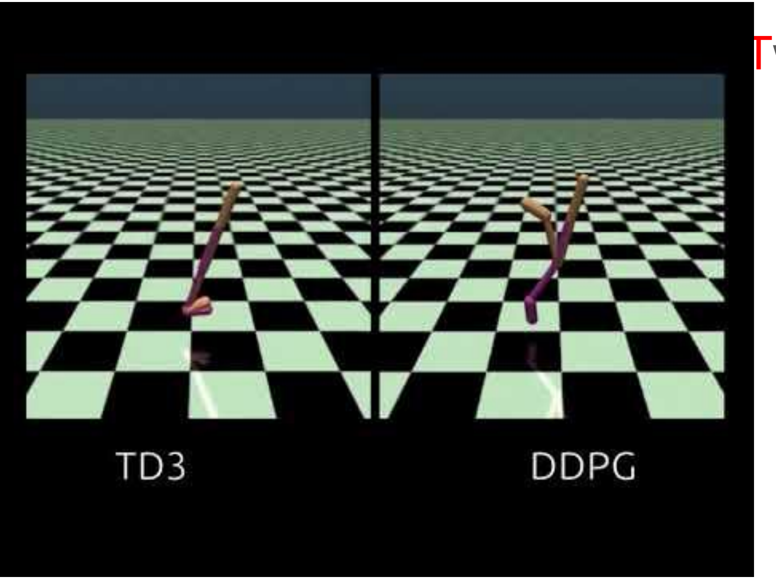
Addressing Function Approximation Error in Actor-Critic Methods(=TD3) 논문 구현



Twin Delayed DDPG

강동구

Introduction

계보 : DQN → DDQN(CDQ) → [DPG] → DDPG → TD3

Quick fact : off-policy, only continuous action, deterministic policy

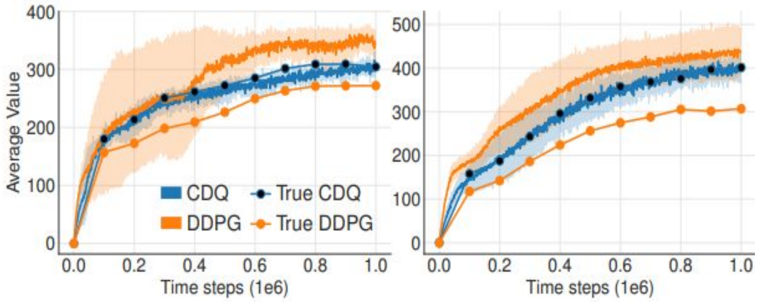
Problem : Function Approximate Error

1. overestimation bias

    solution : Clipped Double Q-learning(CDQ)

2. high variance build-up

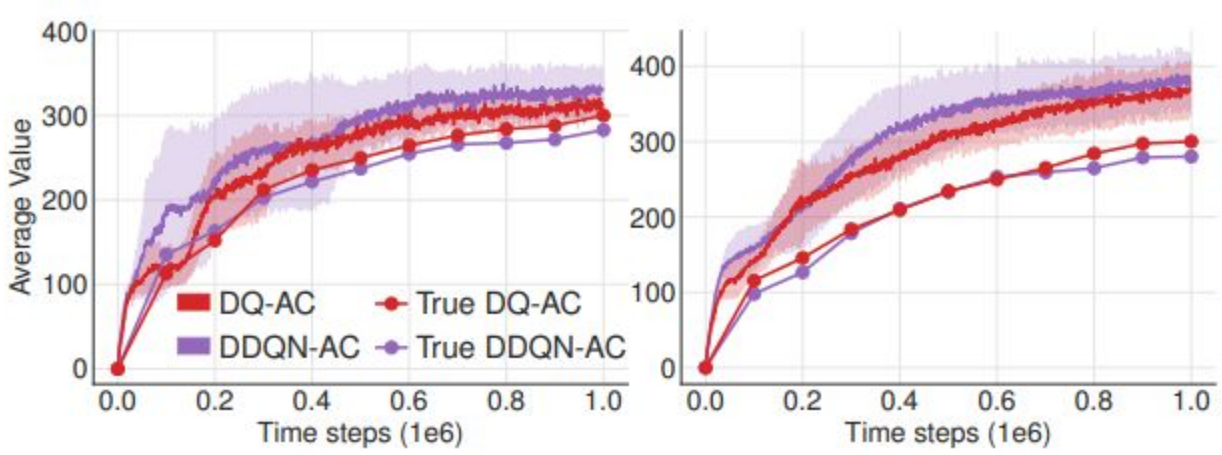
    solution : target-network smoothing update, delaying policy and targets updates



(a) Hopper-v1 (b) Walker2d-v1

Overestimation in actor-critic setting

value based에서 해결방법들(Double DQN, Double Q)이 actor-critic에서는 효과가 없음



(a) Hopper-v1 (b) Walker2d-v1

Why doesn't working in AC setting

기존 방법들의 가정은 독립적인 Q1,Q2를 두어 policy를 업데이트 하면, unbiased estmiation이 되어 문제가 해결된다고 가정.

그러나 critics는 완전히 서로 독립적이지 않다. 같은 replay buffer를 사용하기 때문.

$$Q_{\theta_2}(s, \pi_{\phi_1}(s)) > Q_{\theta_1}(s, \pi_{\phi_1}(s))$$

특정 state s에서 over estimation이 발생하는 우변보다 더 over estimate 하게 된다.

그래서 이 논문에서 최초로 Double Q-learning을 개선한 Clipped Double Q-learning(CDQ)를 소개.

* CDQ는 모든 actor-critic setting에서 critic을 대체가능

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi_1}(s')).$$

While this update rule may induce an underestimation bias, this is far preferable to overestimation bias, as unlike overestimated actions, the value of underestimated actions will not be explicitly propagated through the policy update.

Clipped Double Q-learning

```
with torch.no_grad():
    # Select action according to policy and add clipped noise
    noise = (                                  0.2
        torch.randn_like(action) * self.policy_noise
    ).clamp(-self.noise_clip, self.noise_clip)
                   0.5

    next_action = (
        self.actor_target(next_state) + noise
    ).clamp(-self.max_action, self.max_action)

    # Compute the target Q value
    target_Q1, target_Q2 = self.critic_target(next_state, next_action)
    target_Q = torch.min(target_Q1, target_Q2)
    target_Q = reward + not_done * self.discount * target_Q

# Get current Q estimates
current_Q1, current_Q2 = self.critic(state, action)

# Compute critic loss
critic_loss = F.mse_loss(current_Q1, target_Q) + F.mse_loss(current_Q2, target_Q)
```

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi_1}(s')).$$

Addressing Variance

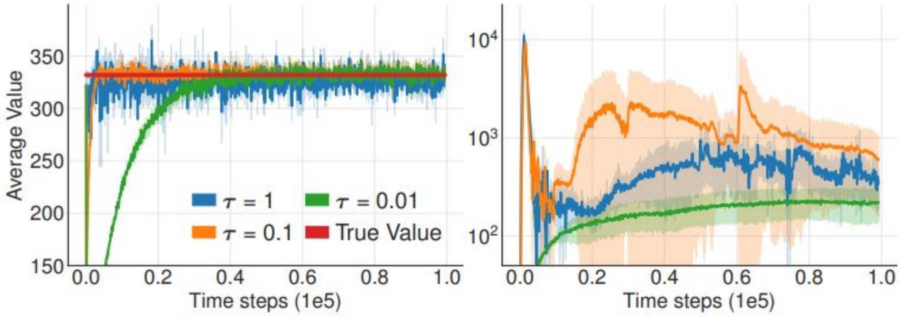
$$Q_\theta(s,a) = r + \gamma \mathbb{E}[Q_\theta(s',a')] - \delta(s,a).$$

위 식이 우리가 예측하고자 하는 value function이고, Expectation을 직접 알수 없기에 아래와 같이 sampling하여 얻어냄.

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

- 위 식은 Q를 배우는 것이 아닌 expected return - sum(δ) 를 학습하는 문제로 봐야 함.
- variance도 expected return, sum(δ)으로 나눠서 봐야하는 문제.
- 여기서 discount factor가 조금이라도 커지면 variance가 매우 커짐.
- 또한 각 업데이트가 mini-batch상에서만 이루어지기에 mini-batch외의 value estimation error의 크기를 고려하지 않음.

delaying policy update



(a) Fixed Policy (b) Learned Policy

τ=1 : no target network, τ=0.1,0.01 : slow-updating

policy를 고정할 경우, variance가 줄어듬.

즉, target이 없을때 variance에 기여하는건, policy의 고정 여부

policy을 value를 계속 따라가게 하지 말고, value보다 덜(½) 업데이트 하자.

delaying policy update + smoothing target update

```
# Delayed policy updates
if self.total_it % self.policy_freq == 0:
                                2

    # Compute actor losse
    actor_loss = -self.critic.Q1(state, self.actor(state)).mean()

    # Optimize the actor
    self.actor_optimizer.zero_grad()
    actor_loss.backward()
    self.actor_optimizer.step()

    # Update the frozen target models
    for param, target_param in zip(self.critic.parameters(), self.critic_target.parameters()):
        target_param.data.copy_(self.tau * param.data + (1 - self.tau) * target_param.data)
                                    0.005
    for param, target_param in zip(self.actor.parameters(), self.actor_target.parameters()):
        target_param.data.copy_(self.tau * param.data + (1 - self.tau) * target_param.data)
```

critic은 매번 업데이트, policy 및 target network는 특정 주기(2)마다 업데이트

d(policy_freq)가 클수록 좋아진다고 하는데…

with $d = 2$. While a larger d would result in a larger benefit with respect to accumulating errors.

Regularization

$$y = r + \mathbb{E}_\epsilon \left[Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon) \right],$$

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon),$$
$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c),$$

```
action = (
    policy.select_action(np.array(state))
    + np.random.normal(0, max_action * args.expl_noise, size=action_dim)
).clip(-max_action, max_action)
```

0.1

deterministic policy에서 value function이 narrow peak(=overfitting)되는 경우를 방지하고자, 비슷한 action은 비슷한 value를 산출해야 한다는 전제로 도입(SARSA에서 최초 사용)

pseudo code

Algorithm 1 TD3

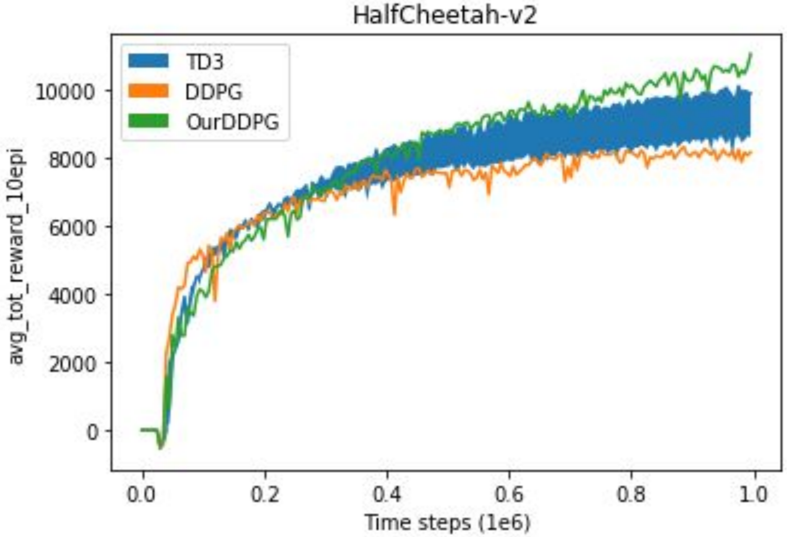
Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s' — target policy smoothing(TPS)
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$ — Clipped double Q-learning(CDQ)
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then** — delayed policy update(DP)
    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:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$ — target policy smoothing(TPS)
  end if
end for

TD3 vs DDPG vs OurDDPG

* DDPG의 Ornstein-Uhlenbeck noise는 별 효과가 없어서 제외된 구현

Unlike the original implementation of DDPG, we used uncorrelated noise for exploration as we found noise drawn from the Ornstein-Uhlenbeck (Uhlenbeck & Ornstein, 1930) process offered no performance benefits.



논문에서의 성능