Doing Deep RL

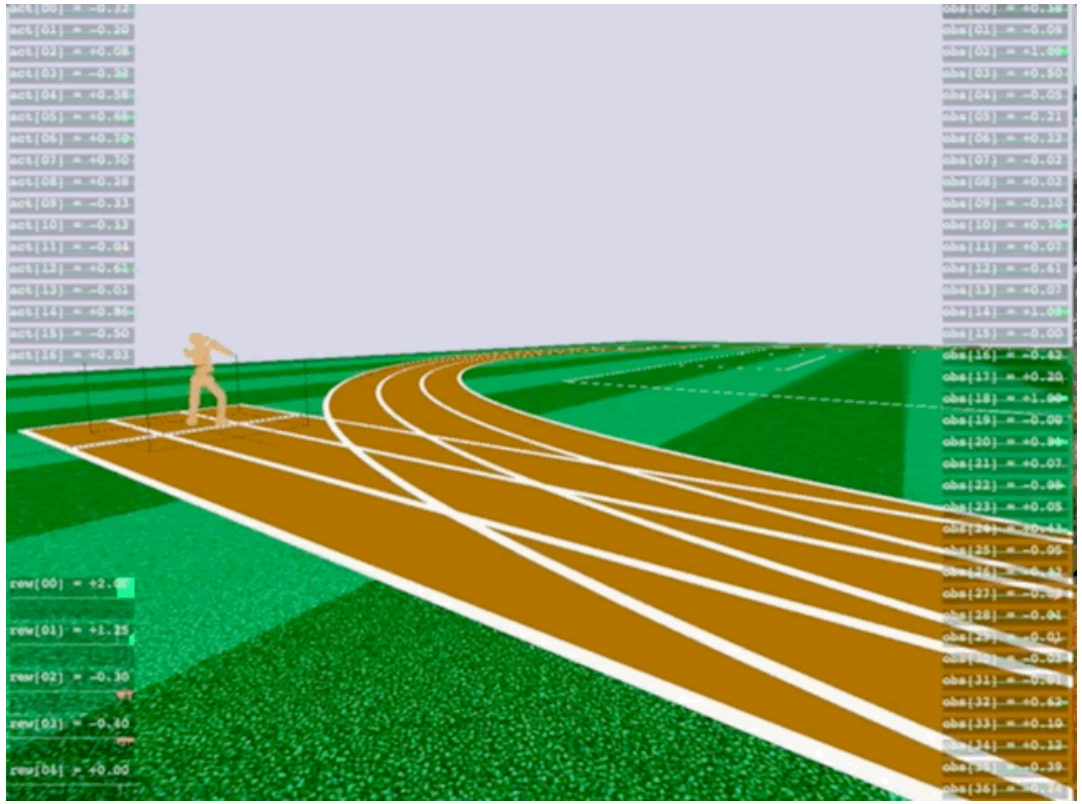
With PPO

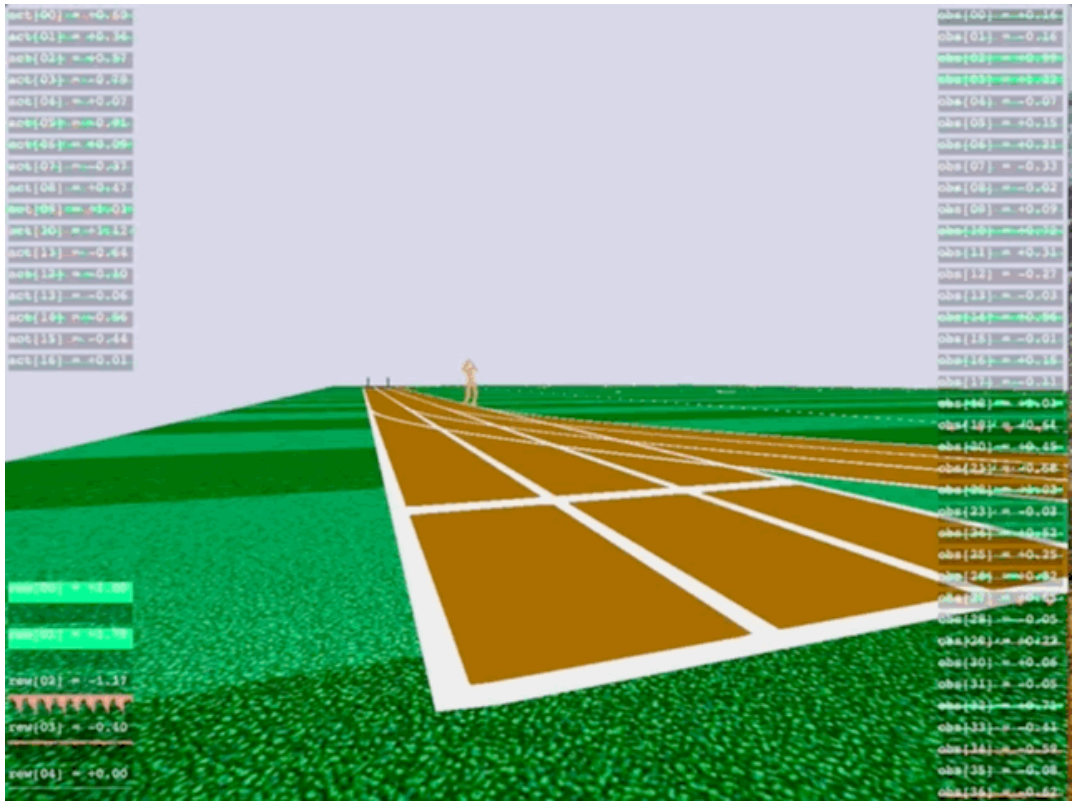
DEVFEST 2017 서울

이의령, 이영무
RLCode

오늘 코드랩 목표!

RoboSchool Humanoid 걷게 만들기





튜토리얼 코드와 설치 가이드

아래 Github 저장소에 오늘 실습 할 코드와 설치 가이드가 있습니다.

https://github.com/wooridle/DeepRL-PPO-tutorial

목차

- 강화학습 Introduction
  - 기초개념 다지기
- Policy gradient 이해하기
  - Policy based RL을 파헤쳐보자
- Actor-Critic 이해하기
  - 강력한 Policy gradient 기반 알고리즘
- PPO(Proximal Policy Optimization) + PPO 코드랩
  - 현재 강력한 강화학습 알고리즘 PPO를 파헤쳐보자

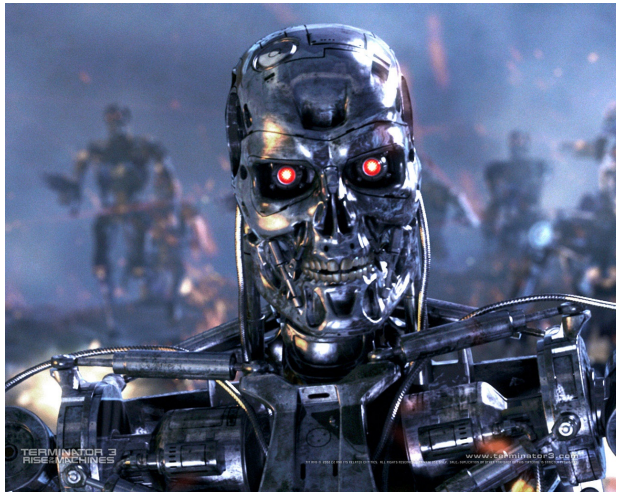
DEVFEST 2017 서울

강화학습 Introduction

DEVFEST

강화학습...?

: 좋은(?) 행동하는 에이전트로 강화시키기



Try - Error - Learning

행동하고 평가받고 학습하고

강화학습

(에이전트가) 순차적으로 좋은 의사결정 방법을 배운다.

Learn to make good sequences of decisions

Don't Know in Advance How World Works
사전에 환경에 대한 모든 정보를 모른다

Reward for Sequence of Decisions
순차적 의사결정에 대한 보상

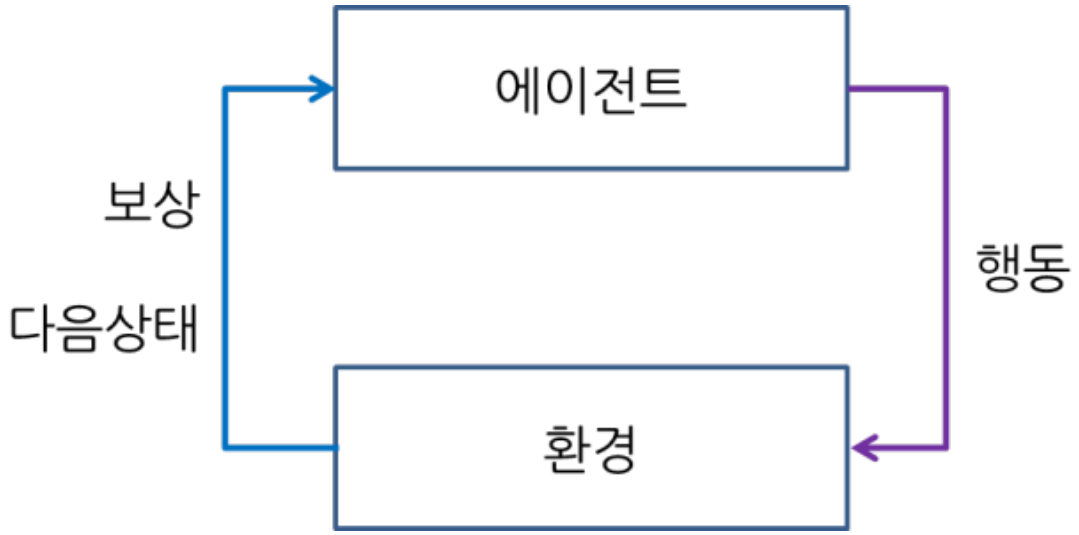
Repeated Interactions with World
반복적인 환경과 상호작용

http://web.stanford.edu/class/cs234/slides/lecture1_introduction.pdf

무엇을 기준으로 학습할까?

: From 에이전트를 둘러싼 환경 + 보상

MDP(Markov Decision Process)



반복! 반복! 반복!

$$s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_T$$

상태 : 환경으로 부터 받는 Input

행동 Action : 연속적 or 이산적

보상 Reward

상태변환확률 $P_{ss'}^{a}$: 다음 상태로 넘어갈 확률

할인율 γ : 미래가치에 대한 가중치

정책 $\pi(a \mid s)$: 특정 상태에서 특정 행동을 할 확률

Q-함수, 가치(V) 함수

가치함수 $v(s) = \boldsymbol{E}[R_{t+1} + \gamma R_{t+2} + \cdots \,|S_t = s]$

지금 상태에서 미래에 받을 것이라 기대하는 보상의 합 = 가치함수

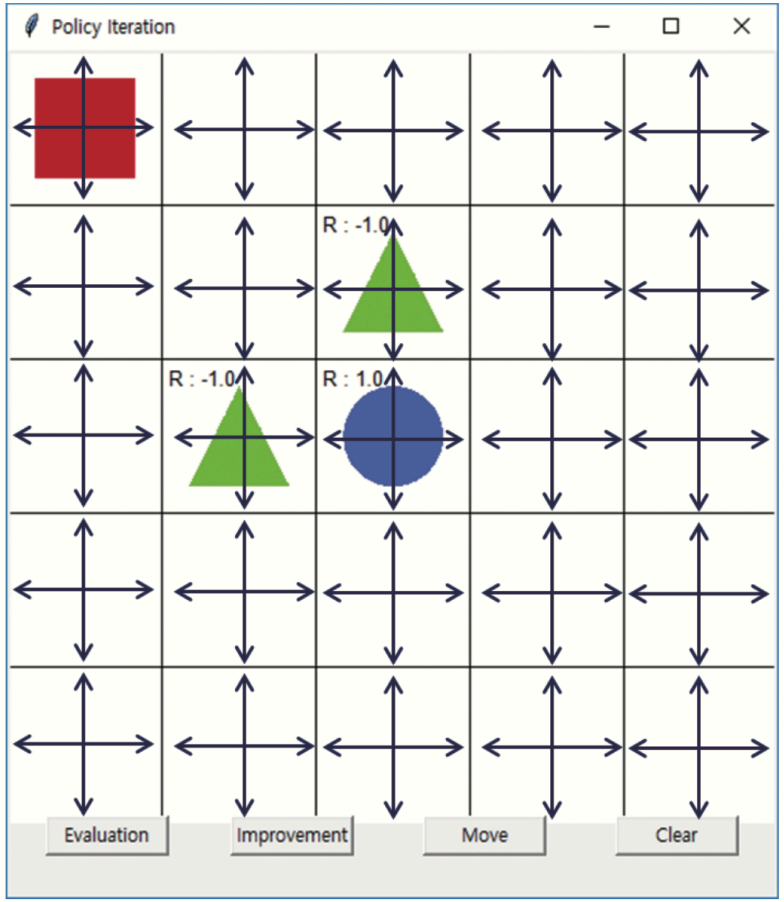
큐함수 $q(s,a) = \boldsymbol{E}[R_{t+1} + \gamma R_{t+2} + \cdots \,|S_t = s, A_t = a]$

지금 상태에서 이 행동을 선택했을 때 미래에 받을 것이라 기대하는 보상의 합 = 큐함수

행동을 고려한 가치 함수 = 큐함수

간단한 예제로 이해해보자!

그리드 월드



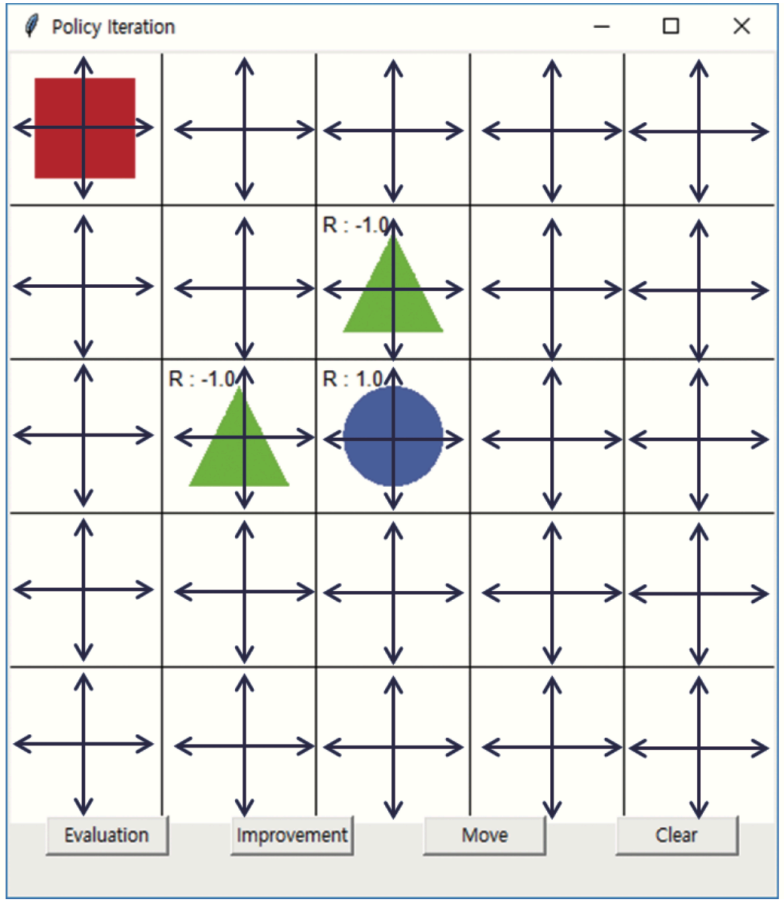
- 목표 :
  에이전트(빨간 네모)가 장애물(초록 세모)을 피해
  목표지점(파란 원)까지 가는 것

- 5 X 5 격자 형태의 환경

- 에이전트, 환경, 상태, 행동, 보상에 대한 정보
  All In One

- 장애물에 도착하면 (-1) 보상
  목표에 도착하면 (+1) 보상

그리드 월드's MDP



- 상태 S = 격자 상의 위치 좌표
- 행동 A = { up, down, right, left }
- 보상 R = { 장애물 (-1), 목표 (+1) }
- 상태변환확률 $P_{ss'}^{a}$ = 다음 상태(s')로 도달할 확률
- 할인율 γ = 미래가치에 대한 가중치
- 정책 $\pi(a \mid s)$ = 특정 상태에서 특정 행동을 할 확률

DEVFEST 2017 서울

Policy Gradient

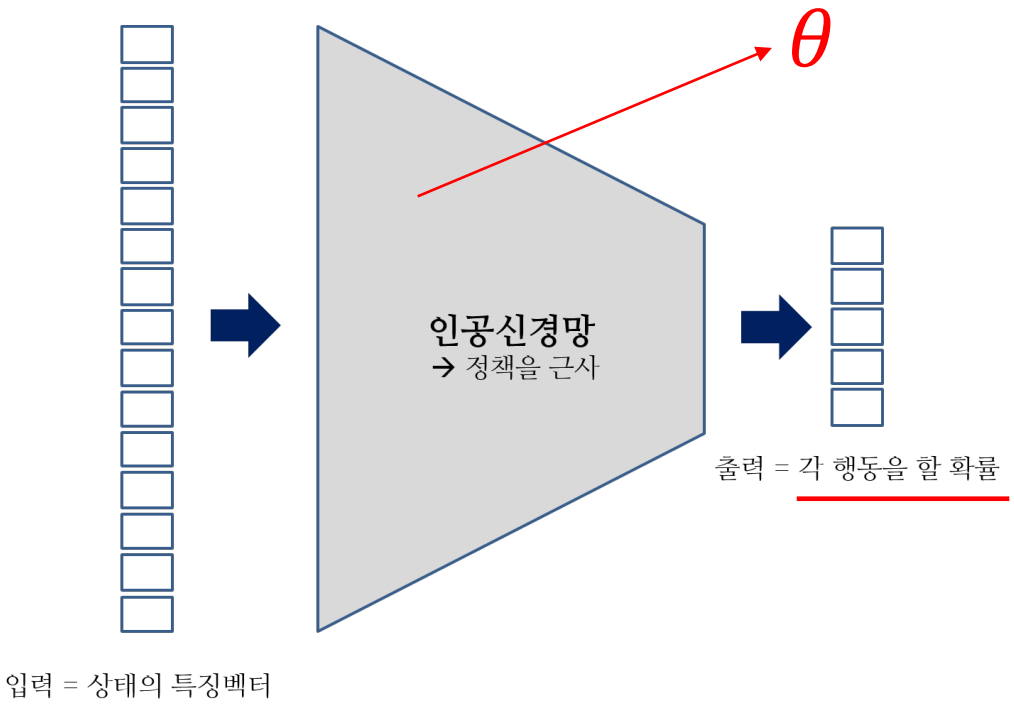
본격적으로 Deep RL!!

Deep Reinforcement Learning

: RL의 Policy (or) Q-value를

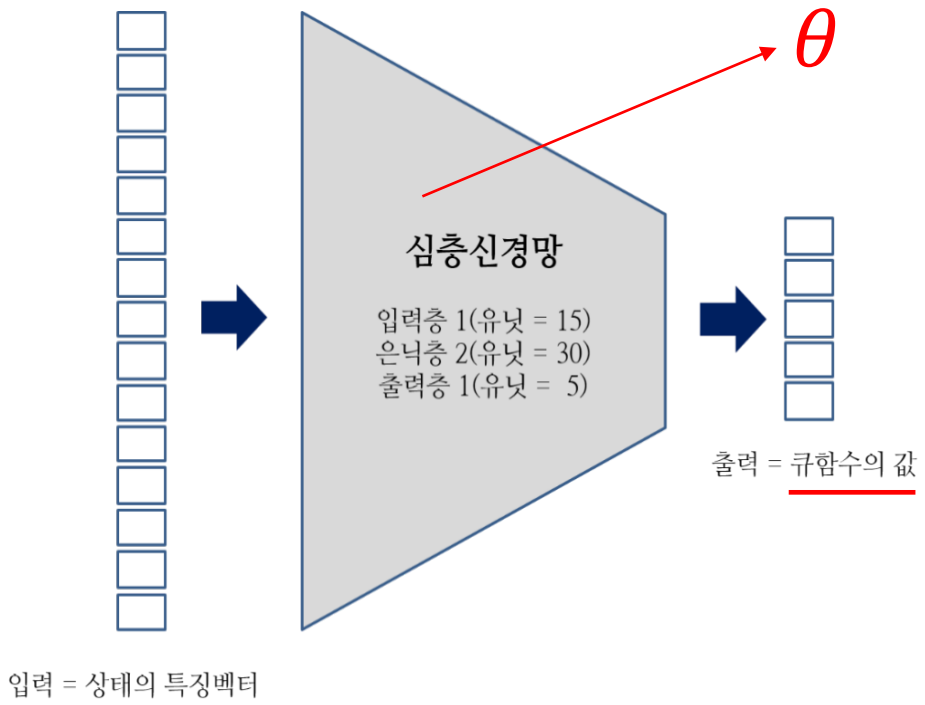
딥러닝이 Approximation 해준다!

Approximation



Policy based RL

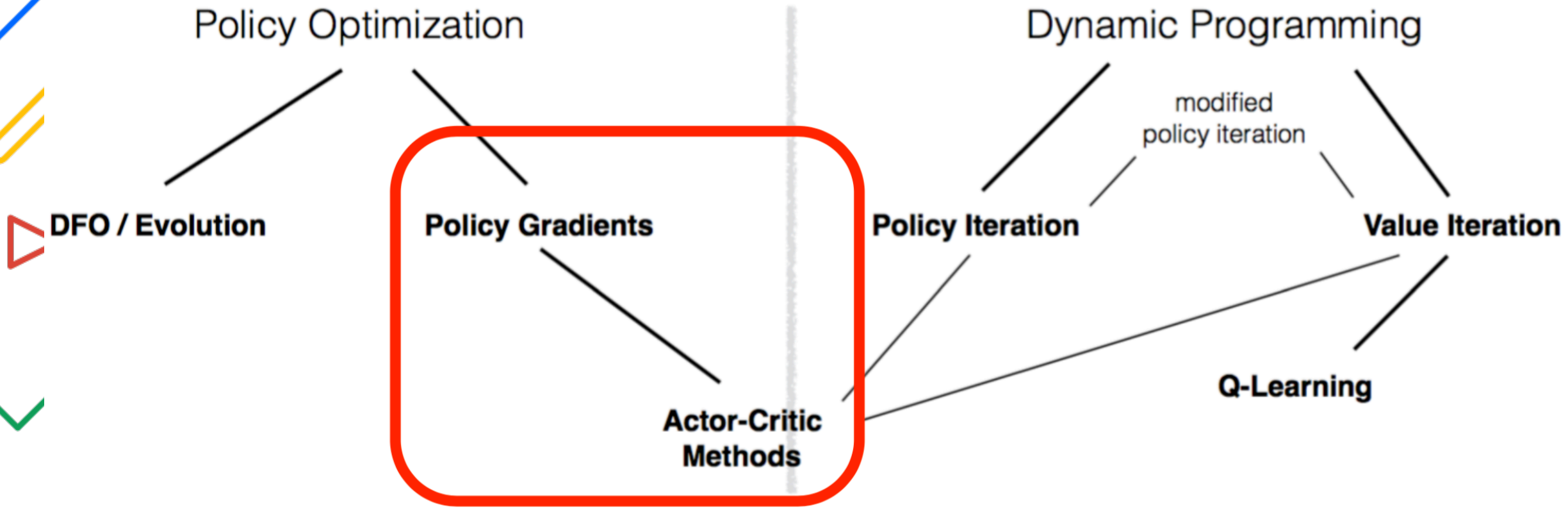
Policy Approximation



Value based RL

Q 함수 Approximation

Reinforcement Learning LandScape



https://sites.google.com/view/deep-rl-bootcamp/lectures

우리가 오늘 볼 PPO는

Policy Optimization을 잘하는 RL

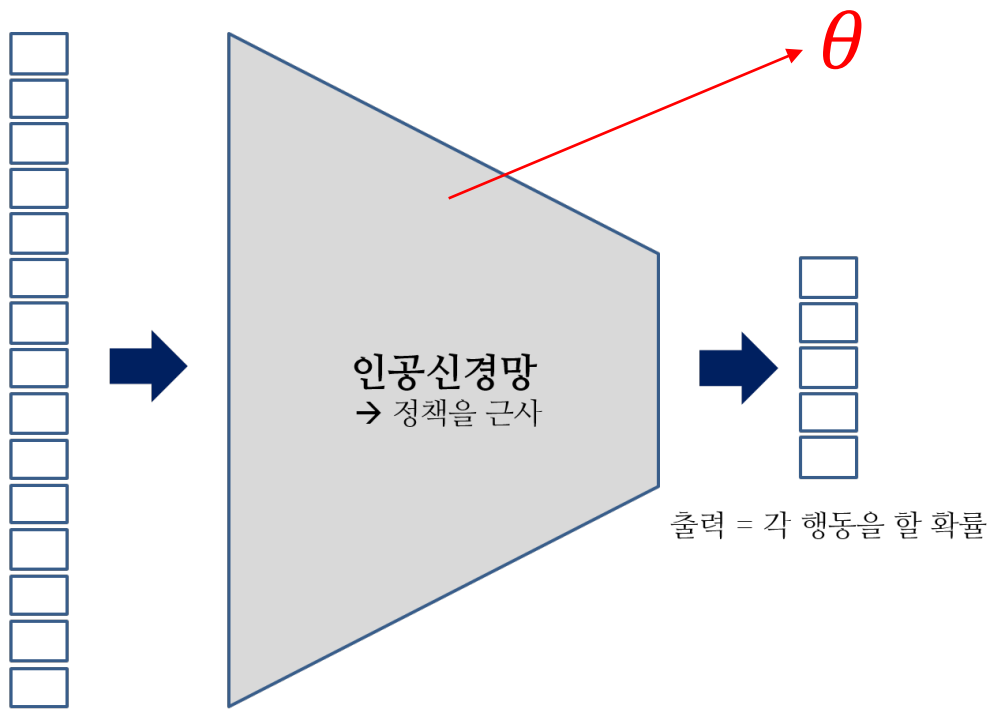
Policy Optimization

: 최적의 정책을 학습하는 것 with 딥러닝

어떻게 정책을 학습하지?

: 우선 Policy Network부터 이해하자

Policy Network



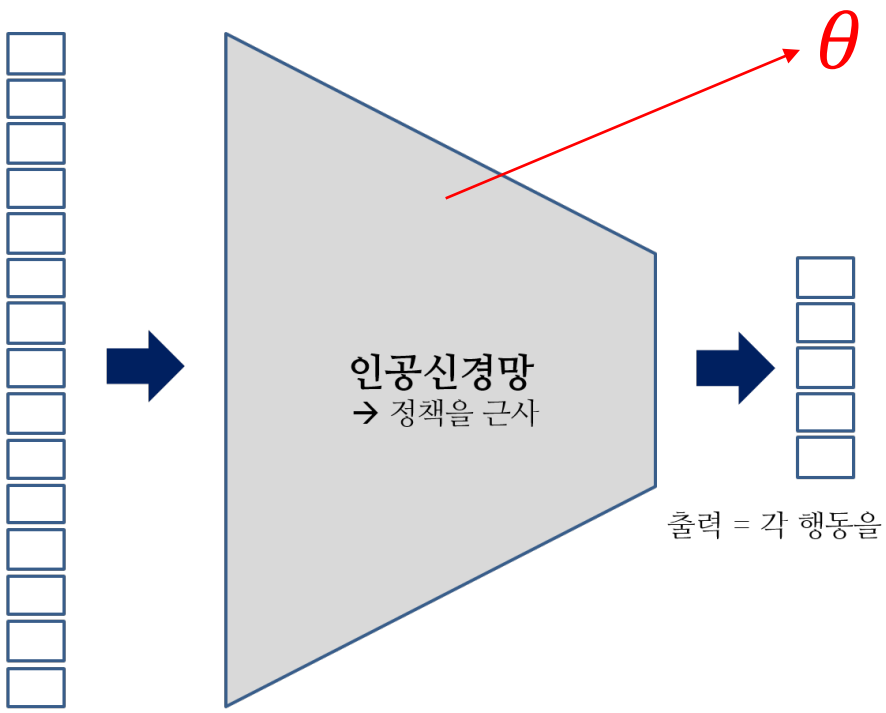
Input : State S

Output : Policy $\pi_\theta(a|s) = \boldsymbol{P}[A_t = a|S_t = s, \theta]$

딥러닝 Weight : 최적의 Policy로 업데이트

최적의 Policy...?

Policy Network



입력 = 상태의 특징벡터

출력 = 각 행동을 할 확률

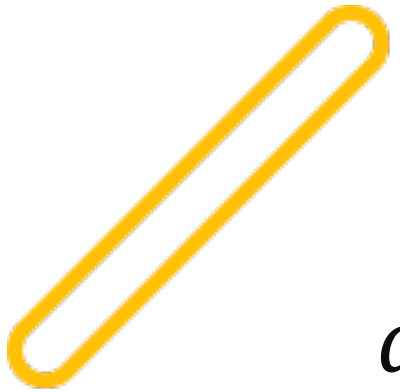
최적 Policy : *보상의 합*을 최대화하는 Policy

$$J(\theta) = \boldsymbol{E}\left[\sum_{t=1}^{T} r_t \mid \pi_\theta\right]$$

$$= \boldsymbol{E}[r_1 + r_2 + r_3 + \cdots + r_T \mid \pi_\theta]$$

즉, 목적함수 $J(\theta)$를 최대화 하는 Policy!!

https://www.slideshare.net/WoongwonLee/rlcode-a3c
RLCode와 A3C쉽고 깊게 이해하기



목적함수를 최대화하는 θ 찾기

$$\underset{\theta}{argmax}\, J(\theta) = \underset{\theta}{argmax}\, E\left[\sum_{t=0}^{T} r_t \mid \pi_\theta\right]$$

$$= \underset{\theta}{argmax} \sum_{t=0}^{T-1} P(s_t, a_t|\tau) r_{t+1}$$

Deep Neural Net이

θ 를 통해 Policy를 근사한다.

Gradient Ascent로 업데이트

: 목적함수를 최대화 하기위해

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

Base 알고리즘

: Policy Gradient이해하기

Vanilla Policy Gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

Policy Gradient

: 에이전트 Policy의 Gradient

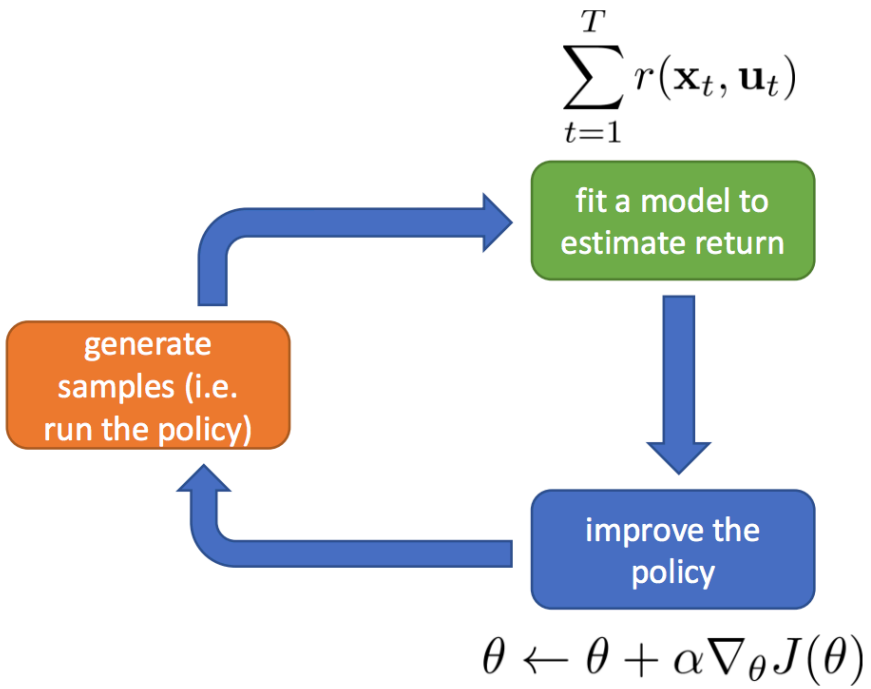
Trajectory의 Reward 총합

: 에이전트의 Policy를 평가

*Trajectory : 경로, 에이전트가 지나간 흔적

$(\tau = s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_T)$

Vanilla Policy Gradient



$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

1. 현재 Policy를 따라 Trajectory 생성 (Sampling의 의미)

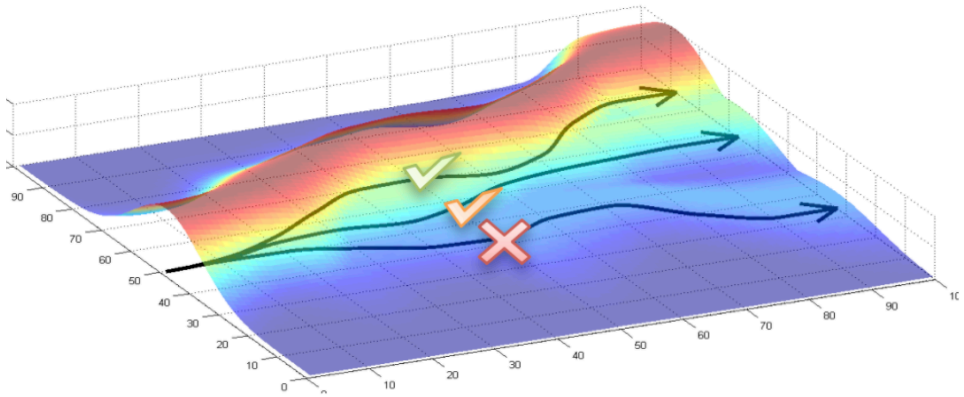
   (Trajectory : $\tau = s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_T$)
2. $\nabla_\theta J(\theta)$ 연산 :

   뉴럴넷이 출력한 $\nabla_\theta log \pi_\theta(a_t|s_t)$ 의 합과 $\sum_{t=1}^{t} r(s,a)$을 곱
3. Gradient Ascent로 Policy 업데이트
4. 1~3 반복

http://rll.berkeley.edu/deeprlcourse/#lecture-videos

Vanilla Policy Gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$



1. 각 path는 한개의 Trajectory를 의미
2. Reward의 총합 ($\sum_{t=1}^{t} r(s,a)$)를 기준으로

   좋은 Path는 더 하게

   나쁜 Path를 덜 하게 에이전트를 학습시킨다.

http://rll.berkeley.edu/deeprlcourse/#lecture-videos

Policy Gradient 한계
: High Variance

High Variance in PG

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \Big(G_t - b(S_t) \Big) \frac{\nabla_{\boldsymbol{\theta}} \pi(A_t | S_t, \boldsymbol{\theta})}{\pi(A_t | S_t, \boldsymbol{\theta})}.$$

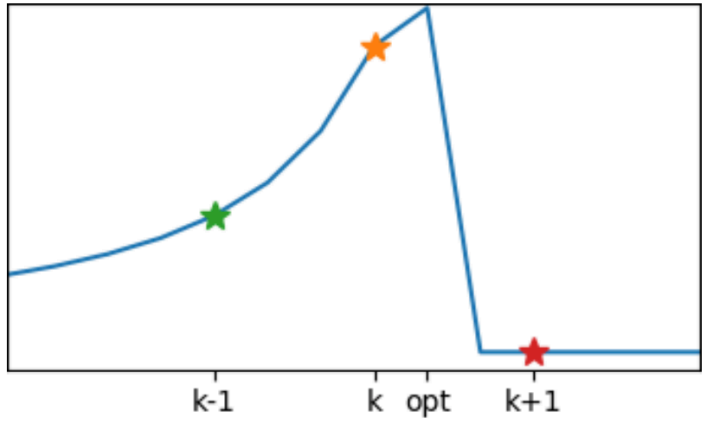
업데이트 크기(step size)

If Step size is

Big : Policy collapse

Small : Slow learning

Need moderate Size!!



- G_t(Reward의 총합)의 Variance가 크면

  업데이트의 크기(Step size)가 안 좋은 영향을 끼친다.

Baseline

$$\nabla_\theta J(\theta) \sim \nabla_\theta log \pi_\theta(a_t|s_t)\left(\sum_{t=0}^{T} r_t(s,a) - V_v(s_t)\right)$$

- Baseline(가치 함수)를 빼준다 → Variance를 낮춰준다.
- Trajectory에서 얻은 (s, a, r, s')로 Baseline(가치 함수)를 구한다.
- $r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)$ = Advantage

큐함수 베이스라인

DEVFEST 2017 서울

Actor-Critic

지금까지
Policy Network는
하나의 네트워크로 학습을 했다.

Policy와 Advantage를 분리해서 별개로 학습한다면?

$$\nabla_\theta J(\theta) \sim \nabla_\theta log \pi_\theta(a_t|s_t)(Q_w(s_t, a_t) - V_v(s_t))$$

Policy Gradient　　Advantage

Actor-Critic

: Policy와 Value를 별개의 Network로 나누어 학습하기

나누면 뭐가좋나요?

: 매번 타임스텝마다 업데이트 할 수 있다.

업데이트 주기 비교

Vanilla Policy Gradient

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)\, G_t$$

에피소드 마다 업데이트

에피소드가 끝날 때 까지 업데이트 X

-> Variance가 크다

Actor-Critic

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t|s_t)\, (r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t))$$

Time Step 마다 업데이트

-> Variance가 작다

Actor-Critic

Actor- Critic

Actor = 정책을 근사

$$\nabla_\theta log\pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\ )$$

Critic = 가치함수(Advantage)를 근사

$$\left(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\right)^2$$

Actor : *행동자, 연기자*

Critic : *비평가, 감시자*

Actor가 정책 신경망을 업데이트 하면

Critic은 가치함수(Advantage)를 지표로 방향성을 평가한다.

Actor-Critic

1. Actor의 loss function

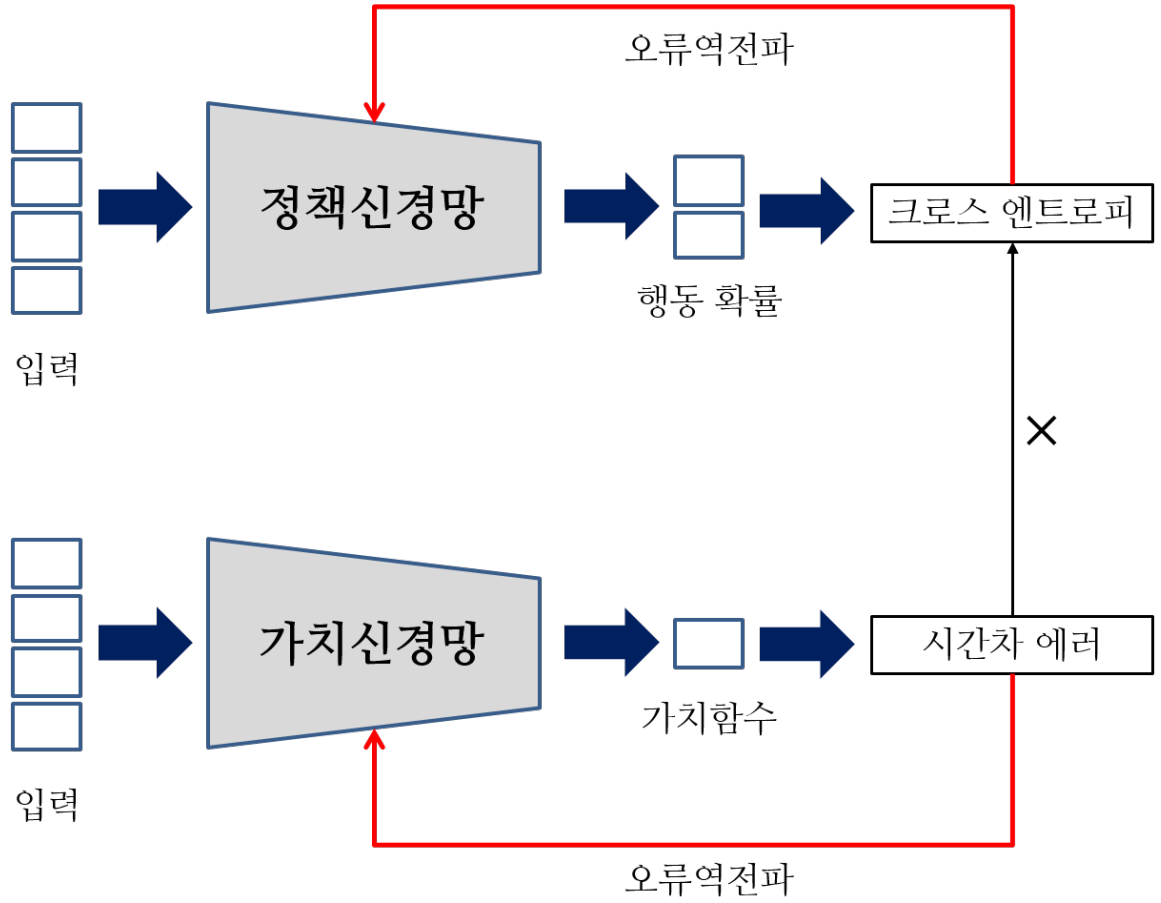
$$\nabla_\theta log\pi_\theta(a_t|s_t)(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t))$$

크로스 엔트로피 시간차 에러

2. Critic의 loss function

$$\left(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\right)^2$$

시간차 에러



시간차 에러 : 한 타임스텝 별로 계산 vs **몬테카를로** : 한 에피소드 별로 계산

https://www.slideshare.net/WoongwonLee/rlcode-a3c
RLCode와 A3C쉽고 깊게 이해하기 / 이웅원

DEVFEST 2017 서울

PPO(Proximal Policy Optimization)

PPO(Proximal Policy Optimization)

현재로썬 강려크하다...

지금까지

1. Policy Gradient의 학습 원리
2. 어떻게 좋은 Advantage를 구할 것인가에 초점

이제부터 Policy를
얼마나 Update 할 것인가의 문제

DEVFEST

Proximal Policy Optimization

: TRPO의 이론과 구현을 간단히 만든 알고리즘

PPO가 강력한 점?

: TRPO의 이론과 구현의 어려움을 단순화 시킴!

TRPO와 PPO의 핵심?

: Policy Performance Bound

Policy Update에 일정한 제한을 둔다.

TRPO(Trust Region Policy Optimization)

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n$$

$$\text{subject to} \quad \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta$$

- New Policy와 Old Policy(이전 Policy)의 차이를 KL-Divergence로 구하고 업데이트의 크기를 δ로 제약(constraint)을 준다.
- KL-Divergence = 두 분포 간의 차이 = 두 Policy 간의 차이
- 2nd Order Gradient 계산을 해야함 = 연산과 구현이 어려움...

목적 함수가

Vanilla Policy Gradient와 다르다.

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n$$

Surrogate function

Surrogate function

$$maximize\ \nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t)\, A_t$$

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n$$

- 지금까지 살펴본 Policy Gradient와 같은 역할을 한다.
- Importance Sampling으로 Policy Gradient식을 유도

PPO는 간단하다.

TRPO는 KL-Divergence로
두 Policy간의 차이를 구했다.
: 2nd order Approximation

하지만 PPO 는
두 Policy간의 비율 = importance Ratio를
Clip하여 제약을 둔다.

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

$r_t(\theta)$ = Importance Ratio

PPO(Proximal Policy Optimization)

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

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

Policy Update : $\theta_{k+1} = \arg\max_\theta \mathcal{L}^{CLIP}_{\theta_k}(\theta)$

- TRPO의 목적함수(Surrogate Loss)를 그대로 가져온 후 $r_t(\theta)$를 제약(constraint) 조건으로 둔다.
- ε (논문에서 0.2)을 기준으로 CLIP 한다.

PPO(Proximal Policy Optimization)

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

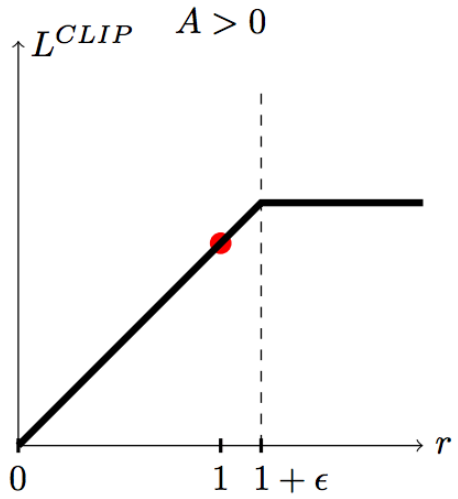
for iteration=1, 2, . . . **do**
  Run policy for T timesteps or N trajectories
  Estimate advantage function at all timesteps
  Do SGD on $L^{CLIP}(\theta)$ objective for some number of epochs
end for

1. Old Policy <- 이전에 업데이트 된 New Policy : Copy
2. Maximize (Surrogate Loss = $\frac{New\ Policy}{Old\ Policy}$ x Advantage)

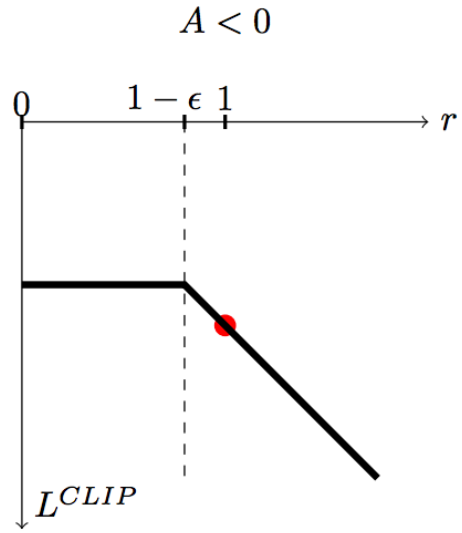
   Time Step을 mini-batch size로 쪼개서 업데이트(Stochastic Gradient Descent)

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}^{CLIP}_{\theta_k}(\theta)$$

PPO(Proximal Policy Optimization)



Advantage가 0보다 클 때



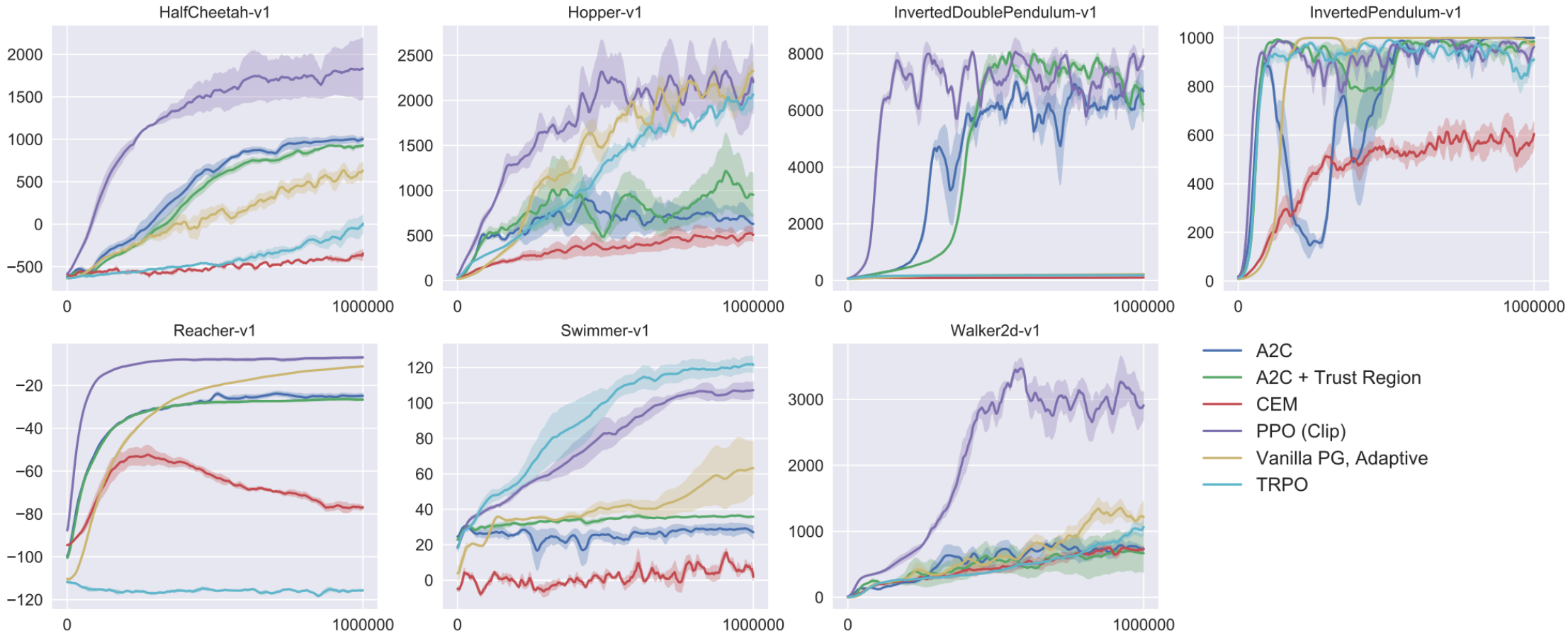
Advantage가 0보다 작을 때

Policy 업데이트가 Clip 기준 이상 커지지 않게 하는 것이 목적

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

PPO 성능 비교(Policy base RL)

PPO : 보라색



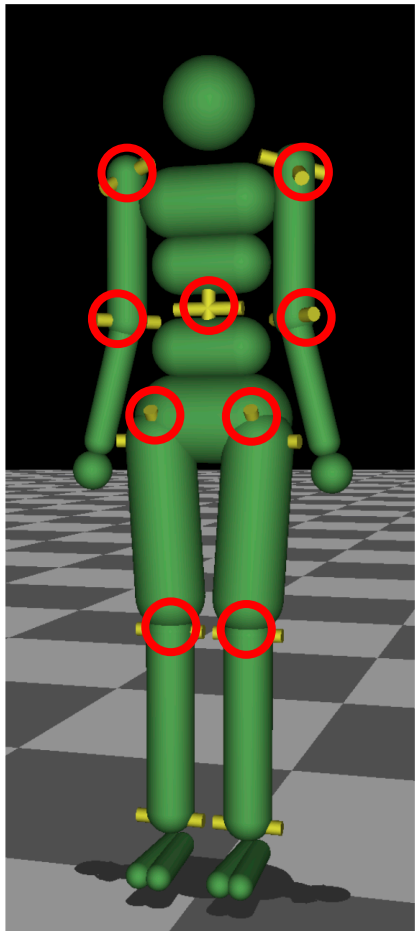
이제 코드로 살펴봅시다!

PPO 코드랩 - Roboschool

Roboschool in OpenAI gym

- 라이센스 제약이 있는 Mujoco의 대안 : 오픈소스!!!
- 액션이 Continuous한 다양한 에이전트 학습 환경 제공
- OpenAI Gym과 같은 인터페이스

Humanoid in Roboschool

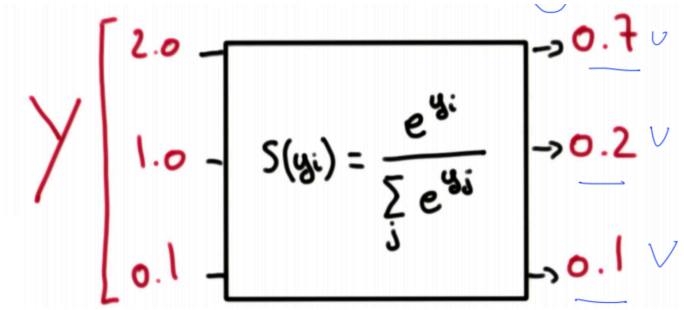


Action Space : 모두 Continuous!!

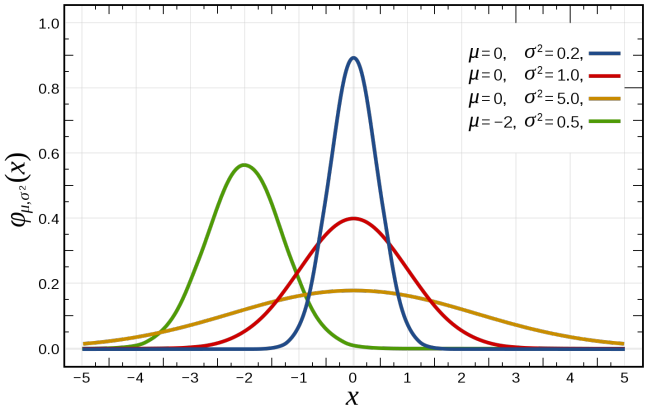
- 복부 : (x, y, z) 좌표 3개
- 어깨 축 : 오른쪽 2개, 왼쪽 2개
- 팔꿈치 : 오른쪽 1개, 왼쪽 1개
- 골반 : 오른쪽 (x, y, z) 3개, 왼쪽 (x, y, z) 3개
- 무릎 : 왼쪽 1개, 오른쪽 1개

총 : 17개

Humanoid in Roboschool



Softmax function



Gaussian Distribution

Continuous Action은 어떻게 하지?

- **Discrete** Action:

  Softmax function 으로 각 행동을 할 확률을 출력

- **Continuous** Action :

  Action Space 개수만큼 정규분포를 만들고

  Policy Network가 정규분포의 평균과 표준편차를

  최적화(Gaussian Policy)

감사합니다.

urleee@naver.com
zzing0907@gmail.com