Reinforce Learning

Policy Gradient

소프트웨어 꼰대 강의

노기섭 교수

(kafa46@hongik.ac.kr)

A Taxonomy of RL Algorithms

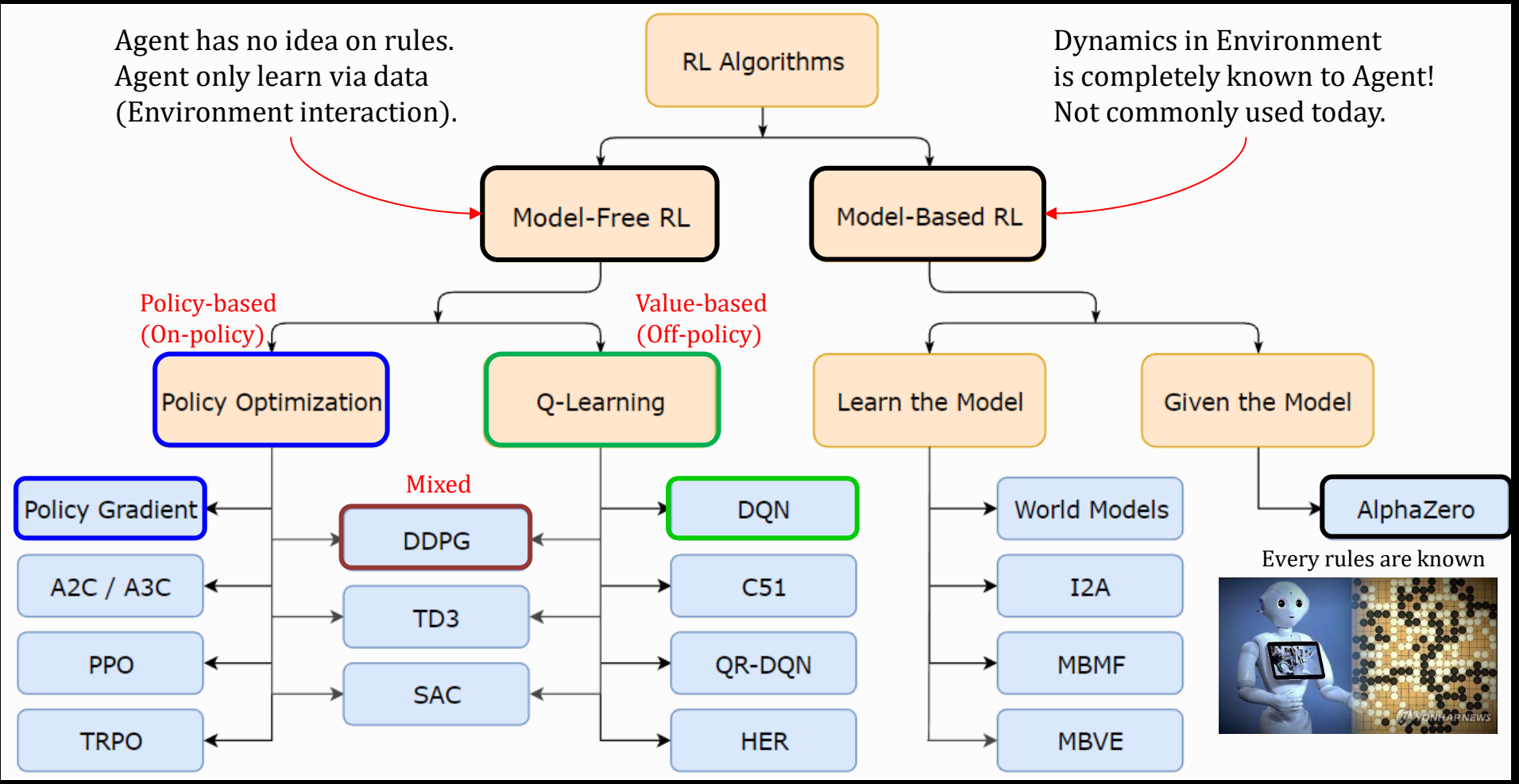
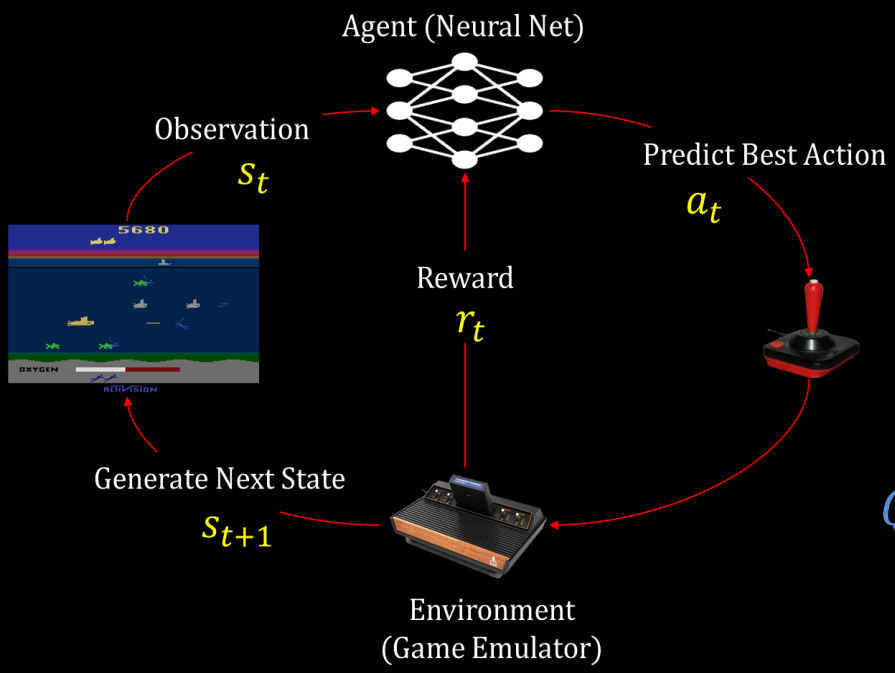


Image source: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

Recap: Q-learning



Goal (idea):
"Agent (neural net) wants to find a Q-function that satisfies the Bellman Equation"

$$Q^*(s,a) = \mathbb{E}_{s' \sim \epsilon}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]$$

How to learn?

Target (Bellman Eq.)

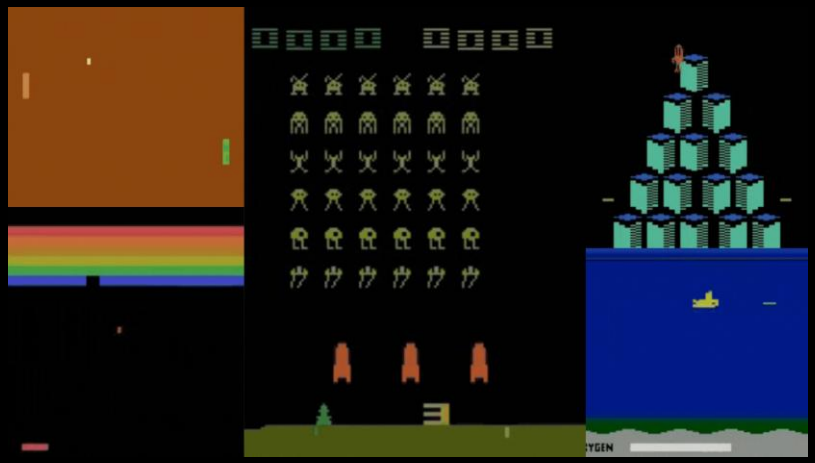
Prediction

$$Q(s_t,a) \leftarrow Q(s_t,a) + \alpha\left[\boxed{r_{t+1} + \gamma \max_{p} Q(s_{t+1},p)} - \boxed{Q(s_t,a)}\right]$$

Ground Truth

Motivation for Policy Gradient

Q-learning (value-based)



Too much data!
In efficient & Difficult!

Hard to apply onto continuous data!

Policy-gradient (policy-based)

When a basketball player shoots,
they don't calculate the probability of scoring.

They simply execute the motion their body remembers.

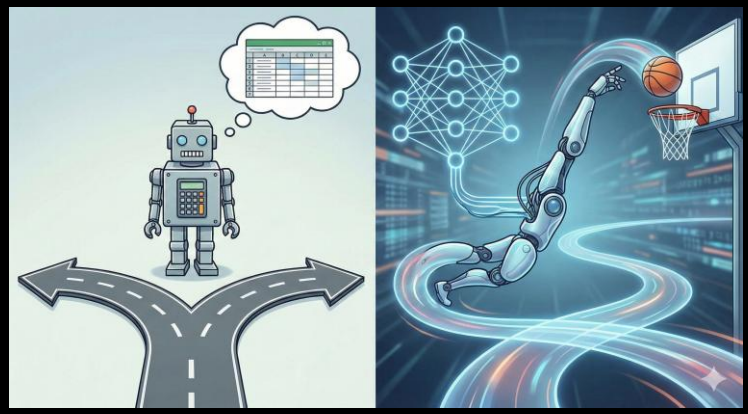


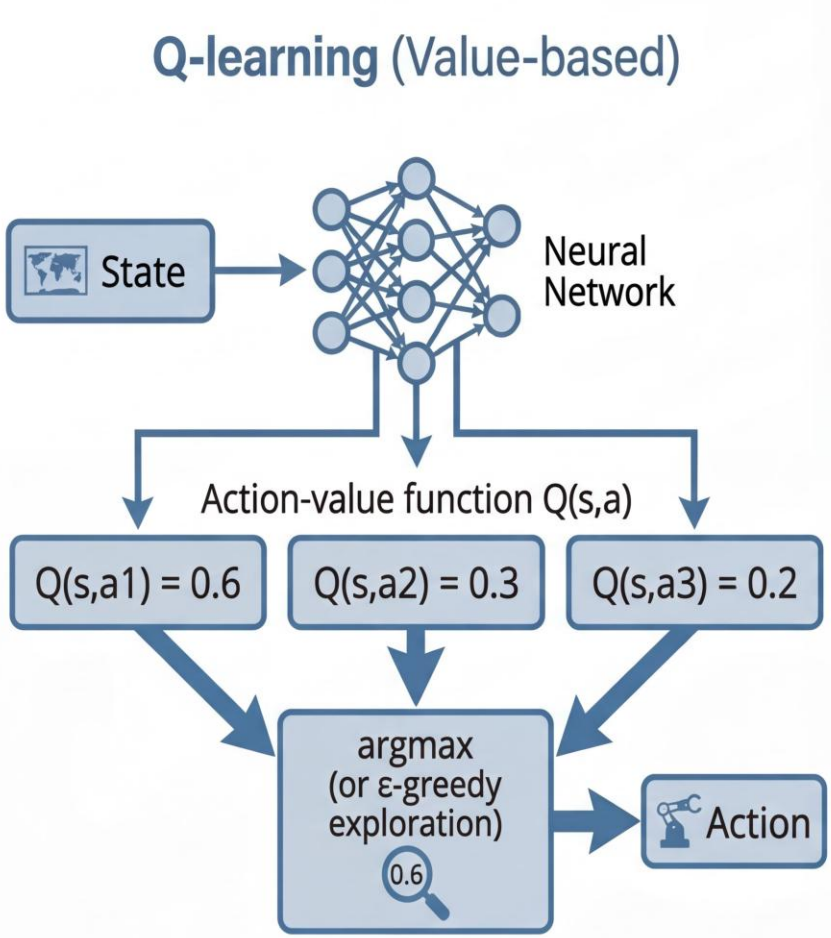
Image source: https://luca-works.com/2026/01/20/policy-gradient-reinforce-algorithm/

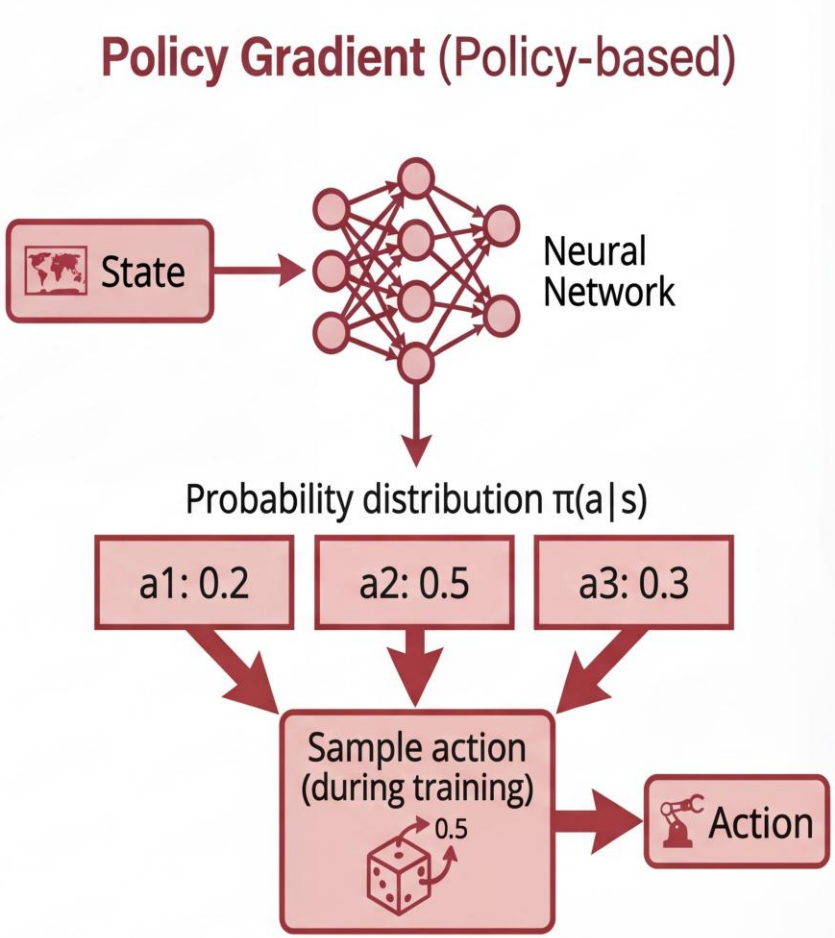
Just learn policy directly!
Sometimes more efficient for simple policy.

Q-learning vs. Policy Gradient

Category	Q-learning (Value-based, Off-policy)	Policy Gradient (Policy-based)
Core Idea	Learn a function approximator for Q(s,a) using the Bellman equation	Learn the policy directly by optimizing action probabilities
Objective	Find a Q-function that satisfies the Bellman equation	Optimize policy parameters to maximize expected return
Bellman Equation	Required	Not required explicitly
Action Space	Difficult to apply when actions are continuous	Naturally handles continuous and multiple actions
Data Requirement	Requires a large amount of (s,a) data	Can be more efficient for simple policies
Reward Structure	Hard to train without intermediate rewards (e.g., robot arm tasks)	Can learn even with sparse rewards
Learning Target	Value function (Q-values)	Policy directly
Efficiency Issues	Inefficient and difficult due to large state-action space	Sometimes more efficient for simple decision rules
Exploration	Implicit via max operator and exploration strategy (e.g., ε-greedy)	Actions sampled from policy distribution
Planning Capability	Limited direct planning	Can support planning and reasoning ahead
Example Intuition	Estimate "how good" each action is	Learn "what action to take" directly

Conceptual Understanding





Policy Gradient - Policy Optimization

Define a set of parametrized policies.

$$\Pi = \{\pi_\theta | \theta \in \mathbb{R}^m\}$$

How?
Possible?

For each policy, define its value
(i.e., Objective Function)

$$J(\theta) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi_\theta\right]$$

Cumulative expected sum of reward

Discount factor

Gradient ascent on policy params

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta (a_t | s_t)$$

Model π : differentiable function

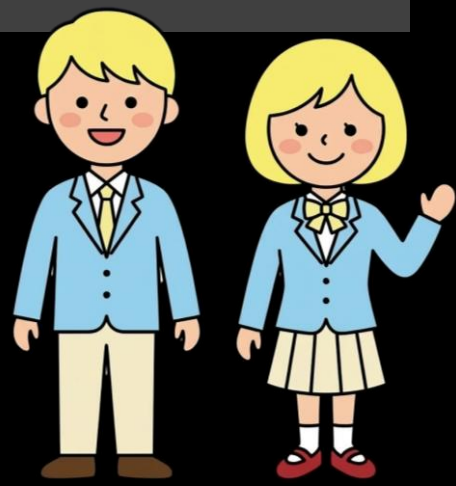
Use Neural Net to emulate π

Find the optimal policy.

$$\theta^* = \arg\max_\theta J(\theta)$$

How can we do?

Derivative



The most beautiful feature of derivative

(Recall High School Math^^)

Natural Logarithm: $\frac{d \ln x}{dx} = \frac{1}{x}, where\ x \neq 0$

Common Logarithm: $\frac{d \log_a x}{dx} = \frac{1}{x \ln a}, where\ x \neq 0, a > 0$

Chain Rule (Composite Function): $\frac{d \ln|f(x)|}{dx} = \frac{f'(x)}{f(x)} = \frac{1}{f(x)} \frac{df(x)}{dx}$

$$\frac{d \ln f(x)}{dx} = \frac{f'(x)}{f(x)} = \frac{1}{f(x)} \frac{df(x)}{dx}, \boldsymbol{where\ x > 0}$$

Differentiation on Policy Gradient

Expected Reward: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)] = \int_{\tau} r(\tau) \cdot p(\tau;\theta)\ d\tau$

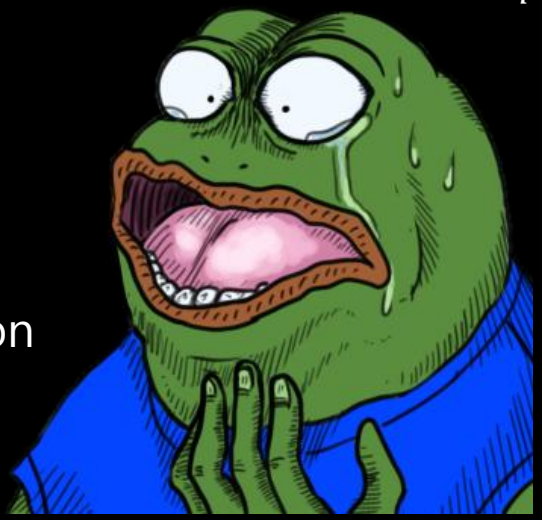
We need to differentiate this!

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau) \boxed{\nabla_\theta p(\tau;\theta)}\ d\tau$$

Intractable!!!

since the trajectory distribution $p(\tau;\theta)$ depends on θ.

확률 p는 모르는 함수



게다가 그 확률은 policy params θ와 연계됨

우리는 θ와 연결된 모르는 함수 $p(\cdot)$를 미분해야 함

어쩌라는 거얌! ㅠㅠ

Why Is It Intractable?

Why is differentiation intractable?

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\ \nabla_\theta\, p(\tau;\theta)\ d\tau$$

The distribution $p(\tau;\theta)$ involves a product over all time steps and depends on unknown environment dynamics.

Initial state distribution

A trajectory is defined as
$\tau = (s_0, a_0, s_1, a_1, \cdots)$

$$p(\tau;\theta) = \rho(s_0) \prod_t \pi_\theta\,(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$$

It represents the full sequence of states and actions over time.

- In continuous state spaces, the trajectory becomes infinite dimensional
- For length T, the number of possible trajectories grows exponentially

We must sum over all possible trajectories.

$$\int_{\tau} (\cdot)\, d\tau$$

means integrating over

all possible trajectories.

In practice, this requires enumerating an exponentially large (or infinite) space.

→ Computationally infeasible!

We can do it with a trick ^^

But we have a well-know, famous, and beautiful & useful trick

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \boxed{\nabla_\theta\, p(\tau;\theta)}\, d\tau$$

Pain Point!

$$\nabla_\theta\, p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta\, p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

We rewrite the gradient in a form that is easier to compute.

Recall the derivatives

$$\frac{d \ln x}{dx} = \frac{1}{x}, where\ x \neq 0$$

log를 취하면
"곱(product)"이 "합(sum)"으로
바뀌어서 미분이 쉬워진다

$$\frac{d \ln f(x)}{dx} = \frac{f'(x)}{f(x)} = \frac{1}{f(x)}\frac{df(x)}{dx}, where\ x > 0$$

Transform Trick: Integral → Expectation

$$\nabla_\theta\, p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta\, p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

$$\nabla_\theta J(\theta) = \int_{\tau=1}^{T} r(\tau)\,\nabla_\theta p(\tau;\theta)\, d\tau$$

$$\nabla_\theta J(\theta) = \int_{\tau=1}^{T} r(\tau) p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)\, d\tau$$

Estimate using Monte Carlo (refer to next slide)

$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

원래는 적분

$$\nabla_\theta J(\theta) = \int_{\tau} r(\tau)\ \nabla_\theta\, p(\tau;\theta)\ d\tau$$

이제는 기대값 조금만 더 조물딱 하면 구할 수 있다 ^^

Now the gradient is expressed as an expectation, but still depends on $p(\tau;\theta)$

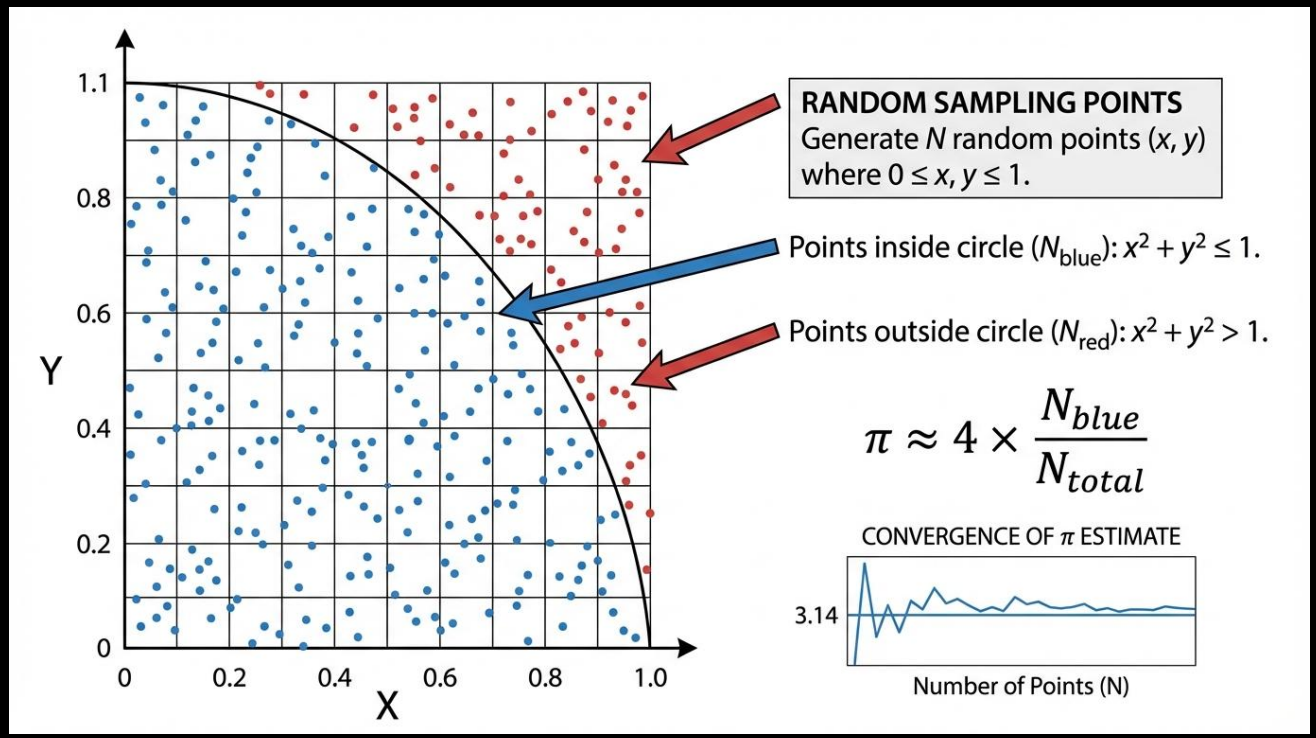
Monte Carlo Approximation (Simulation)

$$\mathbb{E}_{\tau\sim p(\tau;\theta)}[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} r(\tau^{(i)})\nabla_\theta \log p(\tau;\theta)$$

By Monte Carlo Approximation

적분 계산은 해결 됨!
하지만 여전히 $p(\tau;\theta)$는 알 수 없다.



What we know or don't know

What is our knowledge?

We don't know how to compute explicitly!

$p(\tau;\theta)$ ⇦ Unknown environment dynamics (Transition Probability)

We know

π_θ ⇦ Modeling with Neural Network

$$\nabla_\theta\, p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta\, p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$$

After replacing it with the items we know,

it can be differentiated!!!

After Step by Step, You will get a magic!

$$p(\tau;\theta) = \rho(s_0)\prod_{t\geq 0} p(s_{t+1} \mid s_t, a_t)\,\pi_\theta(a_t \mid s_t)$$

These terms do not depend on θ → gradient is zero

$$\log p(\tau;\theta) = \log \rho(s_0) + \sum_{t\geq 0}[\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

원래는 풀 수 없는 수식

$$\nabla_\theta J(\theta) = \int_{\tau=1}^{T} r(\tau)\,\nabla_\theta p(\tau;\theta)\,d\tau$$

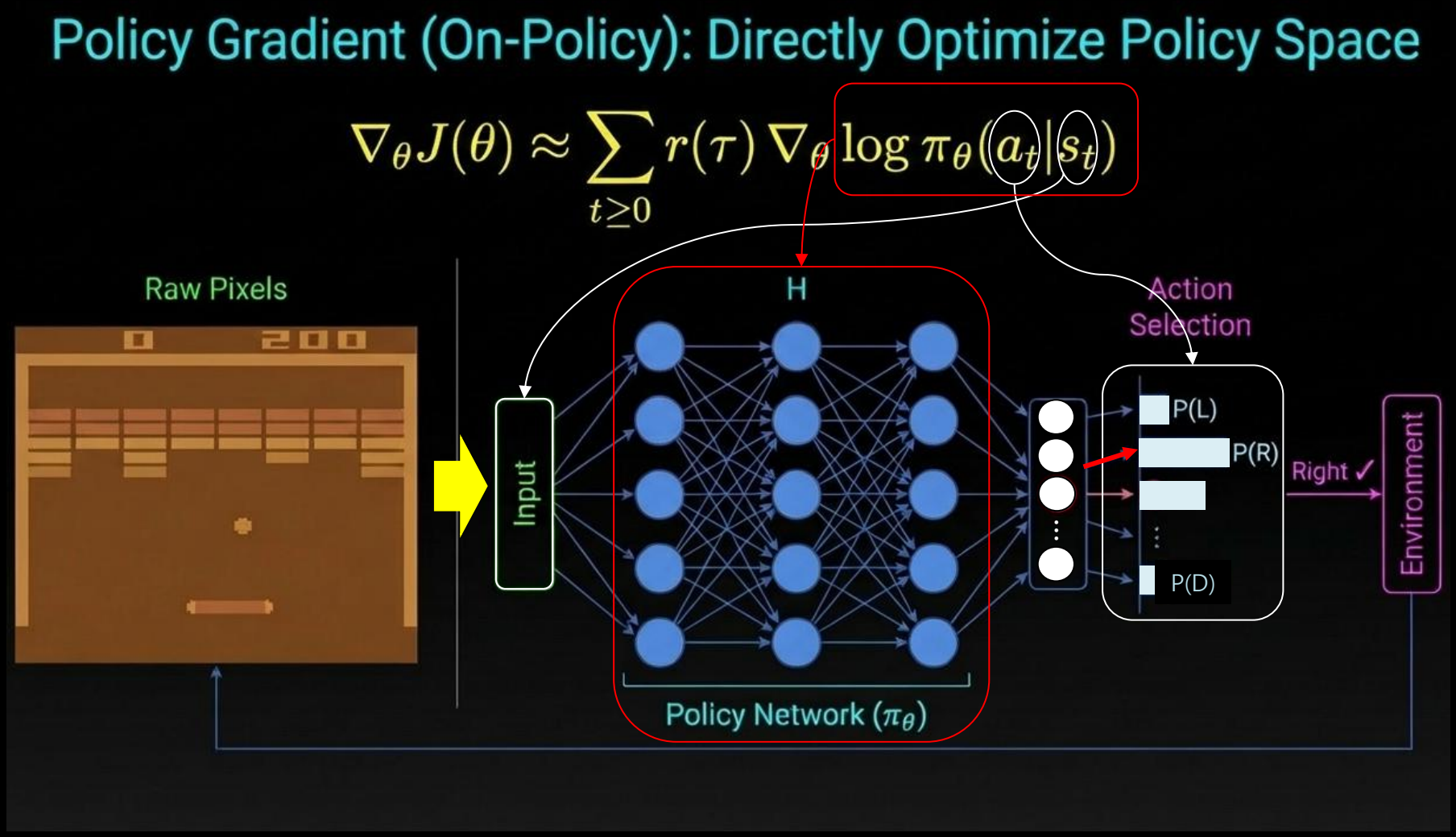
이제는 풀 수 있음!

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau^{(i)}) \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t\geq 0}^{T} r(\tau^{(i)})\,\nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

Model π is differentiable function, since Neural Net is differentiable!

Applying Neural Net



Quick Overview on Policy Optimization Algorithms

A Taxonomy of RL Algorithms

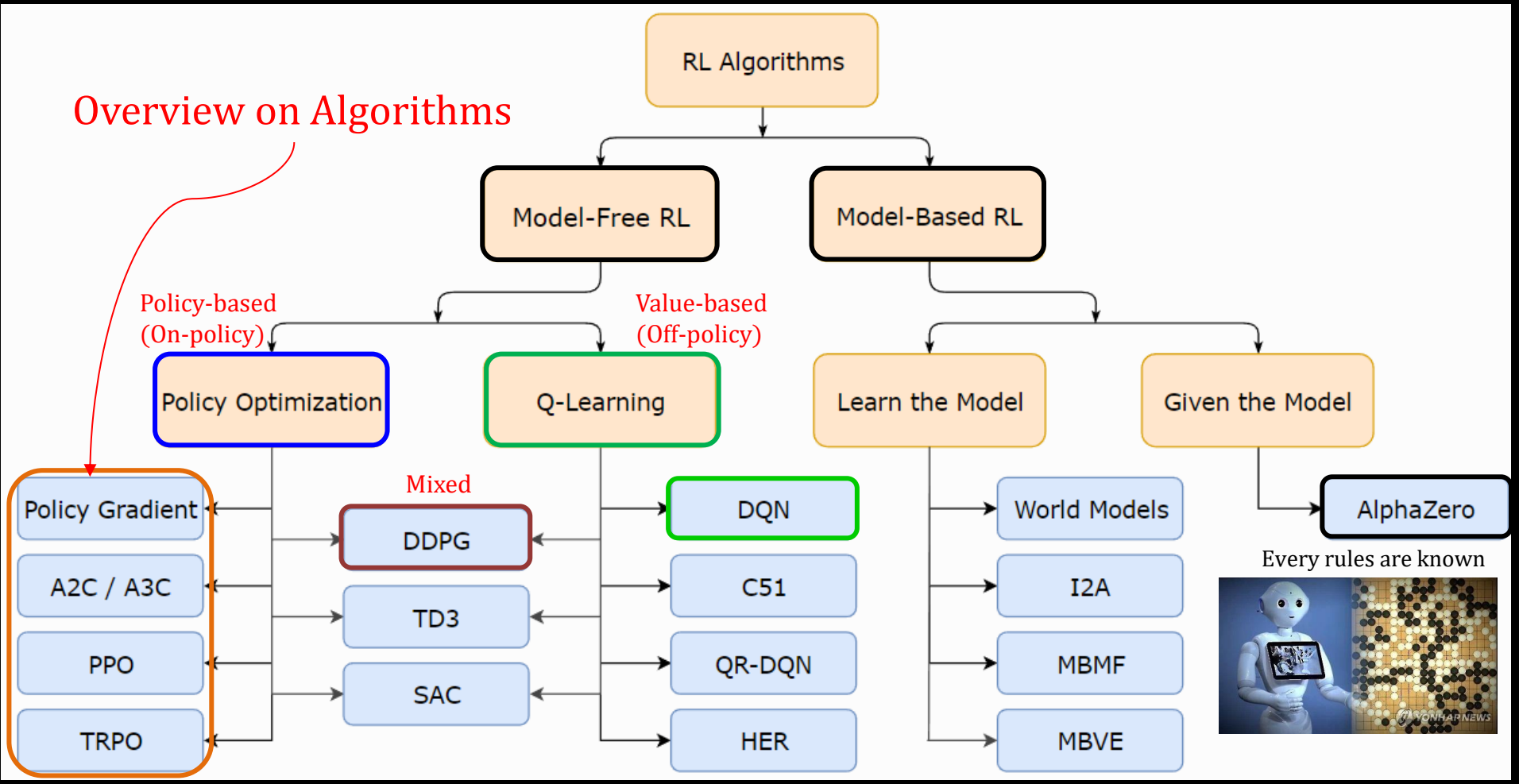


Image source: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below

Policy Gradient (REINFORCE)

Firstly, introduced by Ronald J. Williams

"Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," (1992)

REINFORCE:
REward **I**ncrement =
Nonnegative **F**actor times **O**ffset **R**einforcement times **C**haracteristic **E**ligiblity

Paper link: https://people.cs.umass.edu/~barto/courses/cs687/williams92simple.pdf

(1945~2024)

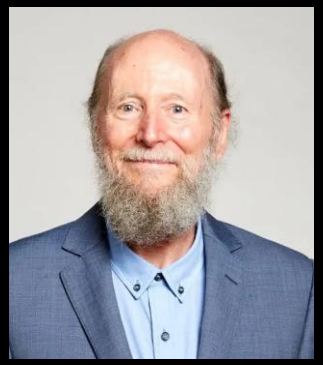
https://en.wikipedia.org/wiki/Ronald_J._Williams

Policy Gradient Theorem is proofed by Sutton et al.

"Policy Gradient Methods for Reinforcement Learning with Function Approximation," (1999)

- ✓ Established the Policy Gradient Theorem!
- ✓ Provided the theoretical foundation for Actor–Critic methods
- ✓ Extended policy gradient methods to continuing tasks and function approximation

Paper link:
https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf



(1957 ~ Now)

https://en.wikipedia.org/wiki/Richard_S._Sutton

Policy Gradient (REINFORCE)

REINFORCE algorithm is know as "Vanilla Policy Gradient".

Actually, nothing but Gradient Ascent!

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$

Algorithm parameter: step size $\alpha > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):

Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$

Loop for each step of the episode $t = 0, 1, \ldots, T-1$:

$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$ ← Gradient Ascent

Policy gradient increases the probability of high-return actions while decreasing the probability of low-return actions.

Source: Sutton & Barto, "Reinforcement Learning: An Introduction 2nd Edition," Chap. 13 "Policy Gradient Methods", pp. 326-329, The MIT Press, 2020.
Download from: http://incompleteideas.net/book/RLbook2020.pdf

Pros & Cons in REINFORCE

Pros	Cons
✓ Handles complex environments where Q-learning struggles	✓ Lower stability: Training can be less stable.
✓ Often converges faster	✓ Poor credit assignment: Delayed rewards make it difficult to attribute outcomes to specific state–action pairs. (See next slide)
✓ Naturally supports stochastic policies	
✓ Well-suited for continuous action spaces	

Problem w/ REINFORCE

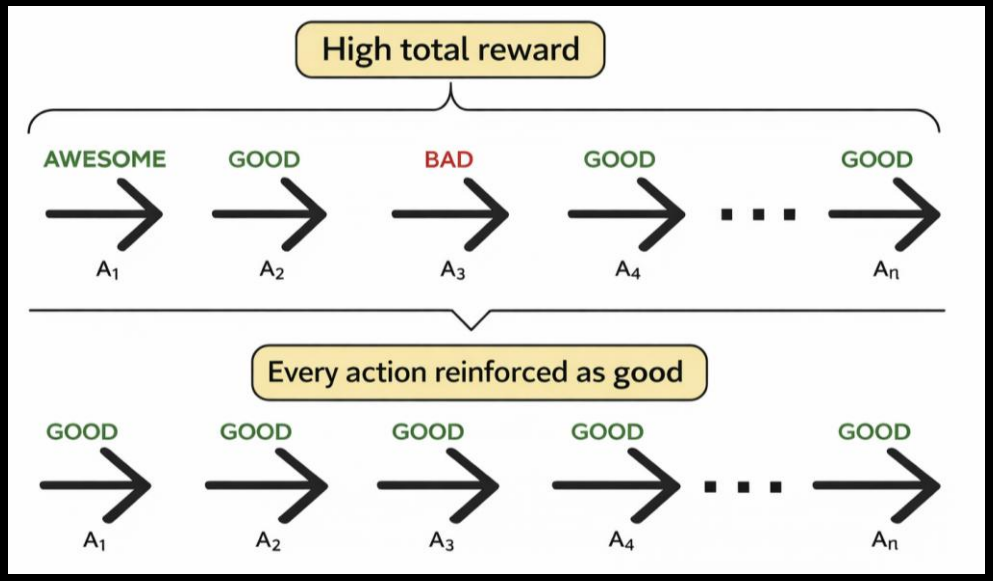
Delayed Reward Averaging!

REINFORCE updates the policy using the total return at the end of an episode.

If the final outcome is good, all actions in the trajectory receive positive reinforcement — even those that were actually poor decisions.

Bad intermediate actions may be incorrectly strengthened because the algorithm cannot distinguish which specific actions truly contributed to the final reward.

"Credit Assignment Problem"



Advantage Actor-Critic (A2C)

Policy Optimization - A2C

Advantage Actor-Critic (A2C)

Combining DQN (value-based) & REINFORCE (policy-based)

Using 2 Networks

- Actor Network
  - ✓ Policy-based
  - ✓ Predict Action from Policy
- Critic Network
  - ✓ Evaluate Action from Actor

$$\nabla\theta = \alpha \cdot \nabla\theta \log \pi(S_t, A_t) \cdot R(t)$$

$$\nabla\theta = \alpha \cdot \nabla\theta \log \pi(S_t, A_t) \cdot A(s_t, a_t)$$

$$, where\ A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

Value of action in the state

Value of the state

Role of baseline to reduce gradient variance.

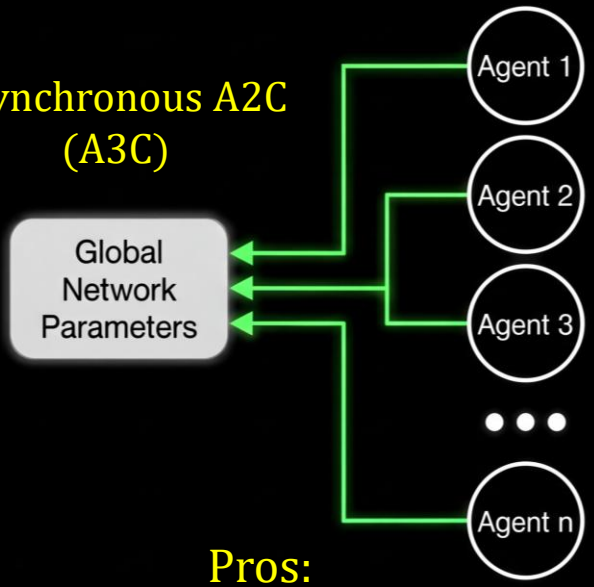
Advantage:
How much better a specific action is compared to the average action at that state.

A2C Network Structure

DeepMind

Training in parallel

Asynchronous A2C
(A3C)

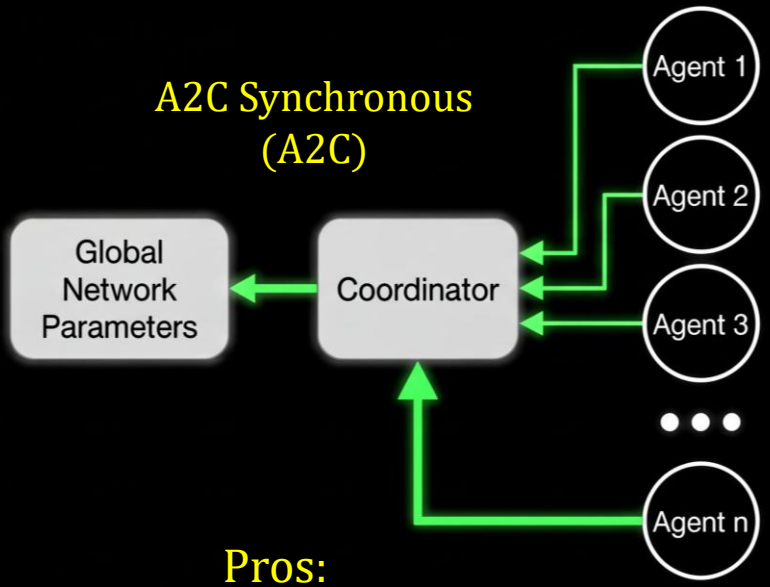


Pros:
Speed ↑
Diversity of Exploration ↑
Cons:
Gradient Collision ↑
Reproduciblilty ↓
GPU Efficiency ↓

OpenAI

Training in parallel

A2C Synchronous
(A2C)



Pros:
Stable Training
GPU Efficiency ↑
Cons:
Still unstable policy update
Large Step → Policy Collapse

TRPO & PPO

Trust Region Policy Optimization (TRPO, Berkeley, 2015)

Intuition: When updating a policy, (Collapse if policy changes significantly)
Restricting from getting too far from previous policies

Paper link: https://arxiv.org/abs/1502.05477

$$\max_{\theta} \mathbb{E}[r_t(\theta)A_t]$$

subject to $D_{KL}(\pi_{old} | \pi_\theta) \leq \delta$

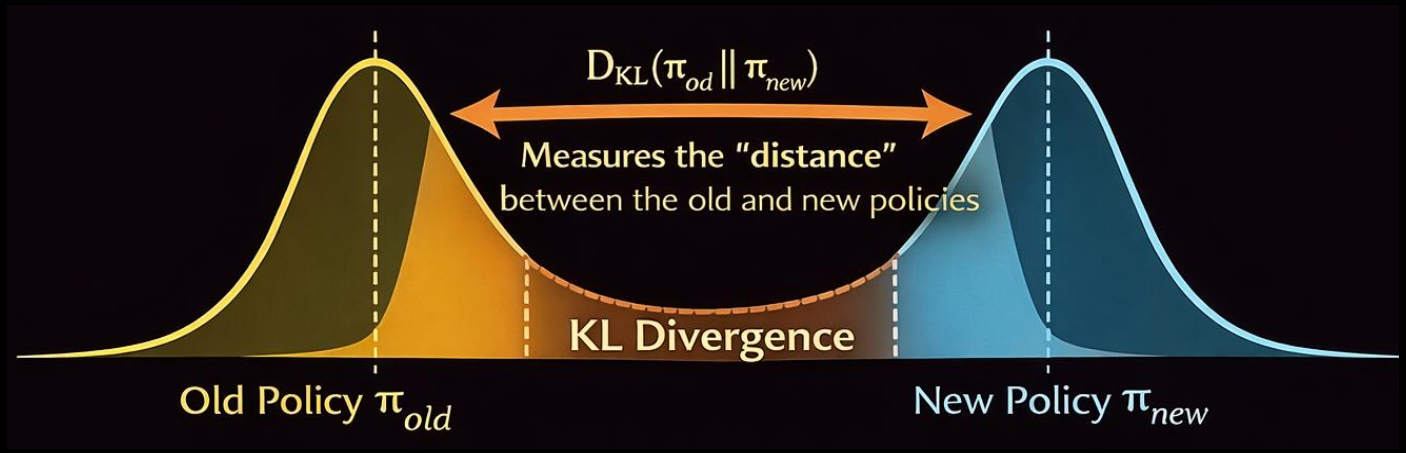
Not wisely deployed,
Due to the heavy computation &
Implementation complexity

Strengths	Guarantee monotonic policy improvement
	Reduces risk of performance collapse
	Strong theoretical foundation
Weaknesses	Computationally expensive
	Complex implementation (second derivatives, conjugate gradient)
	Not GPU-friendly compared to PPO

KL divergence

KL Divergence: Mesuring Distance Between Two Policies

$$D_{KL}(\pi_{old} \mid \pi_{new}) = \sum_{a} \pi_{old}(a) \log \frac{\pi_{old}(a)}{\pi_{new}(a)}$$



Why?
Quantifies how different the new policy is from the old policy

So that:
New policy doesn't devate too much
(ensure stable & safe update)

Proximal Policy Optimization (PPO, OpenAI, 2017)

Intuition: Simplifies TRPO idea by clipping the policy ratio to ensure stable learning

Paper link: https://arxiv.org/abs/1707.06347

$$r_t(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{old}(a \mid s)}$$

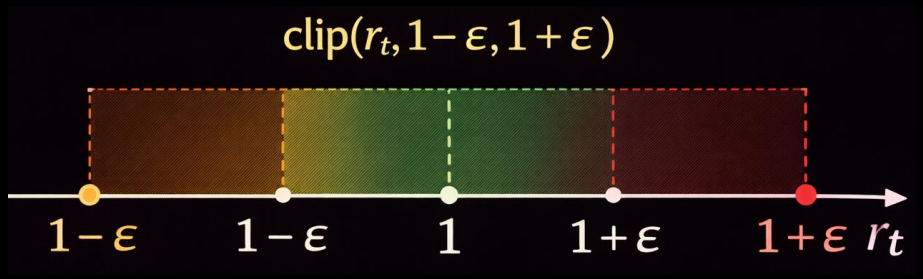
$$L^{CLIP} = \min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)\, A_t)$$

Advantage function

Strengths	Easy to implement
	GPU-friendly and scalable
	Widely used in practice
Weaknesses	No strict theoretical guarantee (no monotonic improvement proof)
	Sensitive to hyperparameters (e.g., clip range)
	Lower sample efficiency (on-policy)

More Details in Clip Function

$$L^{CLIP} = \min(r_t A_t, \mathbf{clip}(\boldsymbol{r_t}, \mathbf{1-\epsilon}, \mathbf{1+\epsilon})\, A_t)$$

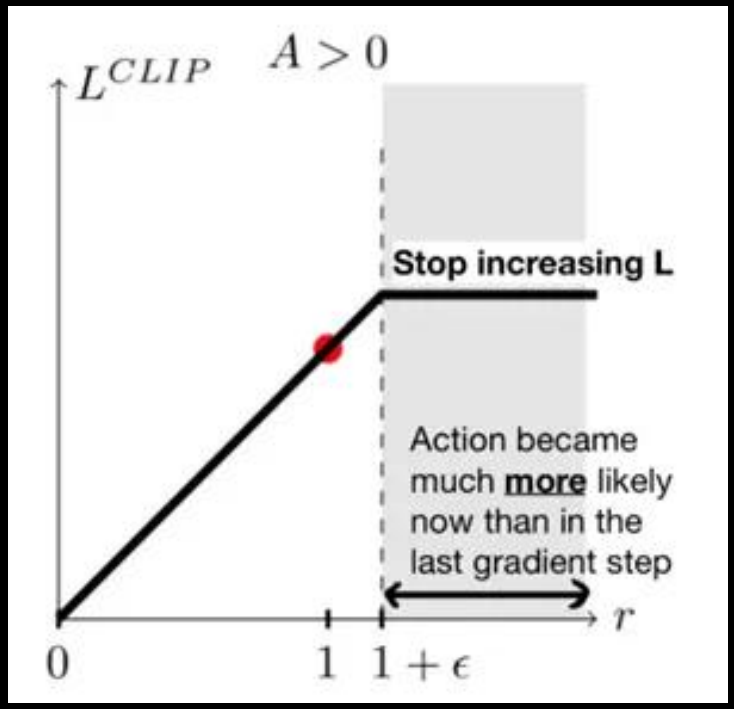


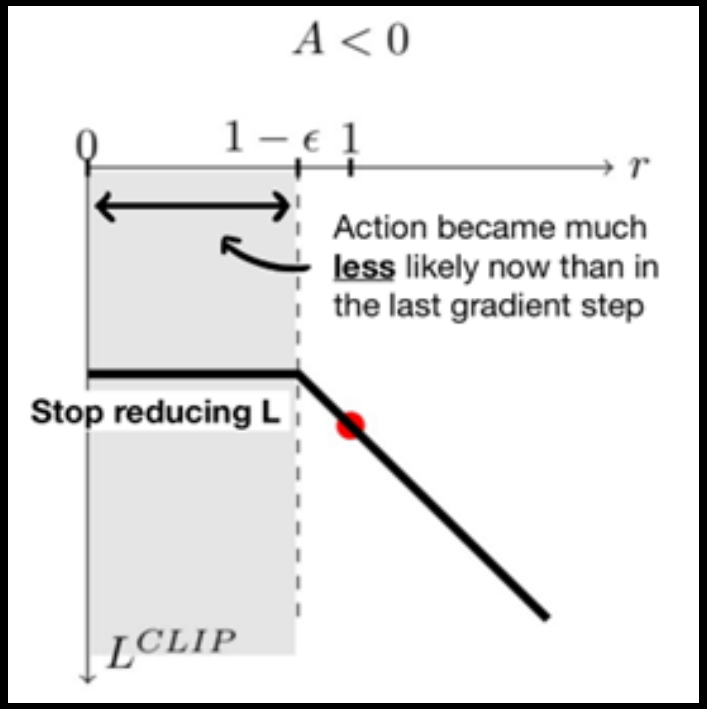
Clipping Effect

Case 1: $r_t > 1+\epsilon$
$\Rightarrow 1+\epsilon$ (clipped!)

Case 2: $1-\epsilon < r_t < 1+\epsilon$
$\Rightarrow r_t$ (unchanged!)

Case 3: $r_t < 1-\epsilon$
$\Rightarrow 1-\epsilon$ (clipped!)





수고하셨습니다 ..^^..