\left[\sum_{i=t}^T \gamma^{i-t}\left(r_i-\delta_i\right)\right]
\end{aligned}
$$

- TD3's solution:

$$
y \leftarrow r+\gamma \min_{i=1,2} Q_{\theta_i'}\left(s', \pi_{\theta_1}(s')\right)
$$

TD3 - Pseudocode

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, 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 $\mathcal{B}$

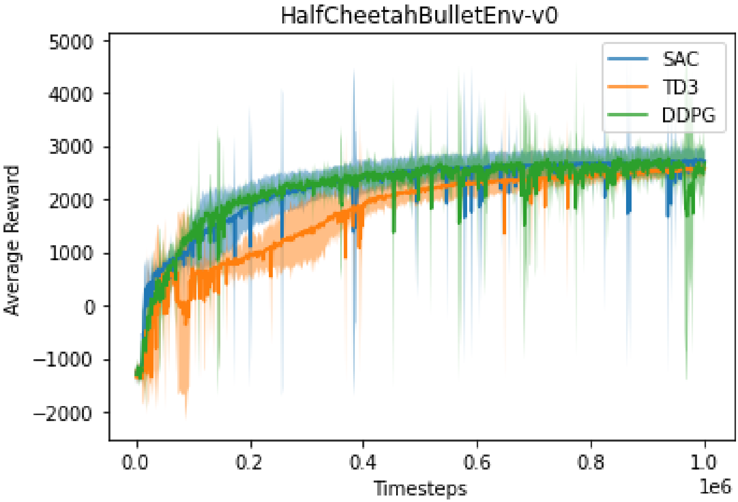
TD3 - Pseudocode

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:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
end if
end for

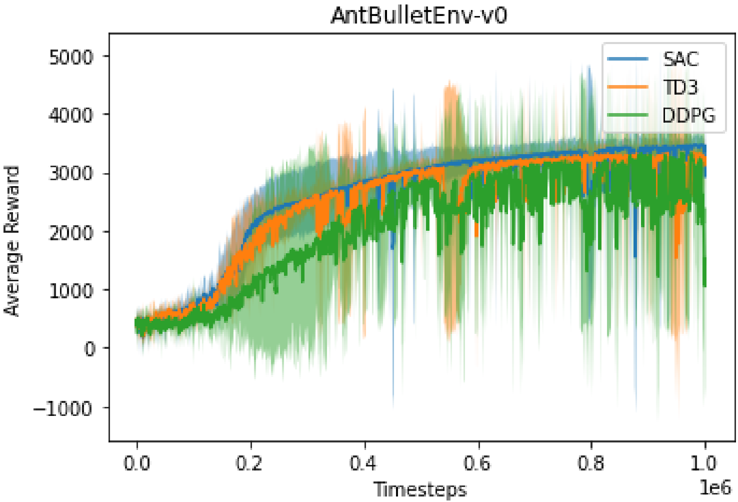
Table of Contents

1. Continuous Control Problem
2. Background
   - MDP and Reinforcement Learning (RL)
   - Policy Gradient
3. DDPG
4. SAC
   - Entropy
   - Stochastic Actor
   - soft actor-critic
5. TD3
6. Comparison

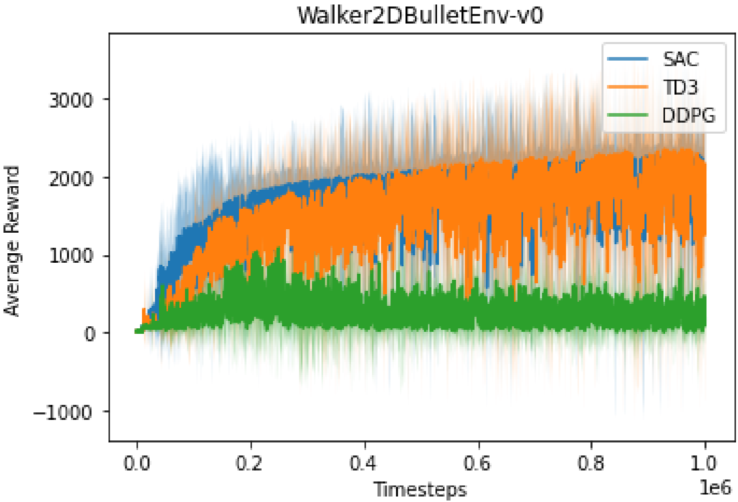
Comparison


HalfCheetahBulletEnv-v0
SAC
TD3
DDPG
Average Reward
5000
4000
3000
2000
1000
0
−1000
−2000
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Timesteps

Comparison


AntBulletEnv-v0
SAC
TD3
DDPG
5000
4000
3000
2000
1000
0
−1000
Average Reward
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Timesteps

Comparison


Walker2DBulletEnv-v0
SAC
TD3
DDPG
Average Reward
3000
2000
1000
0
−1000
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Timesteps


THANK YOU
for your
ATTENTION!