RL 스터디

3주차

Policy base

- 다음과 같이 정책을 바로 파라미터화 해서 함수로 근사시키는 방법

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$

장점

- 빠른 수렴
- 연속되거나 높은 차원의 출력에 강하다.
- 확률적 정책 학습이 편리

단점

- Local optimum에 갇히기 쉽다
- Variance가 높다.

목적함수

- 정책을 학습시키기 위해 해당 정책이 최대화하고자 하는 목적이 목적 함수이고 이를 통해 gradient decent를 진행
- 일반적으로 다음과 같은 목적함수 사용

$$J_{avR}(\theta) = \sum_{s} d^{\pi_\theta}(s) \sum_{a} \pi_\theta(s, a) \mathcal{R}_s^a$$

- 여기서 d는 stationary distribution이라 해서 해당 state를 머물 확률 즉 state의 분포를 나타냄
- 즉 식은 해당 정책을 사용 할때 얻게 되는 보상의 합이다.

REINFORCE – monte carlo gradient

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\
&= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\mathcal{R}_{s,a} \\
\nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_{s,a} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]
\end{aligned}
$$

기대 값으로 표현
=> 확률적 샘플링 사용

$$
\begin{aligned}
\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\
&= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)
\end{aligned}
$$

목적함수

- 옆의 식이 가지는 의미

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right],$$

- 파라미터 공간에서 그라디언트의 방향과 r는 그 크기를 나타낸다.

- 우리는 간헐적으로 r를 사용해서 하기보다 그 의미가 비슷한 여러 지표를 대신 사용하는데
- 여기서는 미래 행동에 대한 예상 보상치인 Q를 사용한다.
- 이에 대한 증명은 여기서는 언급 안함

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty},\\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function).} \tag{3}$$

REINFORCE – monte carlo gradient

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return θ
end function

A3C

- 기존의 high variance 문제 why 바로 return을 이용하기에
- Episode 길이가 길면 학습이 느려짐 => step 별 학습을 원함

- 해결 방법
- Actor critic 이라는 방식 도입
- 비동기적 학습
- Advantage 함수 도입
- N- step 학습
- ....

Actor critic

Critic은 TD 에러를 이용 state action value 최적화

Actor는 critic을 바탕으로 목적함수를 최적화

이를 스텝별로 시행

Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx. $Q_w(s,a) = \phi(s,a)^\top w$

  Critic Updates w by linear TD(0)

  Actor Updates θ by policy gradient

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  end for
end function

A3C

◦ A3C는 Asynchronous Advantage Actor Critic 라서 A3C 이다.

◦ Advantage 함수는 일종의 base line으로 현재 정책이 기존보다 나은지 아닌지를 판단한다.

◦ 여기서 n-step 학습을 도입 성능을 높인다.

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$$

$$\nabla_{\theta'} \log \pi(a_t | s_t; \theta') A(s_t, a_t; \theta, \theta_v)$$

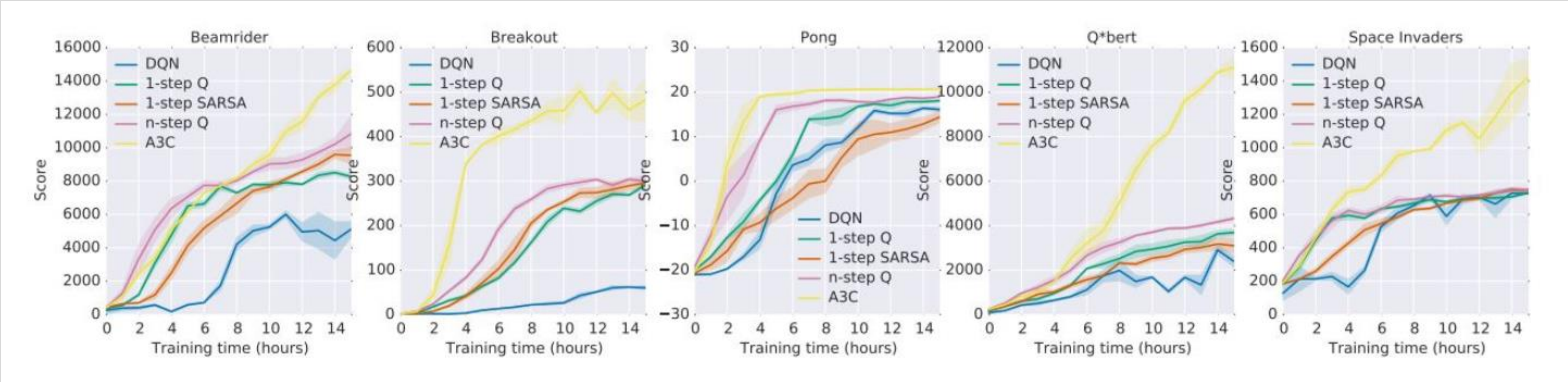
A3C

- 추가적으로 앞서 말했던 local optimal에 빠지는 것을 방지하기 위해 추가적으로 그라디언트에 엔트로피 항을 추가해 학습시킨다.

$$\beta \nabla_{\theta'} H(\pi(s_t; \theta'))$$

- 앞서 나왔던 DQN에서의 replay memory는 없다. 이는 각 데이터별 시간 종속성을 없애기 위함이었는데 이는 비동기적 시행이 해결한다. 여러 개의 쓰레드에서 동시적으로 에이전트가 돌아가며 각 에이전트는 독립적이라 각 에이전트가 보내는 그라디언트는 시간 종속성이 없다. 거기에다 cpu 쓰레드를 사용해서 데이터를 쉽게 주고 받으며 학습 속도도 현저히 올릴 수 있다.

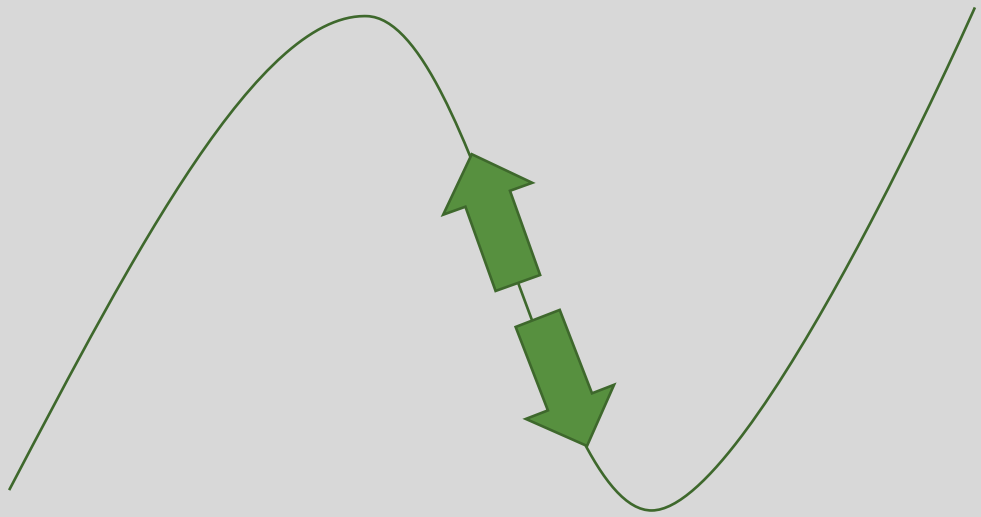
A3C


Beamrider
Breakout
Pong
Q*bert
Space Invaders
DQN
1-step Q
1-step SARSA
n-step Q
A3C
Score
Training time (hours)

TRPO

- Policy gradien의 근본적인 문제란?
- 해당 그라디언트가 성능의 증가를 보장하지 않는다. ???
- 이유는 간단한데 일반적으로 loss 함수는 0으로 수렴시키기 위해 학습하지만 목적함수는 커지게 하기 위해 학습하기 때문이고 그라디언트에 곱해지는 지표가 variance가 너무 높기도 하기 때문이다.

TRPO



2021-09-26

TRPO

- 수학적으로 성능 증가가 보장되는 범위 내에서 학습이 이루어지도록 강제화 한다.
- 여기서 수학적 증명을 설명하기에는 너무나 길다. 본 논문을 찾아서 보도록 하자.

TRPO

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.
for $i = 0, 1, 2, \ldots$ until convergence **do**
Compute all advantage values $A_{\pi_i}(s, a)$.
Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

where $C = 4\epsilon\gamma/(1-\gamma)^2$

and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

TRPO

$$\underset{\theta}{\text{maximize}}\,[L_{\theta_{\text{old}}}(\theta) - CD_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta)]$$

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \le \delta.$$

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

TRPO

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

- 보통 저 surrogate 목적함수를 아래 제약조건내에서 학습 시키는게 좀 까다롭다.
- 가장 구현적으로 편한건 위 아래를 1차 2차 근사화해서 NPG로 학습시키는데 이것도 좀 힘듬

PPO

- 그래서 나온 것이 PPO

- 아래 식은 정책 파라미터에 따라 적절한 베타 선택이 어렵고 variance가 너무 커서 문제가 있었는데

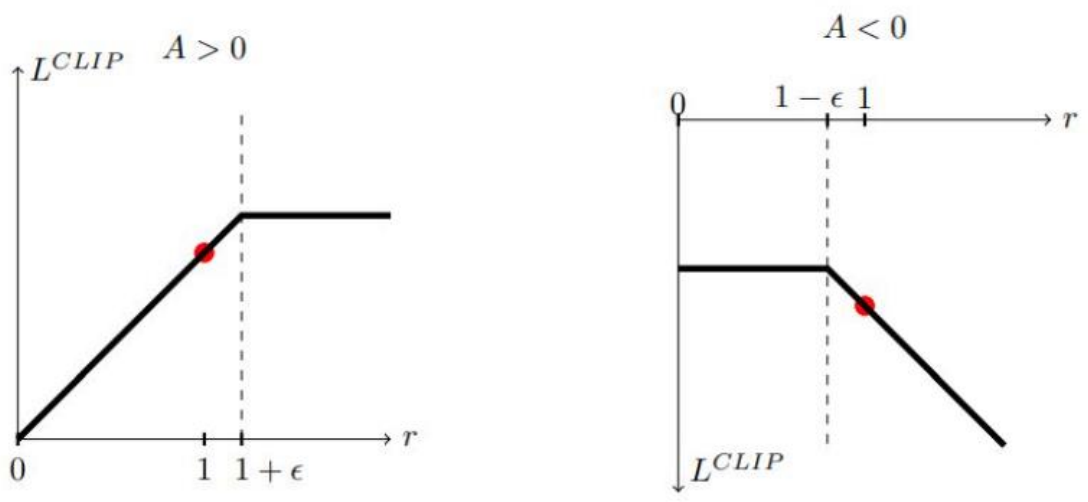
$$\underset{\theta}{\operatorname{maximize}}\, \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

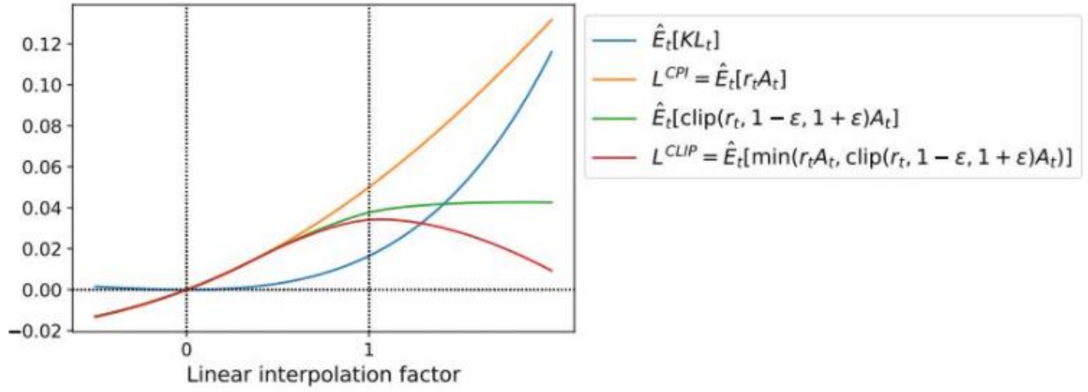
PPO

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

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

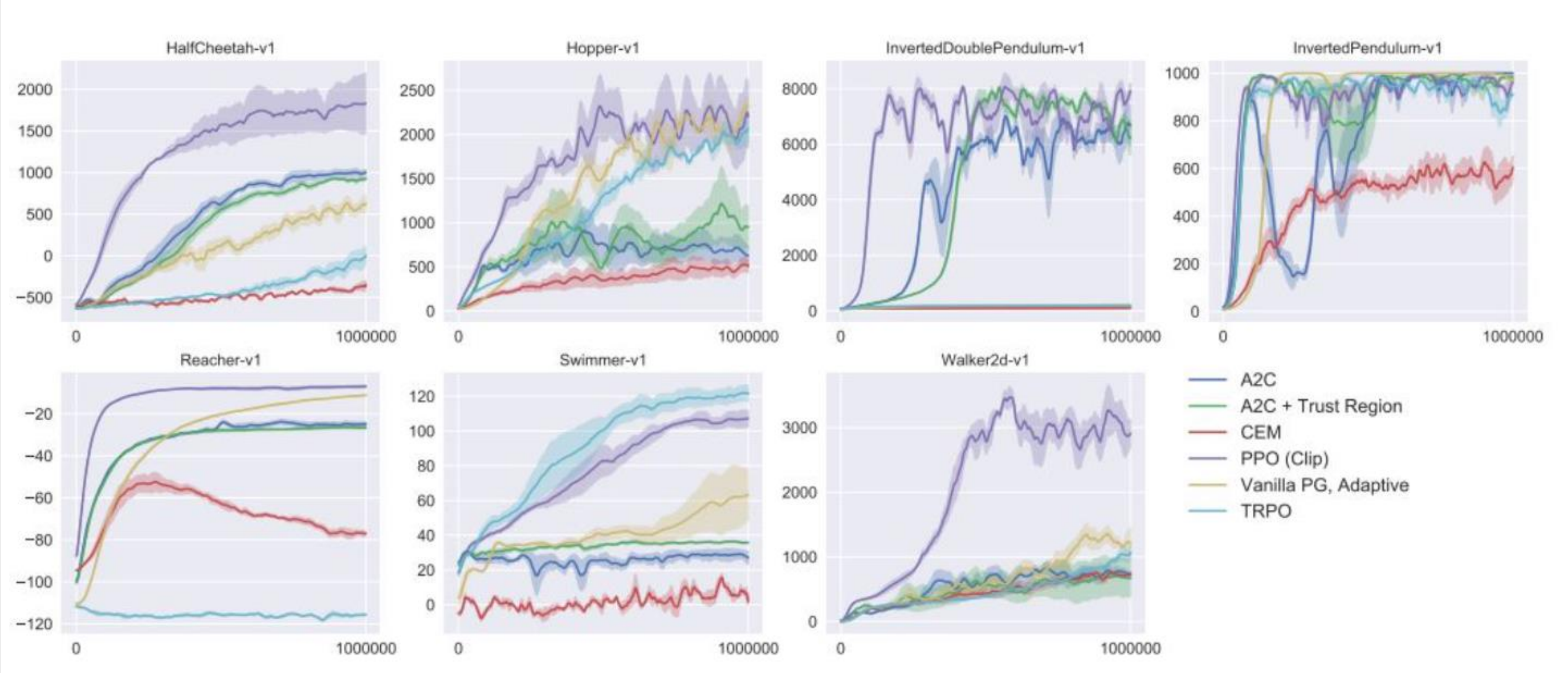
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$





PPO

- 이런식으로 clip과 min 함수가 일종의 lower bound가 되도록 해서 학습의 안정성을 높일 뿐더러 구현도 쉽고 속도도 빠르다.



끝