PDF가 보기 불편하면 다음 사이트에서 보시길
https://hackmd.io/s/S1a6g133V

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

이 논문은 정말 읽기가 좋다. TRPO로 고통받다가 이걸 보는 순간 힐링되는 기분이 들었다. 이상하게 논문이 간단해서 편하게 쓰는 스타일인가보다 했었는데 알고보니 팡요랩 영상에서 이 논문은 어디에 제출하지 않고 아카이브에만 있기 때문이라고 알려주었다. 구현이나 이론면에서 정말 편해서 좋다.

목차

기존 논문과의 차별성

- **DQN** : 왜 잘되는지 정확한 이론이 없음(**poorly understood**), **continuous 문제에 약함**
- **PG** : 데이터를 한 번 쓰고 버림(**데이터 비효율성**), 다양한 문제에 적용하기 힘듦(**not robust**)
- **TRPO** : 이론적으로 **복잡함**(2차 근사), 복잡한 네트워크(출력이 두 스트림으로 나뉜다거나 dropout 같은 노이즈 들어간 형태)와 잘 맞지 않음

PPO는 결국 저 위의 문제들을 해결한 알고리즘이다. 근본적으로 TRPO와 같이 안정적인 업데이트를 위한 방법이면서 KL divergence나 2차근사를 사용하지 않기 때문에 어렵지않게 구현할 수 있는 좋은 알고리즘이라고 생각한다.

배경지식: 정책 최적화 (Policy Optimization)

Policy Gradient

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

- θ: 정책 파라미터
- π: 확률적인 정책(stochastic policy)
- $\hat{\mathbb{E}}$: 어떤 샘플들로 구한 평균값
- $\hat{A}$: Advantage function

위 식을 미분해서 gradient를 구하면 policy gradient가 된다.

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

기존의 policy gradient는 trajectory를 저장했다가 여러 번 학습시켜볼 수도 있으나 경험상(empirically) 가끔 엄청 큰 업데이트(destructively large policy update)를 해버려서 성능이 안좋아진다고 한다. 애초에 이론상 on-policy이기 때문에 현재 정책에 대한 trajectory만 사용해야 한다.

결국 Policy gradient의 문제점인 **데이터 비효율성**을 얘기하고 싶은 것이다.

Trust Region Methods

Constraint

TRPO에서는 KL divergence가 δ 이하인 상태로 surrogate objective를 최대화하도록 업데이트하게 했었다.

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

그런데 이를 최적화 하려면 1차 미분도 하고 2차 미분도 한 후에 conjugate gradient를 사용해야 한다. (난 여전히 conjugate gradient가 어떻게 동작하는 지 모른다.) 근데 딥러닝 프레임워크들은 1차 미분만 지원하기 때문에 이를 구현하기가 아주 복잡해진다.

Penalty

그리고 TRPO는 constraint 방식 말고 penalty 방식이 존재했다. 이는 β라는 값으로 패널티량을 조절해서 업데이트 크기가 커지는 것을 막았다.

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

constraint와 달리 1차미분으로 구할 수 있지만, β를 아주 잘 선택해야 한다는 단점이 있다.

논문에서는 위 방법을 각각 따로 해결책을 제시한다.

- Clipped Surrogate Objective
- Adaptive KL Penalty Coefficient

Clipped Surrogate Objective

$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$라고 하자.

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

> 여기서 CPI는 TRPO에서 나왔던 conservative policy iteration을 의미한다.

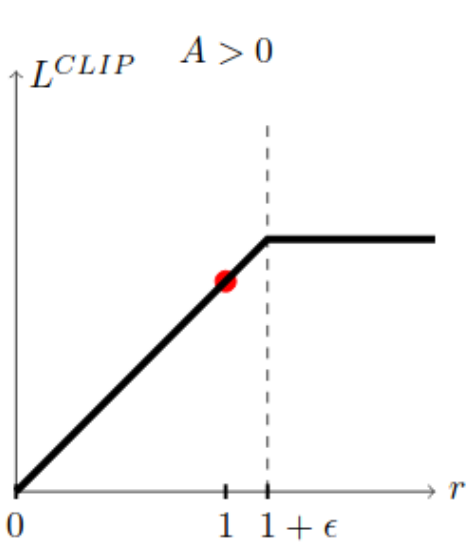
$r(\theta_{old})$라면 분모 분자 이 동일해서 값이 1이 된다. ($\frac{\pi_{\theta_{old}}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} = 1$) 즉, 정책이 동일하다면 1인 것이고 1에서 멀어질 수록 두 정책의 차이가 커진다는 것으로 볼 수 있다. 그렇다면 1에서 멀어지지 않도록 일정 값(ϵ)을 두어 그 값만큼만 안 벗어난다면, 정책도 많이 차이 나지 않을 것이다.

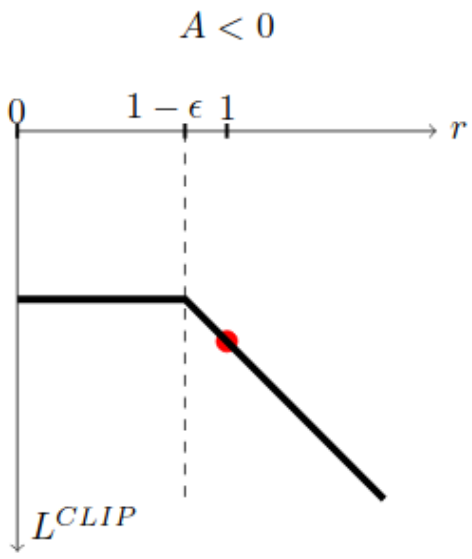
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

다소 복잡해 보이지만, 사실 아주 간단하다. $\min$은 두 가지 중 작은 하나를 고른다.

1. $\hat{\mathbb{E}}_t \left[r_t(\theta)\hat{A}_t \right]$, surrogate objective
2. $\hat{\mathbb{E}}_t \left[\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right]$, clipped surrogate objective

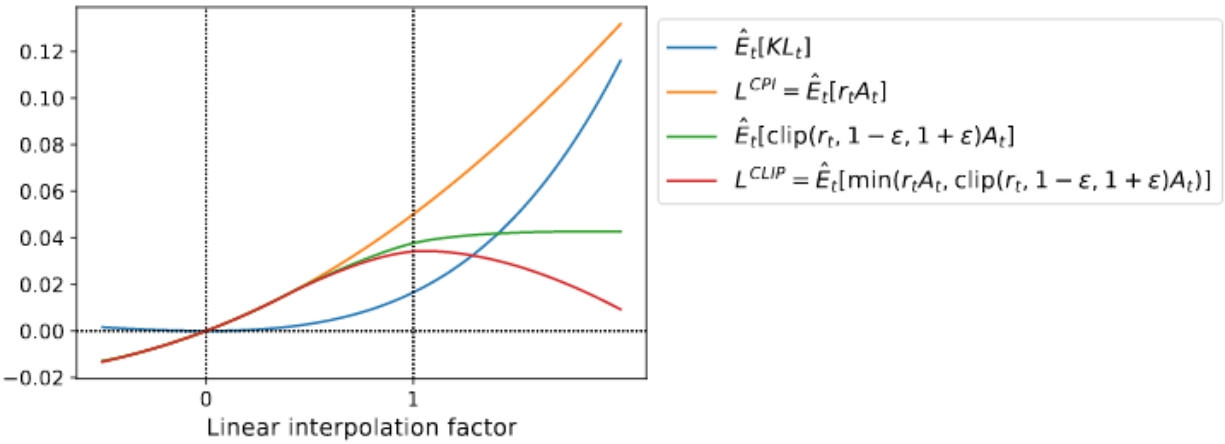
이 숫자들이 과연 어떤 범위로 가질지 감이 안잡히기 때문에 그림으로 보는 것이 좋다.





- $A > 0$ 양수일 때
  - Advantage가 양수 → 좋은 행동 → 새로운 정책 확률 증가 → $r_t(\theta)$ 값 증가
  - $r_t(\theta)$가 일정 이상으로 증가하면 $1+\epsilon$으로 clip
- $A < 0$ 음수일 때
  - Advantage가 음수 → 나쁜 행동 → 새로운 정책 확률 감소 → $r_t(\theta)$ 값 감소
  - $r_t(\theta)$가 일정 이상으로 작아지면 $1-\epsilon$으로 clip

결과적으로 π_θ가 덜 변하도록 제한시키는 것으로 볼 수 있다.



위 그림을 보면 결국 Clip한 L^{CLIP}(빨간선)을 보면 surrogate objective(주황선)과 clip한 objective(초록색) 중에서 작은 값의 평균이기 때문에 1이상으로 갈수록 내려가는 것을 볼 수 있다. (라고 PG에서는 얘기를 하는데 정확히 왜인지는 잘 모르겠다.)

Adaptive KL Penalty Coefficient

Constraint 문제 대신 Penalty 문제를 해결하기 위해 β를 고정하지 않고, 알아서 자동으로 바뀔 수 있게 (adaptive) 하는 방법을 사용한다. (이것이 위에 나왔던 clip 방법보다 성능이 더 좋았던 것은 아니지만, 다른 누군가가 언젠가 발전시킬 수 있으므로 그를 위한 baseline으로 만든 것 같다.)

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

objective 식이 딱히 바뀐 것은 아니다. 다만 β가 이제 고정된 값이 아니라 KL divergence가 d_{targ}라는 값과 얼마나 차이가 나는 지에 따라 달라진다.

- If $d < d_{targ}/1.5,\ \beta \leftarrow \beta/2$
- If $d > d_{targ}/1.5,\ \beta \leftarrow \beta \times 2$

여기서 위 방식에서 1.5나, 2같은 숫자는 경험상으로 구한 값이고 d_{targ} 또한 그렇게 크게 민감하지 않다. 왜냐하면 빠르게 알아서 조정되기 때문이다.

Algorithm

Loss L^{CLIP}과 L^{KLPEN}에 대해 구해보았다. 물론 이렇게 해서 구현해도 좋지만 Advantage처럼 분산을 줄이는 방법으로 GAE나 finite-horizon estimator를 사용할 수도 있을 것이다. 어쨌든 이를 위해서는 policy와 value라는 두 개의 값을 필요로 하다. Network가 별개로 있다면 value는 $L_t^{VF} = (V_\theta(s_t) - V_t^{targ})^2$, policy는 위에서 구했던 L_t^{CLIP}을 이용해 별개로 최적화하면 되지만, 하나의 네트워크에서 두 개의 출력이 나온다면 이를 하나로 합쳐서 loss를 구해야 한다. exploration을 위해 entropy(S)를 추가할 수도 있다.

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t) \right]$$

$c1, c2$는 계수로써 합쳐질 때 나름 조정해줘야하는 하이퍼 파라미터이다. 하지만 논문에서는 이렇게 하지않고 그냥 네트워크를 따로 쓴다. 그리고 Entropy도 사용하지 않는다. (그럼 왜 얘기한ㄱ..)

GAE에서는 Return값을 구하고 γ와 λ에 따라 TD-λ처럼 1-step부터 n-step까지의 Advantage 값을 잘 조율했다. 여기서 다만 달라지는 것은 Return이 아니라 일정 T step까지만 딱 자르고 거기까지의 보상의 합을 이용한다. 즉, episode가 끝나기도 전에 일정시간마다 보상의 합을 구하고 이를 통해 Advantage를 구한다. (Truncated version of GAE)

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$$

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$
$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

T, γ, λ 또한 어디까지 하느냐가 관건이지만, 이에 대해서는 경험적으로 구할 수밖에 없다.

이제 PPO actor-critic을 알고리즘으로 표현한 의사코드를 보면 정리가 좀 된다.

Algorithm 1 PPO, Actor-Critic Style

```
for iteration=1, 2, ... do
    for actor=1, 2, ..., N do
        Run policy π_θold in environment for T timesteps
        Compute advantage estimates Â_1, ..., Â_T
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size M ≤ NT
    θ_old ← θ
end for
```

Experiments

< L^{PG} vs L^{CLIP} vs L^{KLPEN} >

- OpenAI gym MuJoCo
- 각각 100만 time step 실험
- policy, value network 두 개 별개로 사용
- entropy 사용하지 않음
- MLP(64 뉴런을 가진 hidden 레이어 두 층, tanh)
- 출력은 어떤 정해준 표준 편차를 갖는 가우시안 분포의 평균
- 7개의 환경(HalfCheetah, Hopper, InvertedDoublePendulum, InvertedPendulum, Reacher, Swimmer, and Walker2d, all “-v1”)
- random seed 3가지 사용 (결국 총 7 x 3 = 21가지 결과)
- 끝에서 100 에피소드 보상 평균을 구함

- random policy (0), 가장 최고의 결과를 (1)로 정규화(normalization)

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

Comparison to Other Algorithms in the Continuous Domain

$\epsilon = 0.2$인 PPO를 다른 알고리즘들과 비교한 결과이다.

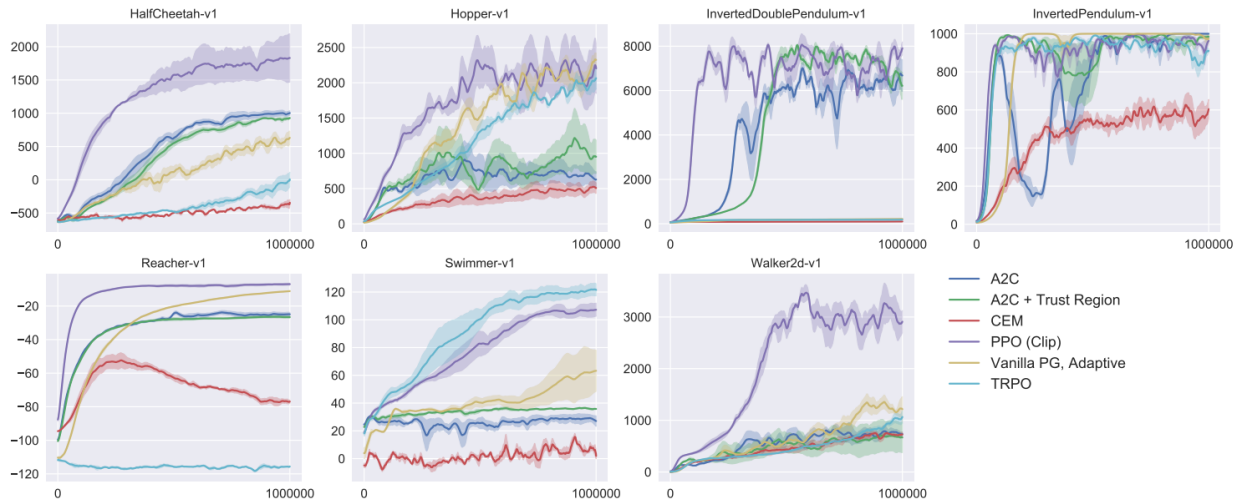


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

Showcase in the Continuous Domain: Humanoid Running and Steering

OpenAI gym 환경 중에 하나인 Roboschool을 사용

- RoboschoolHumanoid : 앞으로 걸어가기
- RoboschoolHumanoidFlagrun : 200 step마다 다른 위치에 나타나는 타겟에 걸어가기

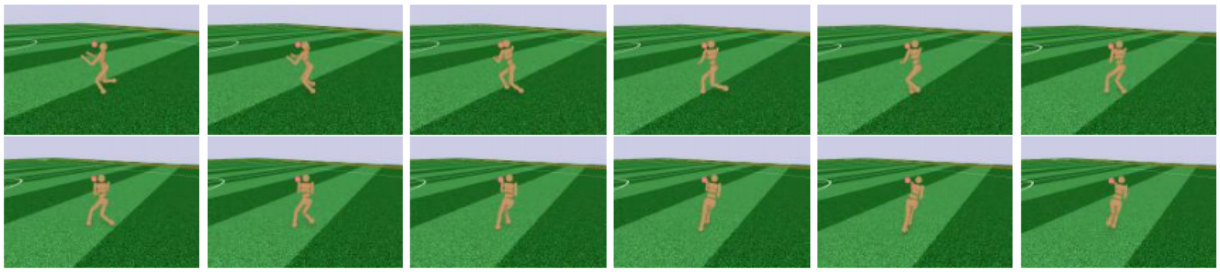


Figure 5: Still frames of the policy learned from RoboschoolHumanoidFlagrun. In the first six frames, the robot runs towards a target. Then the position is randomly changed, and the robot turns and runs toward the new target.

- RoboschoolHumanoid-FlagrunHarder : 다 똑같지만 큐브가 날아와서 로봇이 움직이는걸 방해한다.

Comparison to Other Algorithms on the Atari Domain

Atari에서 비교. A3C 논문에서 나온 Network와 동일하게 사용한다. 약간 독특한 비교를 하는데 보통은 끝날 때즈음 성능이 얼마나 좋은지를 보는데 전체적인 평균도 비교를 한다. 팡요랩의 추측에 의하면 생각보다 ACER에 비해 성능이 안나와 이렇게 비교하는 것이 아닐까 한다.

	A2C	ACER	PPO	Tie
(1) avg. episode reward over all of training	1	18	**30**	0
(2) avg. episode reward over last 100 episodes	1	**28**	19	1

Hyperparameters

Hyperparameter	Value
Horizon (T)	2048
Adam stepsize	3×10^{-4}
Num. epochs	10
Minibatch size	64
Discount (γ)	0.99
GAE parameter (λ)	0.95

Table 3: PPO hyperparameters used for the Mujoco 1 million timestep benchmark.

Hyperparameter	Value
Horizon (T)	512
Adam stepsize	*
Num. epochs	15
Minibatch size	4096
Discount (γ)	0.99
GAE parameter (λ)	0.95
Number of actors	32 (locomotion), 128 (flagrun)
Log stdev. of action distribution	LinearAnneal$(-0.7, -1.6)$

Table 4: PPO hyperparameters used for the Roboschool experiments. Adam stepsize was adjusted based on the target value of the KL divergence.

Hyperparameter	Value
Horizon (T)	128
Adam stepsize	$2.5 \times 10^{-4} \times \alpha$
Num. epochs	3
Minibatch size	32×8
Discount (γ)	0.99
GAE parameter (λ)	0.95
Number of actors	8
Clipping parameter ϵ	$0.1 \times \alpha$
VF coeff. c_1 (9)	1
Entropy coeff. c_2 (9)	0.01

Table 5: PPO hyperparameters used in Atari experiments. α is linearly annealed from 1 to 0 over the course of learning.

Reference

- 논문 원본
- OpenAI Spinning-up PPO
- PG여행 7번째 논문 PPO
- 팡요랩의 '쉽게읽는 강화학습 논문 6화 PPO 논문 리뷰'