Proximal Policy Optimization

Bongseokkim

Introduction : Backgorund

Limitation of Vanila Policy Gradient Methods

- Hard to choose stepsizes
  - Input data is nonstationary
    : 학습이 진행됨에 따라 observation과 reward distribution이 계속해서 변함
  - Bad step more damaging than Supervised learning
    1) Step too for → bad poliy
    2) → Next data: collected under bad policy
    3) → Can't cover : collapse in performance
- Sample efficency
  - Only one gradient step per sample
- TRPO & PPO
  - 관점 : changing the underlying optimization algoritms more stable
  - cf) ActorCritic .. : wants lower variance advantage estimate

Likelihood Ratio Policy Gradient

Objective : find θ that maximize $\mathbb{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta]$

- τ : state-action sequence (trajectory)
  - $s_0, a_1, s_1, a_1..$
- $R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$

$$
\begin{aligned}
U(\theta) &= \mathbb{E}[\sum_{t=0}^{H} R(s_t)|\pi_\theta] \quad (timestep) \\
&= \sum_{\tau} P(\tau;\theta)R(\tau) \quad (trajectory)
\end{aligned}
\tag{1}
$$

Likelihood Ratio Policy Gradient

$$
\begin{aligned}
\nabla U(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\
&= \sum_\tau P(\tau;\theta) \nabla_\theta log P(\tau;\theta) R(\tau) \\
&\approx \frac{1}{m} \sum_{m=1} \nabla_\theta log P(\tau^{(i)};\theta) R(\tau)
\end{aligned}
$$

- where, $\nabla_\theta log P(\tau^{(i)};\theta) = \sum_{t=0}^{H} \nabla log \pi_\theta(a_t, s_t)$
- $E[g\hat{}] = \nabla U(\theta)$

Importance Sampling

- policy gradient를 구하는 또 다른 방법 (TRPO, PPO 사용)

importance Sampling

- 기대값을 계산하고자 하는 $p(x)$의 확률 밀도는 알고 있지만 , p의 sample 들을 생성하기 어려울때, 비교적 생성하기가 쉬운 $q(x)$에서 p의 기대 값을 계산하는 것이다.
- 여기서는 Policy $\pi_{\theta_{old}}$로 sample trajectory를 생성하고 이를 이용해서 $\pi_{\theta_{new}}$에대한 기대값을 추정하는 것이다

ex)

$$E_{x\sim p}[f(x)] = \int p(x)f(x)dx = \int \frac{p(x)}{q(x)}q(x)f(x)dx = E_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]$$

importance Sampling

- $\nabla U(\theta) = \mathbb{E}_{\tau \sim \pi_{old}}[\frac{\nabla P(\tau|\theta)}{P(\tau|\theta_{old})} R(\tau)]$
  - 현재 Policy $\pi_{\theta_{old}}$ 로 새로운 policy π_θ 의 performence를 추정하고싶다!
- if, $\theta = \theta_{old}$
  - $\nabla U(\theta)|_{\theta=\theta_{old}} = \mathbb{E}_{\tau \sim \pi_{old}}[\frac{\nabla P(\tau|\theta)|_{\theta_{old}}}{P(\tau|\theta_{old})} R(\tau)]$
  - 본질 적으로, $\theta = \theta_{old}$ 일때는 likelihodd ratio (vanila policy gradient) 방법과 같다
- 의의 : importance sampling을 통해서 과거의 sample 재 사용 할수 있다 (off-policy)
  - 기존에는 사용하고 버려야 했음

importance Sampling

- $\theta \approx \theta_{old}$ 비슷 하다면 근사적으로 같다
  - 너무 격차가 많이 나면 사용할 수 없다! (step size 결정 문제)
- 비슷 하다면이 어느정도까지 되야되는가? → TRPO , PPO idea의 시작점
- 가능한 큰 step size로 update를 하고 싶은데 안전한 범위는 무엇인가?
  - (Trust Region Policy Optimization)

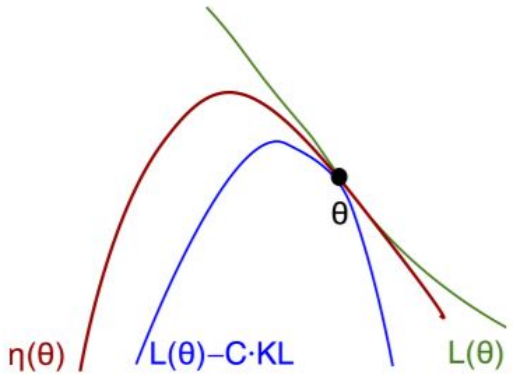
TRPO 맛보기

- policy가 변하는 양을 제한 시킨다
  - policy가 변하는 양을 측정하는 지표 : KL divergence를 이용
    - Define the following trust region update:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

    - Consider KL penalized objective

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] - \beta\hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$



TRPO 맛보기

1) 제약조건이 있는 최적화 문제를 푸는 것은 **매우** 어려움
2) TRPO에서는 Jaccobian/Hessian을 활용해서, 비선형 함수를 선형함수로 근사 + quadratic programming을 반복적으로 사용해서 근사적으로 품
3) 최적화 과정이 매우느리고 메모리 연상량도 매우 큼
4) 실용적으로 사용하기 힘듬!

Proximal Policy Optimization

PPO idea

PPO의 출발점

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

1) TRPO와 같은 질문에서 시작 : Update를 좀더 안전하게 하고싶다
   - 지정한 KL divergence 범위안에서 constraint optimization problem
2) TRPO : constraint를 2차 근사로 푸는 법은 너무 어렵다, 좀더 쉽게 풀 수는 없을까?
3) tf, pytorch 등 Auto diff library 잘되어 있으니깐 1차 근사로 끝내보자

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right].$$

Clipped Surrogate Objective

- $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
- $r(\theta_{old}) = 1$

TRPO

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right].$$

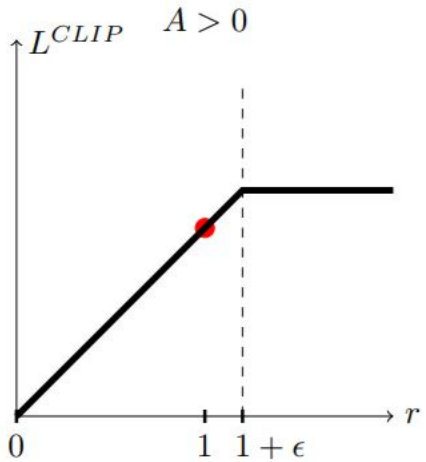
- KL 제약 조건을 만족하면서 위의 식을 maximize
- 제약조건이 없을시 vanila policy gradinet

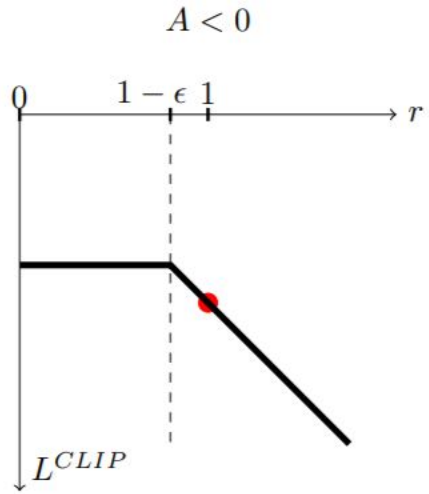
Clipped Surrogate Objective

PPO

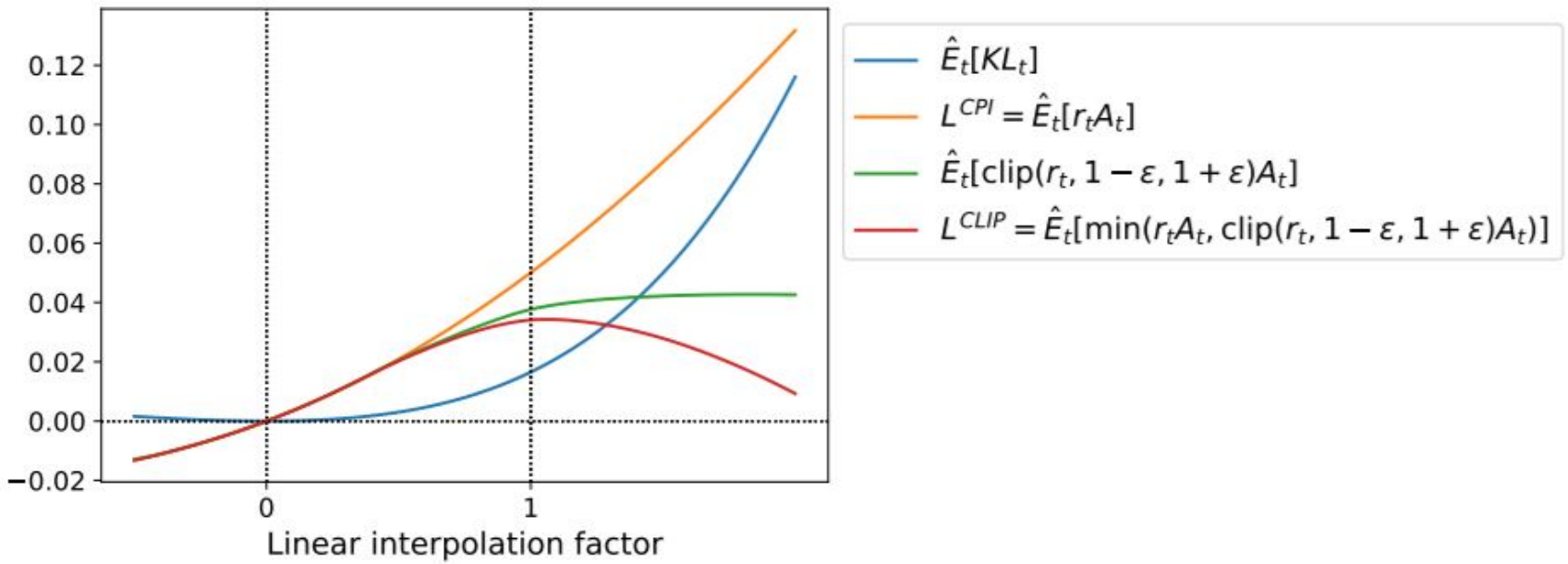
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

- min(Original Loss, Cipped Loss) : 둘중 더 작은 것을 쓰자
- Lower bound of unclipped object L^{cpi}
  : 그러면 원래 목적함수의 Lower bound가 된다





Clipped Surrogate Objective



- $L^{clip}(\theta) \leq L^{cpi}(\theta)$
- $L^{clip}(\theta)$ 가 $L^{cpi}(\theta)$ 보다 항상 $\pi(\theta)$ 를 보수적으로 업데이트 한다!

Adaptive KL Penalty Coefficient

- 논문에서 제시한 두번째 방법
- TRPO에서는 Penalty 형태로 풀지 않고 constraint 형태로 풀었음
  - because it is hard to choose a single value β that performs well across different problems

Adaptive KL Penalty Coefficient

- Using several epochs of minibatch SGD, optimize the KL-penalized objective

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

- Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$
  - If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

- 어떤 기준선 d를 정해놓음, d_{target}
  - 타겟보다 커지면 β 패널티를 2배를 줌
  - 타겟보다 작아지면 β 패널티를 1/2배를 줌
- 그러나 실험적으로 성능이 Clipped 방식보다 떨어진다고 함

PPO Algorithm

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

- 대다수의 On-policy PG 의 경우 sample을 한번 사용하고 버렸지만
- PPO의 경우 전체 샘플을 한번에 업데이트 하지 않고 여러번 (K epochs) 만큼 update
- PPO는 L^{clip} 의 구조 덕분에 실질적으로 π_θ 와 $\pi_{\theta_{old}}$ 가 너무 멀어지면 파라미터 업데이트가 되지 않기 때문에 가능

Result

Result

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

No clipping or penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t$

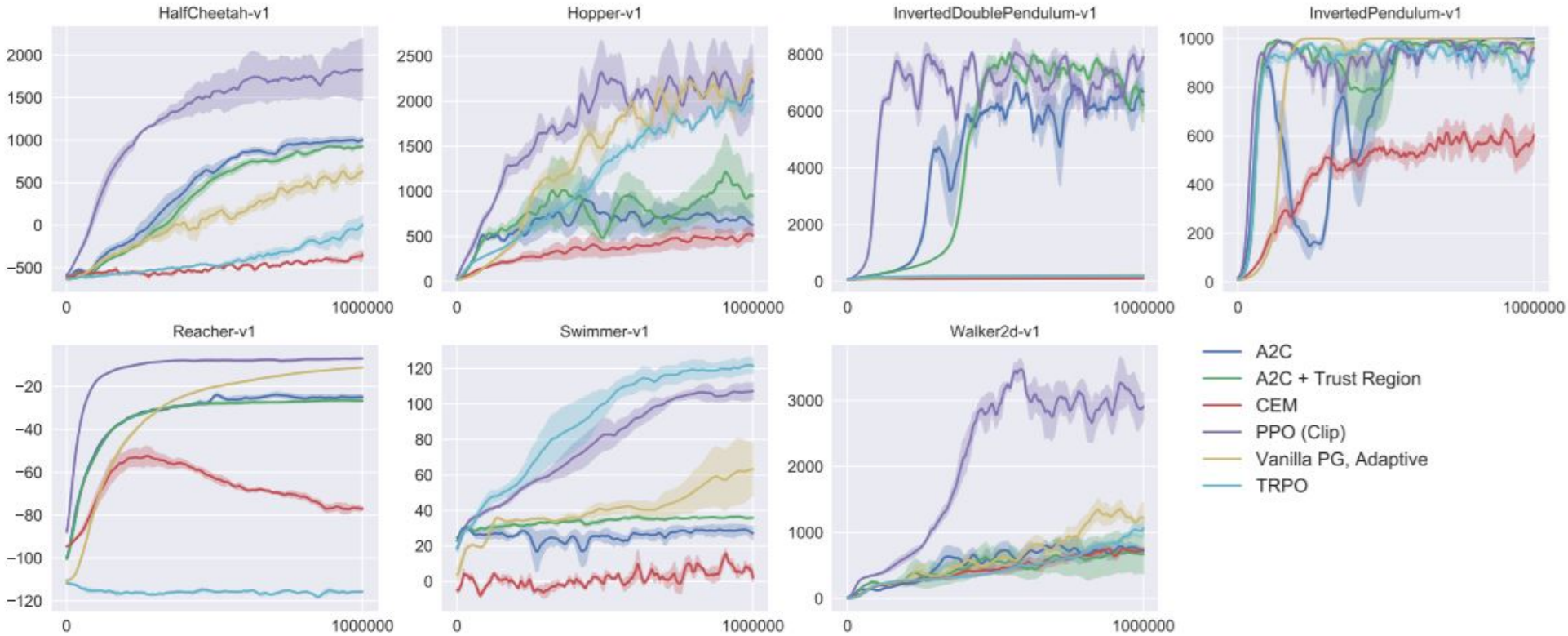
Clipping: $L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$

KL penalty (fixed or adaptive) $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

- Mujuco 환경 7개의 문제에 대해서 3번씩 학습시킴 → 21개의 평균 점수를 구함
- 마지막 100 episode의 평균 성능을 측정
- 이후 random policy 0 가장좋은 policy 1 기준으로 normalize

Result

- Continuous Domain에서의 알고리즘 비교



"Success isn't permarnent, and failure isn't fatal. - Mike Ditka"