IMPLEMENTATION MATTERS IN DEEP POLICY GRADIENTS
: A CASE STUDY ON PPO AND TRPO

곽윤혁

Motivation (Why?)

- Measuring reward can be tricky/unreliable.
  - 랜덤 시드에 따라서 성능 차이가 심하게 난다. (벤치마크 결과를 신뢰할 수 없다. 알고리즘이 진정 발전했는지 알기 힘들다.)
  - 하이퍼 파라미터 변화에도 민감하다.

- 같은 알고리즘이라도 구현체에 따라 성능 차이가 심하다.
  - 가끔 아주 심플한 베이스라인에도 성능을 역전당하기도 한다.

- 알고리즘 간 비교를 위해서는 성능 향상의 정확한 원인 파악이 필요하다.

미리 보는 이 논문의 contribution

- 알고리즘 간 비교를 위해서는 성능 향상의 정확한 원인 파악이 필요하다.
  - 논문에는 언급이 없지만 구현체에는 들어가 있는(실험에는 사용한) 여러 optimization들이 이를 어렵게 만든다는 것을 확인했다.
- PPO가 TRPO에 비해 잘되는 이유는 trust region enforcement mechanism의 변화보다는 사소한 내용으로 치부되었던 code-level optimizations(CLO) 덕분인 것 같다.
- Reliable한 DRL알고리즘의 발전을 위해 각 부분을 모듈화해서 쉽게 갖다 쓸 수 있게 하고 여러 방면으로 정확히 성능에 기여하는 요소를 실험하자.

Ablation study

- PPO와 TRPO에 대한 case study를 진행하였다.
- 공식 PPO 구현체에는 TRPO에서 메인 알고리즘만 바뀐 게 아닌 논문에 언급이 없는 다른 code-level optimizations들도 추가되어 있었다.
- 별거 아닌 것처럼 보였으나 ablation study 결과 이것들은 성능에 상당한 영향을 끼치는 것으로 밝혀졌다.

PPO에서 사용된 code-level optimization

1. Value function clipping
2. Reward scaling
3. Orthogonal Initialization and layer scaling
4. Adam learning rate annealing
5. Reward clipping
6. Observation normalization
7. Observation clipping
8. Hyperbolic tan activations
9. Global gradient clipping

이 중 첫번째 네 개에 대해 ablation study 진행.

PPO에서 사용된 code-level optimization

1. Value function clipping

$$L^V = (V_{\theta_t} - V_{targ})^2,$$ 대신

$$L^V = \max\left[(V_{\theta_t} - V_{targ})^2, (\text{clip}\left(V_{\theta_t}, V_{\theta_{t-1}} - \varepsilon, V_{\theta_{t-1}} + \varepsilon\right) - V_{targ})^2\right],$$ 사용

ε is fixed to the same value as the value used to clip probability ratios in the PPO loss function

PPO에서 사용된 code-level optimization

2. Reward scaling
Discount-based scaling scheme.
The rewards are divided through by the standard deviation of a rolling discounted sum of the rewards

ref.) Appendix A.2.

PPO에서 사용된 code-level optimization

3. Orthogonal Initialization and layer scaling

ref.) Exact solutions to the nonlinear dynamics of learning in deep linear neural networks - Saxe, A. et al. (2013)

PPO에서 사용된 code-level optimization

4. Adam learning rate annealing

starting with a relatively high learning rate and then gradually lowering the learning rate during training.

(Adam은 이미 adaptive method인데도 사용)

PPO에서 사용된 code-level optimization

5. Reward clipping
clips the rewards within a preset range (usually [–5, 5] or [–10, 10]).

PPO에서 사용된 code-level optimization

6. Observation normalization
 the states are first normalized to mean-zero, variance-one vectors.

PPO에서 사용된 code-level optimization

7. Observation clipping
the observations are also clipped within a range, usually [−10, 10].

PPO에서 사용된 code-level optimization

8. Hyperbolic tan activations
hyperbolic tangent function activations between layers in the policy and value networks.

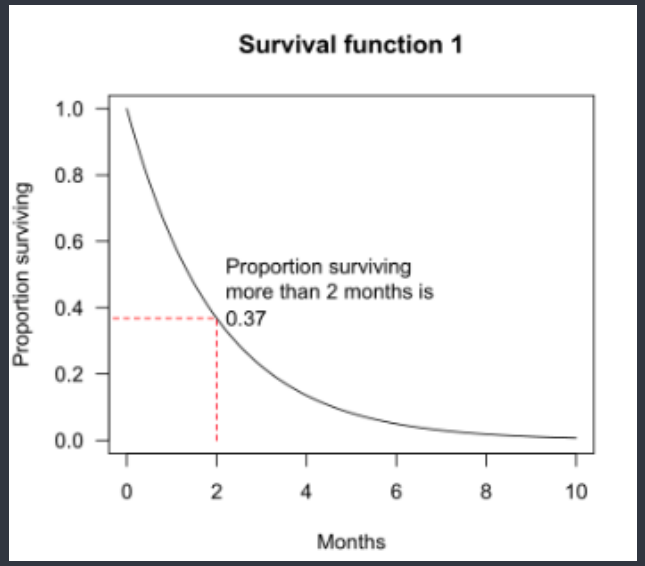
PPO에서 사용된 code-level optimization

9. Global gradient clipping
After computing the gradient with respect to the policy and the value networks, the implementation clips the gradients such the “global l2 norm” (i.e. the norm of the concatenated gradients of all parameters) does not exceed 0.5.

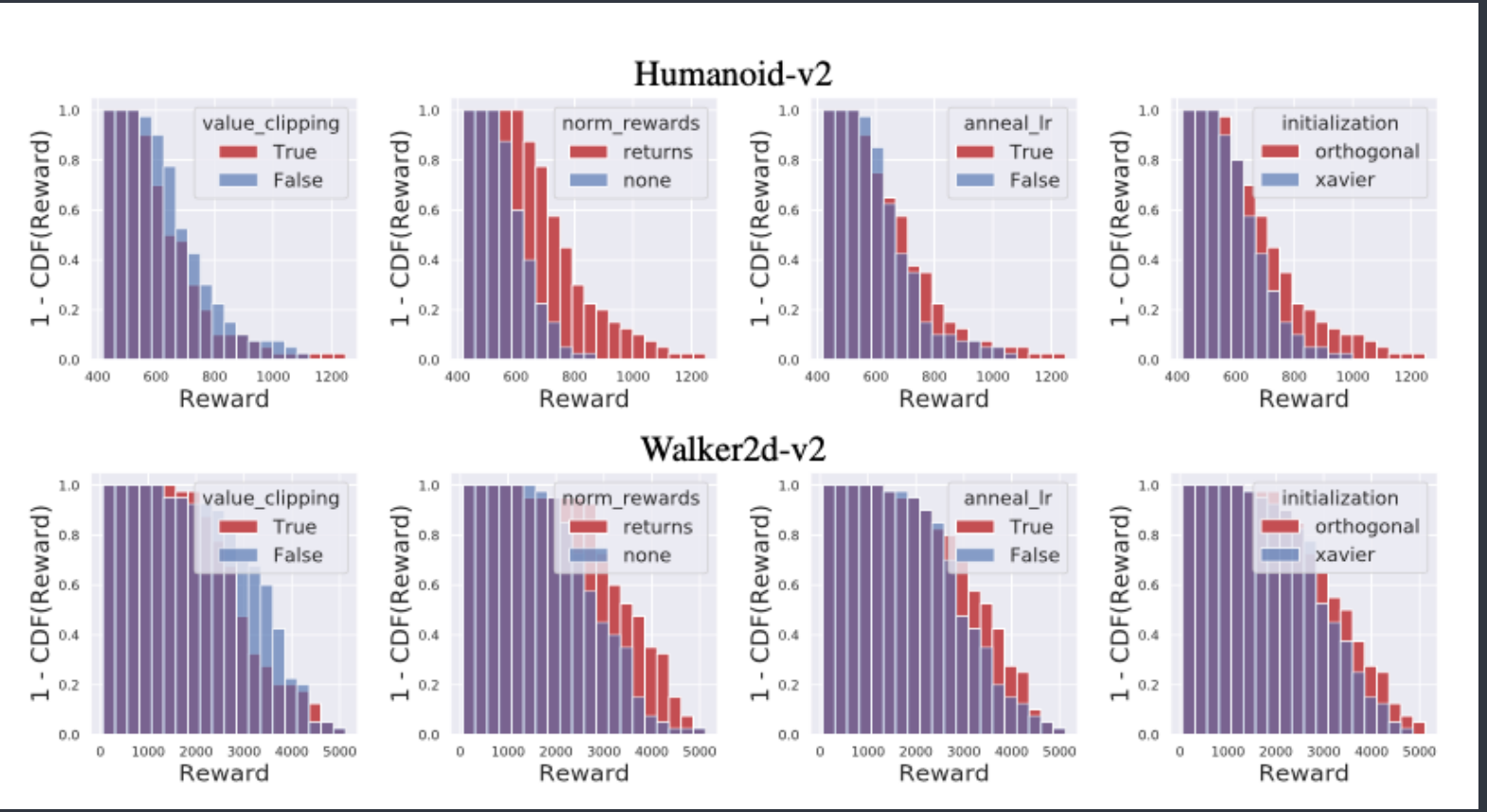
실험 결과를 보기에 앞서

Survival function

- 환자가 일정 기간 뒤에 살아남을 확률
- Inverse CDF 또는 Complementary CDF라고도 불림
- S(t) = 1-CDF(t). S: survival function, CDF: Cumulative distribution function



Ablation study 결과



PPO 구현체를 대상으로
*ablation study 진행

Reward normalization,
Adam learning rate annealing,
orthogonal initialization
의 영향이 크다는 것을 확인
(reward landscape를 바꾼다.)

* Ablation study: 어떠한 요소를 제거해봄으로써 해당 요소가 미치는 영향을 알아보는 방법

AAI VS ACLI

- AAI: Average algorithmic improvement
- ACLI: Average code-level improvement

$$\text{AAI} = \max\{|\text{PPO} - \text{TRPO+}|, |\text{PPO-M} - \text{TRPO}|\},$$

$$\text{ACLI} = \max\{|\text{PPO} - \text{PPO-M}|, |\text{TRPO+} - \text{TRPO}|\}.$$

TRPO+:
TRPO + PPO's CLO

PPO-M:
PPO – PPO's CLO

	MuJoCo Task		
Step	Walker2d-v2	Hopper-v2	Humanoid-v2
PPO	3292 [3157, 3426]	2513 [2391, 2632]	806 [785, 827]
PPO-M	2735 [2602, 2866]	2142 [2008, 2279]	674 [656, 695]
TRPO	2791 [2709, 2873]	2043 [1948, 2136]	586 [576, 596]
TRPO+	3050 [2976, 3126]	2466 [2381, 2549]	1030 [979, 1083]
AAI	242	99	224
ACLI	557	421	444

ACLI > AAI인 것을 확인할 수 있다.

PPO-NoCLIP

- 심지어 PPO-M* 보다 PPO-NoCLIP*의 성능이 균일하게 더 좋다.

	Walker2d-v2	Hopper-v2	Humanoid-v2
PPO	3292 [3157, 3426]	2513 [2391, 2632]	806 [785, 827]
PPO (baselines)	3424	2316	—
PPO-M	2735 [2602, 2866]	2142 [2008, 2279]	674 [656, 695]
PPO-NoClip	2867 [2701, 3024]	2371 [2316, 2424]	831 [798, 869]

*PPO-M: PPO – CLO (policy ratio clipping은 여전히 적용)
*PPO-NoCLIP: PPO – policy ratio clipping (CLO는 여전히 적용)

Contribution

- 알고리즘 간 비교를 위해서는 성능 향상의 정확한 원인 파악이 필요하다.
  - 논문에는 언급이 없지만 구현체에는 들어가 있는(실험에는 사용한) 여러 optimization들이 이를 어렵게 만든다는 것을 확인했다.

- PPO가 TRPO에 비해 잘되는 이유는 trust region enforcement mechanism의 변화보다는 사소한 내용으로 치부되었던 code-level optimizations 덕분인 것 같다. (CLO가 trust region을 확보해주는 역할을 하였다.)

- Reliable한 DRL알고리즘의 발전을 위해 각 부분들을 모듈화해서 쉽게 갖다 쓸 수 있게 하고 여러 방면으로 정확히 성능에 기여하는 요소를 실험하자.

그렇다면

- 더더욱 코드 공개를 하지 않는 논문을 믿기 힘들다.
- 연구자 스스로도 자신의 연구의 성능 향상에 대한 원인을 찾기 위해 책임감있는 ablation study가 필요하다.
- 현실적 문제. 안그래도 환경에 따라 달라지는 성능도 측정해야 하고, 하이퍼 파라미터 서치도 해야하는 데, CLO 단위로까지 실험을 해야한다?!?
  - Code-level optimization 반영해서 이것이 미치는 성능뿐만 아니라 다른 요소들과의 조합을 생각하려면 code-level optimization이 하나씩 추가될 때마다 필요한 실험의 개수가 두 배씩 늘어난다.
    - 우린 DeepMind, OpenAI가 아닌데 😂

그렇다면

- 현실적으로는 모든 조합을 서치하는 건 경제적/시간적 여건 상 불가능하다.
  - 합리적인 선을 정하되, 실제 연구를 진행할 때는 최신 연구들과, code-level optimization, 모듈화된 라이브러리들을 최대한 활용해서
    참신한 아이디어와 reliable한 progress에 집중하고
    구현에 드는 시간과, 구현에서 발생되는 버그는 줄이자.
  - 연구자 개인의 철저한 실험이 장기적으로는 이 분야의 연구 비용을 줄일 것이다.

- 같은 선상에서 강화학습 알고리즘들을 비교할 baseline과 모듈화된 라이브러리들이 필요하다
  - 완전히 공정한 비교는 있을 수 없다.
  - 주기적으로 여러 알고리즘을 자신만의 동일 선상에 놓고 비교하는 논문들이 나와주면 좋을 것 같다.

🌟 Robust한 알고리즘 연구가 필요하다.