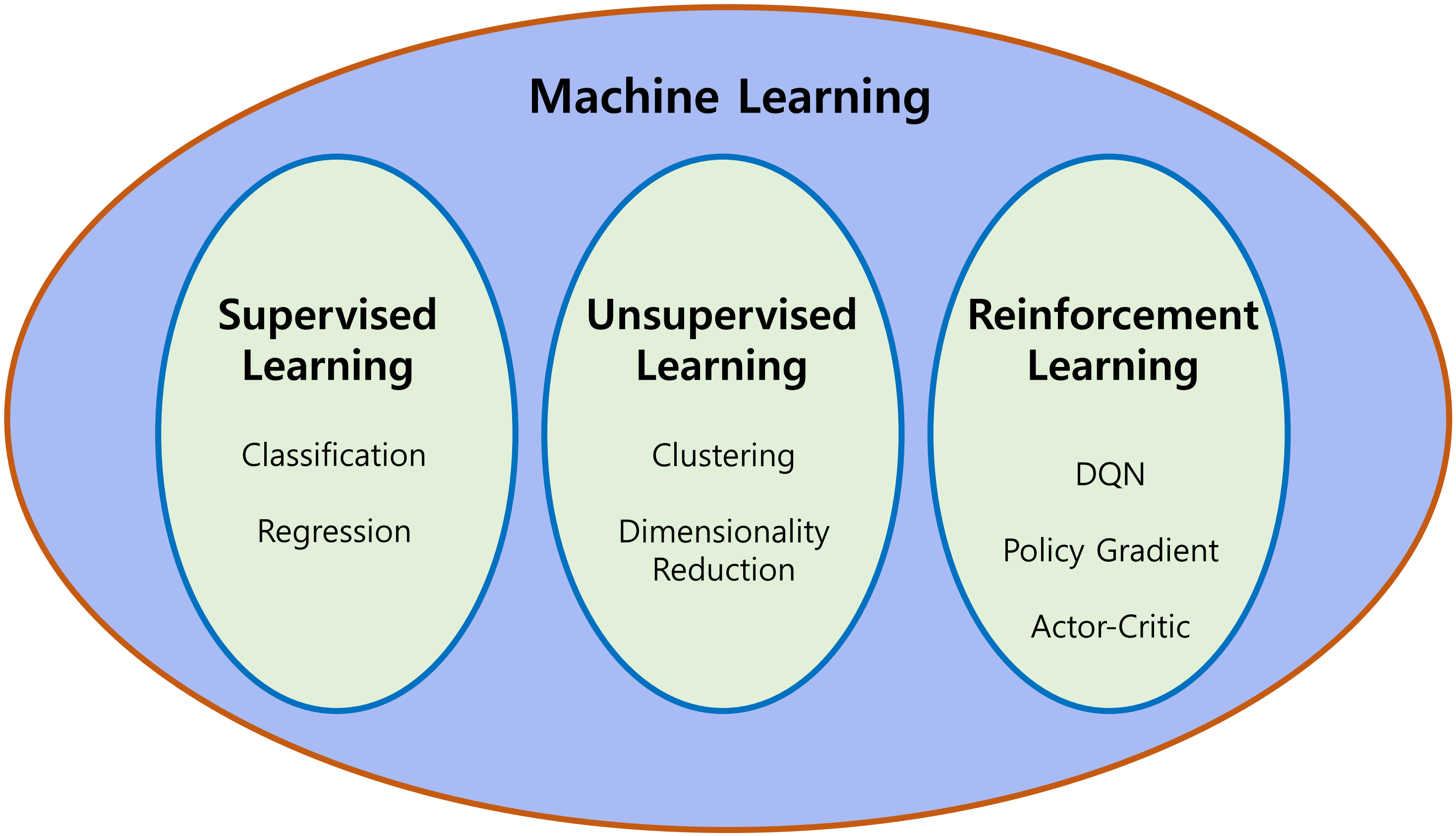
강화학습

DMQA

머신러닝의 분류

❖ 문제 상황(Task) 혹은 목적에 따라 크게 3가지로 분류

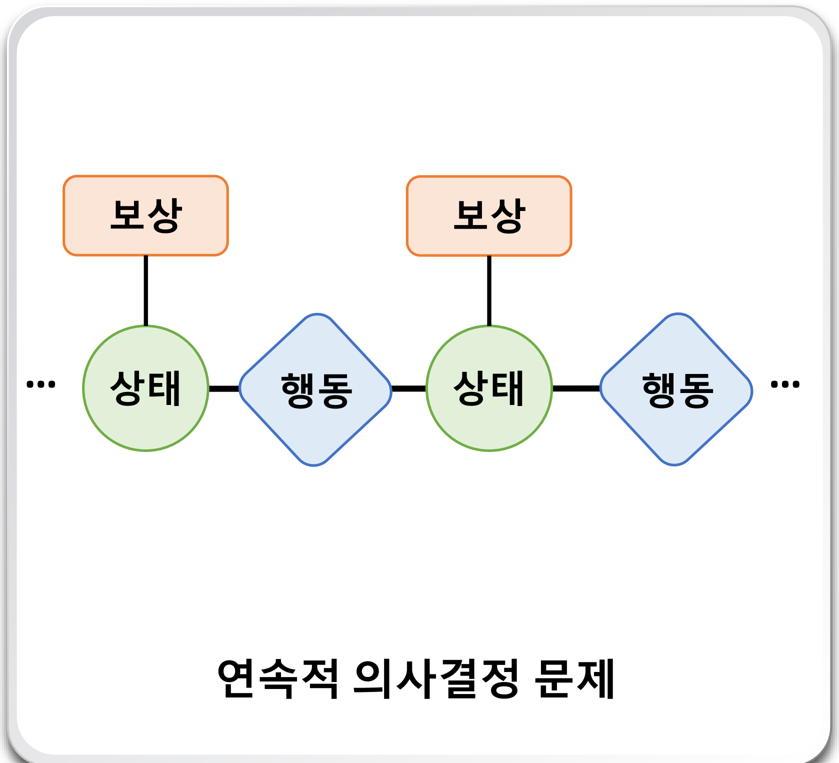


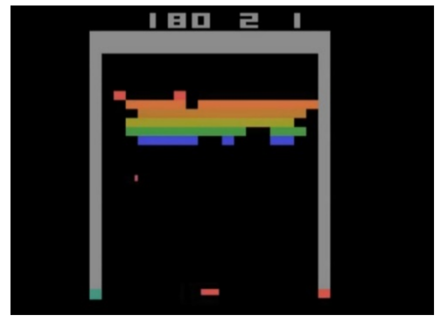
강화학습

❖ Reinforcement Learning + Deep Learning

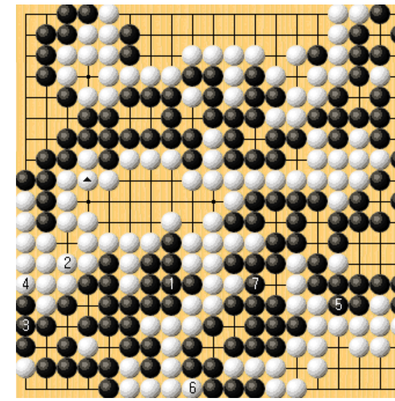
연속적인 의사결정

❖ 현실에는 다양한 종류의 연속적 의사결정 문제가 있음





비디오 게임[1]



바둑[2]

로봇 제어[3]

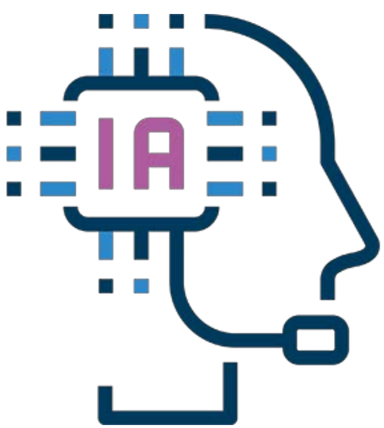
자율주행[4]

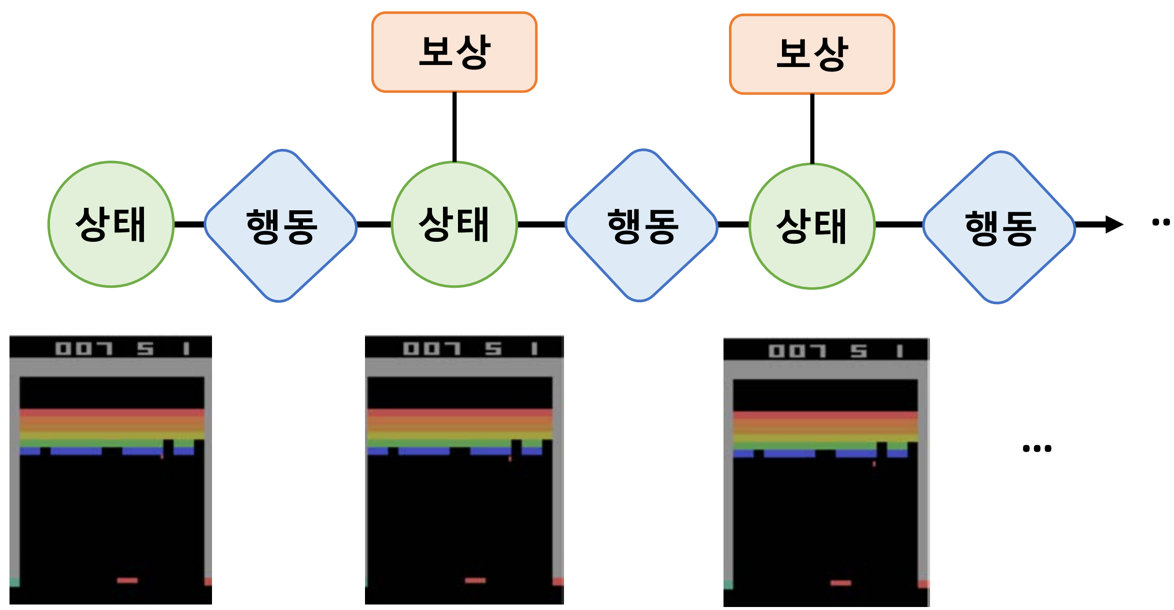
DMQA

강화학습

- 에이전트가 환경과 상호작용 하면서 연속적인 의사결정을 학습
- 누적 보상을 최대화 하는 행동을 학습하는 것이 목표
- Policy $\pi(a \mid s;\theta)$ 는 State를 바탕으로 Action을 결정하는 함수

$$\max_{\theta} E[G|\pi_\theta] \qquad where\ G = \sum_{t=0}^{T-1} \gamma^t r_t$$





강화학습 용어

❖ **S: state**

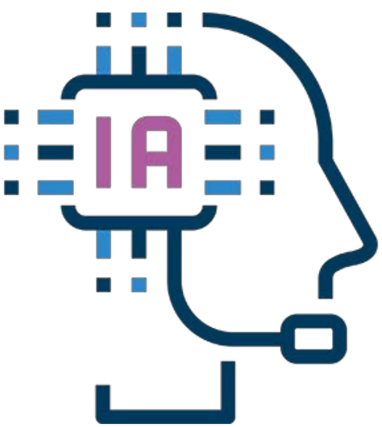
e.g. (19, 19) 바둑판

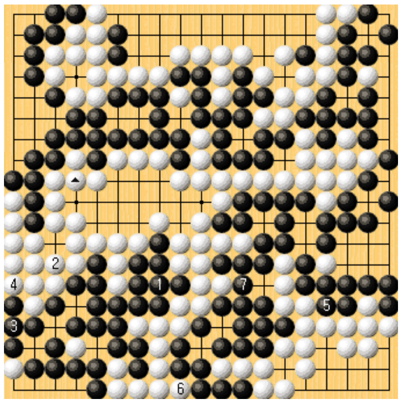
❖ **A: action**

e.g. 돌을 놓는 위치 (19, 19)

❖ **R: reward**

e.g. 승/패





DMQA

Policy

❖ In Deep RL, policy is parametrized by neural networks (θ)

❖ Deterministic policy

$$\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{S})$$

❖ Stochastic Policy

$$\boldsymbol{a} \sim \boldsymbol{\pi}(\boldsymbol{S})$$

Value Functions

❖ State-value functions

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

❖ Action-value functions

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

강화학습의 유형

❖ 학습 방식에 따른 분류

Reinforcement Learning

- **Model-Based**
- **Model-Free**

환경을 모델링

- Step 1) Environment Model 학습
- Step 2) 시뮬레이션 및 최적화를 통해 Policy 도출

실제환경

모델

시행착오 기반 학습

- 온라인 학습 방식으로 Policy 도출
- 학습하기 위해 환경과 많은 상호작용이 필요

실제환경

실제 환경에서 주행

DMQA

Model-Based & Model-Free 기법의 장단점

Reinforcement Learning

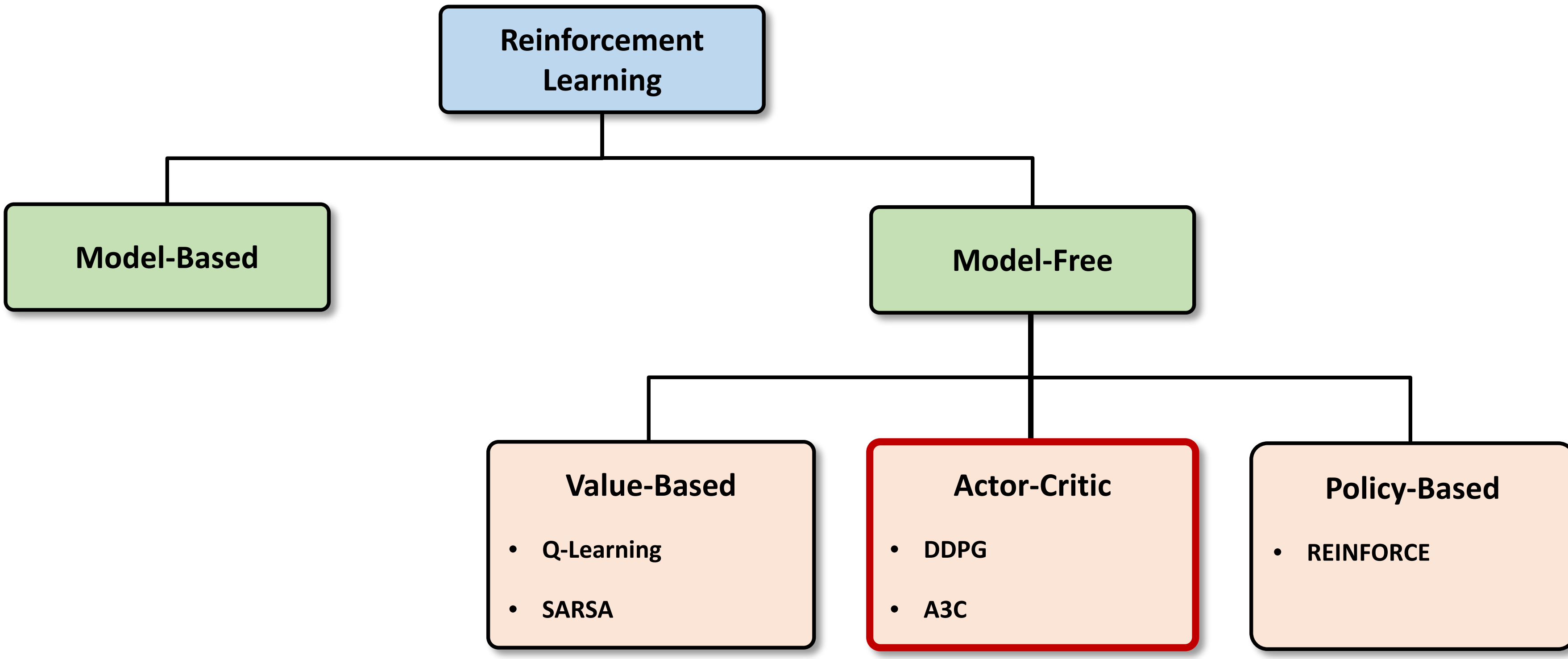
- **Model-Based**
  - 장점
    - 적은 상호작용 필요
    - 모델이 정확하면 최적 Policy 도출 가능
  - 단점
    - 모델 학습이 어려움
    - 계산 비용이 큼
    - 환경에 대한 완전 정보를 가정
- **Model-Free**
  - 장점
    - 환경에 대한 정확한 정보 없이도 학습 가능
    - 계산비용이 작음
  - 단점
    - 많은 상호작용 필요
    - 환경의 다이나믹스를 몰라 학습이 어려움

Renaudo, E., Girard, B., Chatila, R., & Khamassi, M. (2015). Respective advantages and disadvantages of model-based and model-free reinforcement learning in a robotics neuro-inspired cognitive architecture. *Procedia Computer Science*, *71*, 178-184.

Model-Free 기법의 분류

❖ 딥러닝의 발전으로 Model-Free 기법이 활발히 연구

❖ 최근 Value-Based 기법과 Policy-Based 기법을 결합한 Actor-Critic 기법이 주로 사용



DMQA

강화학습과 지도학습의 차이

❖ 공통점

- 무언가 예측한다 (action / 레이블)

❖ 차이점

- 강화학습은 환경과 상호작용

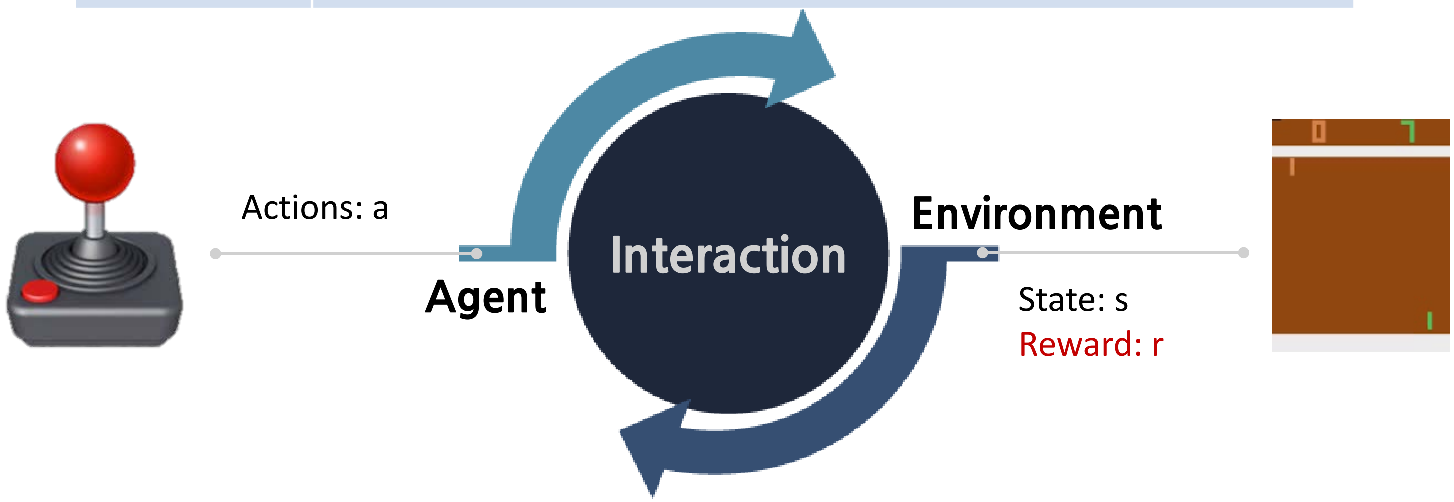
	Reinforcement Learning	Supervised Learning
Training Data	$S, A, R, S, A, \ldots$	(X, Y)
Model	$\pi(a \mid s)$	$\hat{\mathrm{Y}} = F(\mathrm{X})$
Objective	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$	$\left(\mathrm{Y} - \hat{\mathrm{Y}}\right)^2$

DMQA

강화학습의 학습 데이터

❖ 에피소드 (S, A, R 시퀀스)

Episode	Sequence
1	S, A, R, S, A, R, S, A, R, S, A, R
2	S, A, R
3	S, A, R, S, A, R
4	S, A, R, S, A, R, S, A, R
…	…



강화학습과 비지도학습의 차이

❖ 공통점

- 레이블이 없음 (No supervision)

❖ 차이점

- 강화학습은 누적 보상의 합을 최적화 (연속적인 의사결정문제)

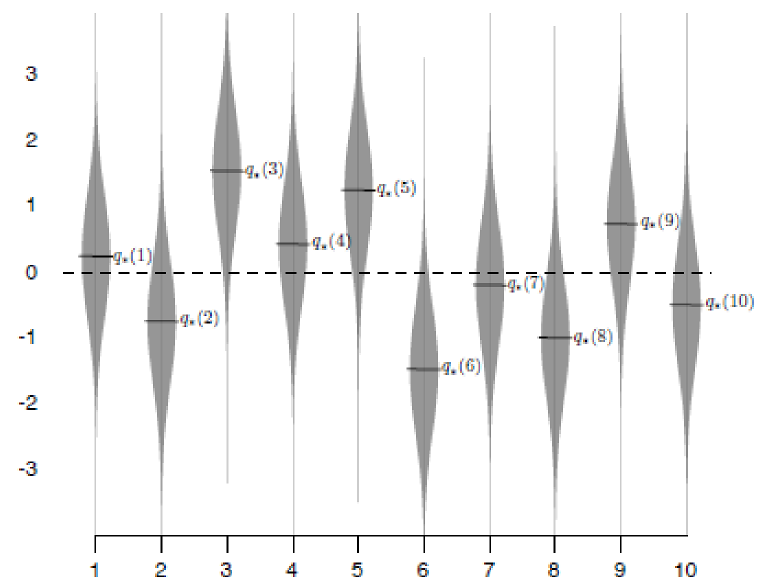
	Reinforcement Learning	Unsupervised Learning
Training Data	$S, A, R, S, A, \ldots$	X
Model	$\pi(a\|s)$	$F(\mathrm{X})$
Objective	$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$	$\|\mathrm{X} - F(X)\|$

DMQA

간단한 강화학습 문제

❖ Multi-Armed Bandits 문제

- K개의 슬롯 머신
- S: [0, 0, …, 0], 단일 state
- A: 머신 선택
- R: 획득한 돈



DMQA

문제를 푸는 방식

- ❖ **Action-Value 기법**

  Value-Based RL

- ❖ **그래디언트 기반 기법**

  Policy-Based RL

DMQA

Action-Value Function

❖ $Q_t(a)$은 행동(a) 가 얼마나 좋은지 평가

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

Bandit	Episode1	Episode2	Episode3	Episode4	Q
1				2	2
2		3			3
3	5				5
4			1		1

Action-Value Function

❖ $Q_t(a)$은 행동(a) 가 얼마나 좋은지 평가

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

Bandit	Episode1	Episode2	Episode3	Episode4	Episode5	Q
1				2	3	2.5
2		3				3
3	5					5
4			1			1

Q로 행동 도출하기

- $Q_t(a)$은 행동(a) 가 얼마나 좋은지 평가
- 평가를 잘 할 수 있다면 좋은 행동을 찾을 수 있지 않을까?

$$a_t = \underset{a}{\operatorname{argmax}}\, Q_t(a)$$

Bandit	Episode1	Episode2	Episode3	Episode4	Episode5	Q
1				2	3	2.5
2		3				3
3	5					5
4			1			1

Estimate Q function → Play (run policy)

Estimation of Action-Value

❖ Incremental update of $Q_t(a)$

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}
$$

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}.$$

NewEstimate ← OldEstimate + StepSize (Target - OldEstimate)

Exploitation vs Exploration

❖ Adding ϵ-greedy policy

Action Trajectory

Bandit	Episode1	Episode2	Episode3	Episode4	Episode5	Episode6
1					5	
2		2				
3	3					
4			1	1		

Q Trajectory

Bandit	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
1	0	0	0	0	0	5
2	0	0	2	2	2	2
3	0	3	**3**	3	3	3
4	0	0	0	1	1	1

ϵ-greedy policy

❖ Exploration을 위해 사용

❖ 더 좋은 행동을 학습 하기 위해 탐색 하는 과정

Effect of ϵ-greedy Policy

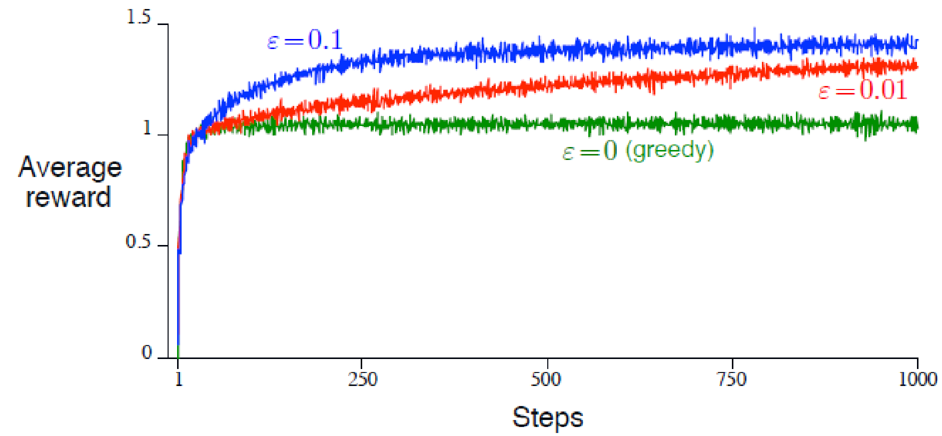
❖ ϵ-greedy policy

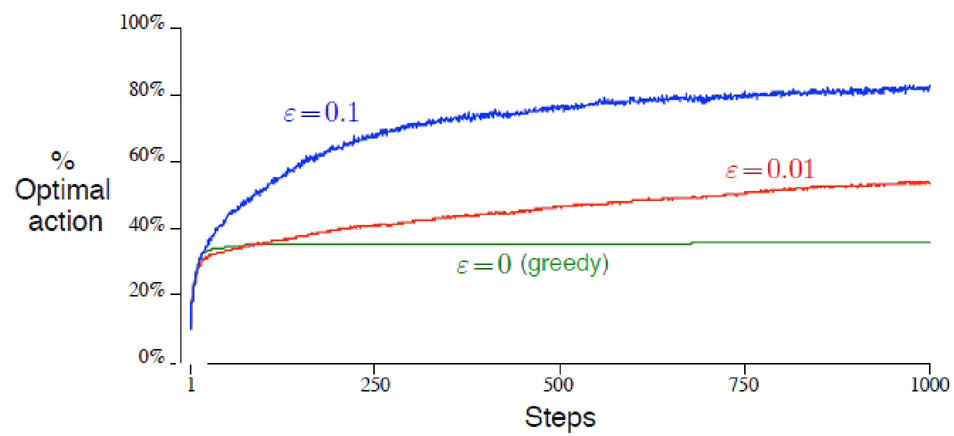
Initialize, for $a = 1$ to k:

$$Q(a) \leftarrow 0$$
$$N(a) \leftarrow 0$$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$
$$R \leftarrow bandit(A)$$
$$N(A) \leftarrow N(A) + 1$$
$$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$$



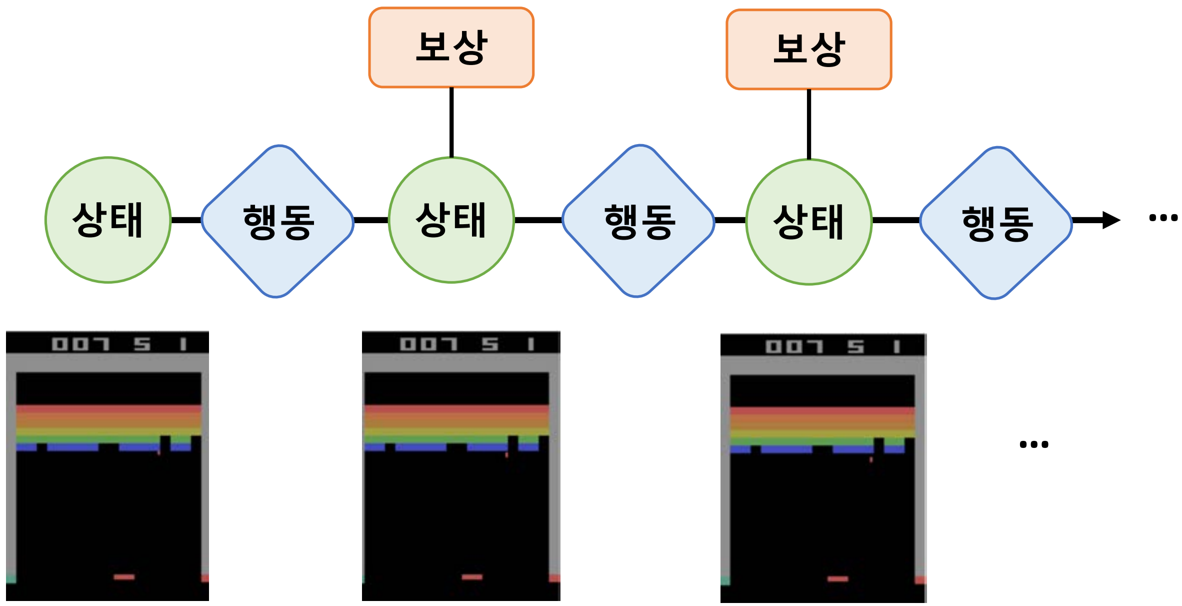


DMQA

더 복잡한 문제를 풀기 위한 확장

❖ Value function

❖ Bellman Equation



더 복잡한 문제를 풀기 위한 확장

❖ Value function

❖ Bellman Equation

Actions

States

	A1	A2	A3	A4	A5	A6	A7
S1							
S2							
S3	10	2	56	47	999	5	11
…							
S7056							

Q Table

DMQA

Value Functions

❖ State-value functions

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

❖ Action-value functions

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

DMQA

Bellman Equations

❖ Bellman Eq. for v_π

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

❖ Bellman **Optimality** Eq.

$$
\begin{aligned}
v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s', r} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big]
\end{aligned}
$$

$$
\begin{aligned}
q_*(s, a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\Big|\, S_t = s, A_t = a\Big] \\
&= \sum_{s', r} p(s', r|s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big],
\end{aligned}
$$

Dynamic Programming

❖ Model-Based Method

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

Model-Free Methods

❖ Monte Carlo (MC) Methods

- S, A, R, S, A, R, S, A, R, S, A, R, Terminate → *Update*
- S, A, R, S, A, R, Terminate → *Update*

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

❖ Temporal Difference (TD) Learning

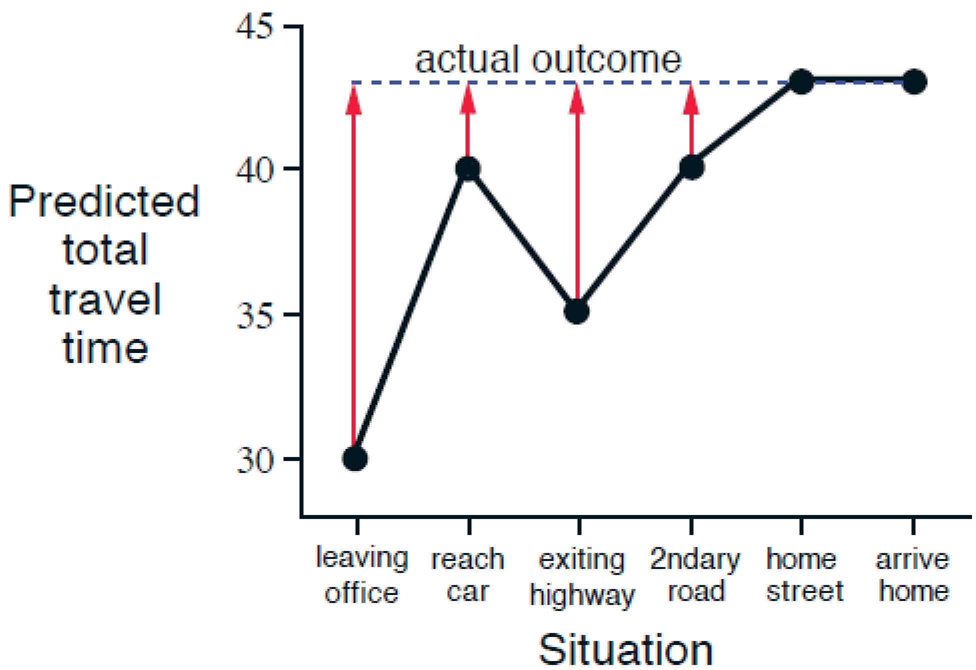
- S, A, R, S, A, R, *Update*, S, A, R, *Update*, S, A, R, Terminate, *Update*
- S, A, R, S, A, R, Terminate, *Update*

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

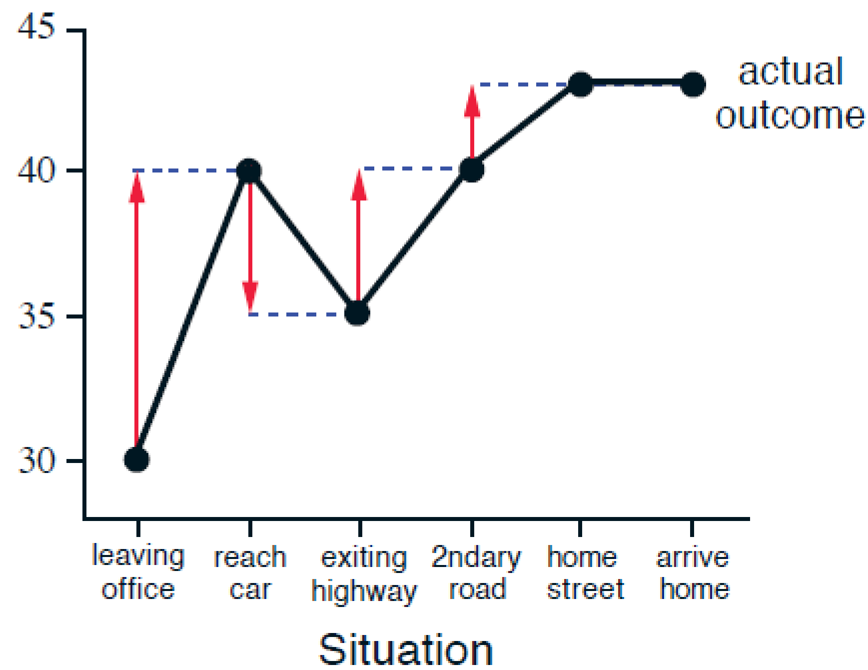
NewEstimate ← OldEstimate + StepSize (Target - OldEstimate)

MC vs TD

MC Methods



TD Methods

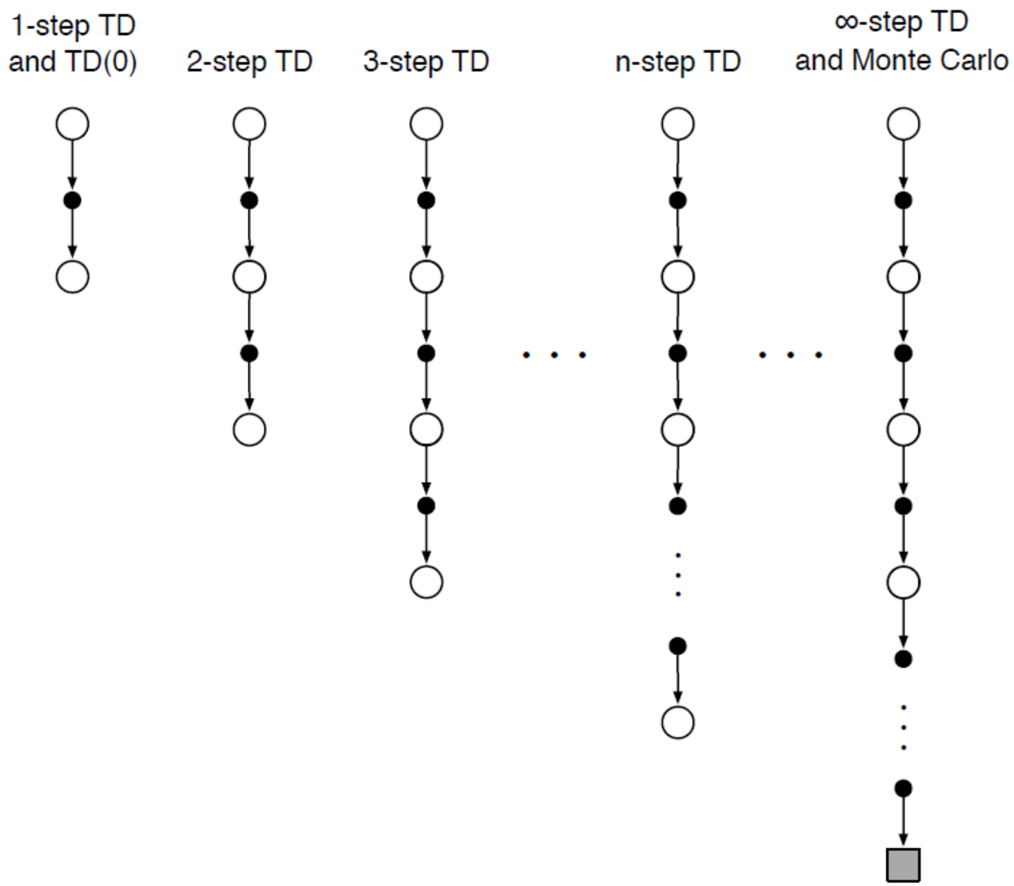


DMQA

Estimation Method: n-step TD

❖ Bias-variance tradeoff

Low variance
High bias

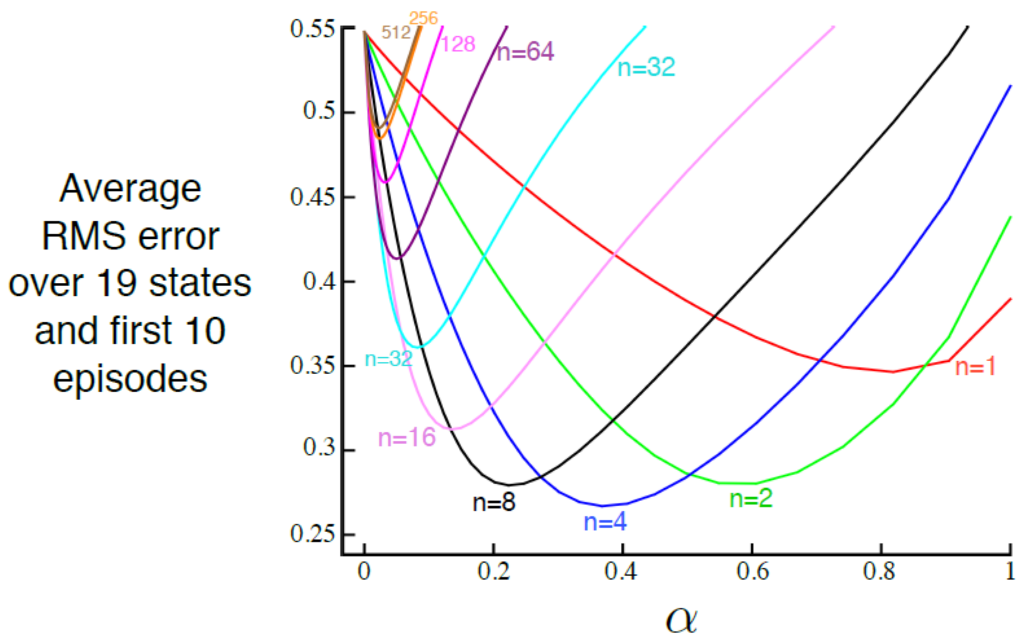


High variance
Low bias

https://www.quora.com/Intuitively-why-is-there-a-bias-variance-tradeoff-between-TD-k-0-and-TD-k-%E2%88%9E

Estimation Method: n-step TD

❖ n-step TD example



DMQA

On-Policy vs Off-policy

- To enhance exploration, use off-policy methods
  - One policy (target policy) learn (*exploitation*)
  - Another policy generate behavior (*exploration*)

- SARSA (on-policy TD)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \boxed{Q(S_{t+1}, A_{t+1})} - Q(S_t, A_t)\right]$$

- Q-learning (off-policy TD)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \boxed{\max_a Q(S_{t+1}, a)} - Q(S_t, A_t)\right]$$

SARSA

❖ 내가 선택한 행동으로 학습

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-Learning

❖ 내가 선택한 행동은 학습 데이터를 모으기 위한 과정

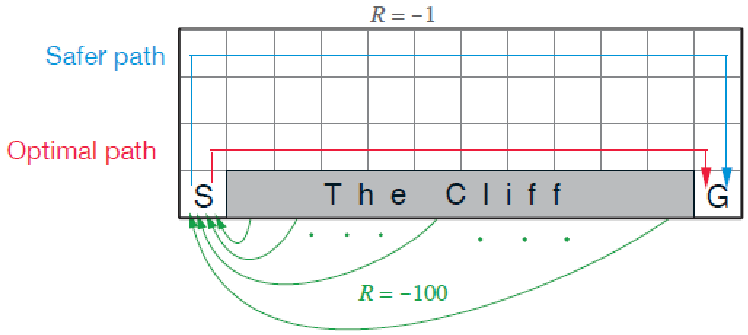
❖ 학습할 때는 다시 행동을 선택

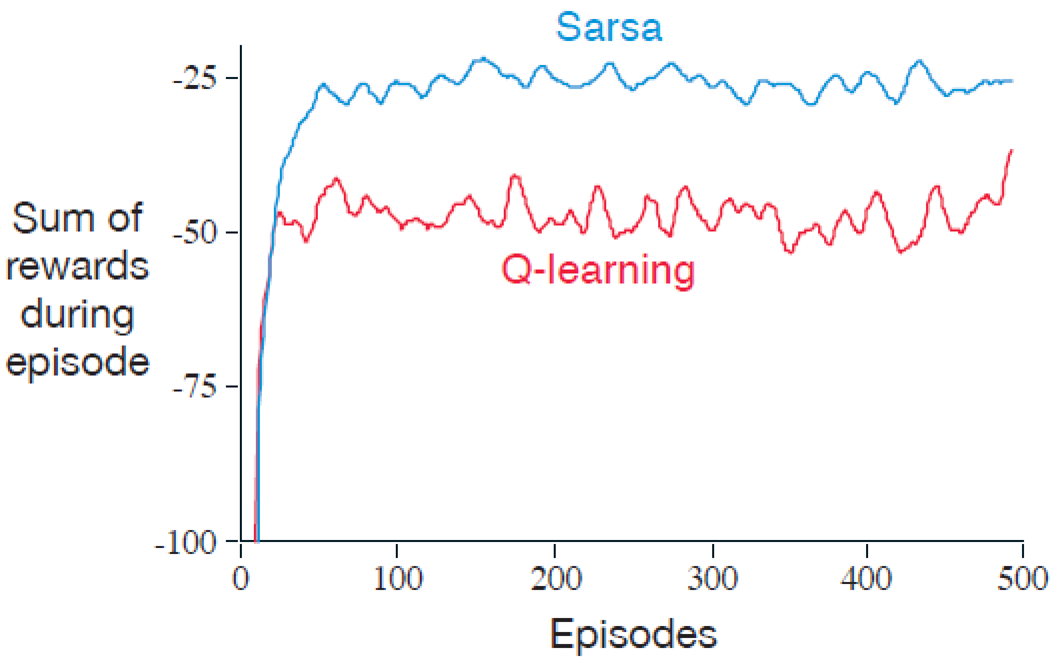
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

DMQA

On-Policy vs Off-policy

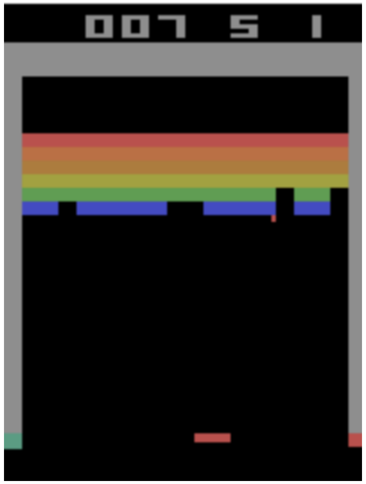


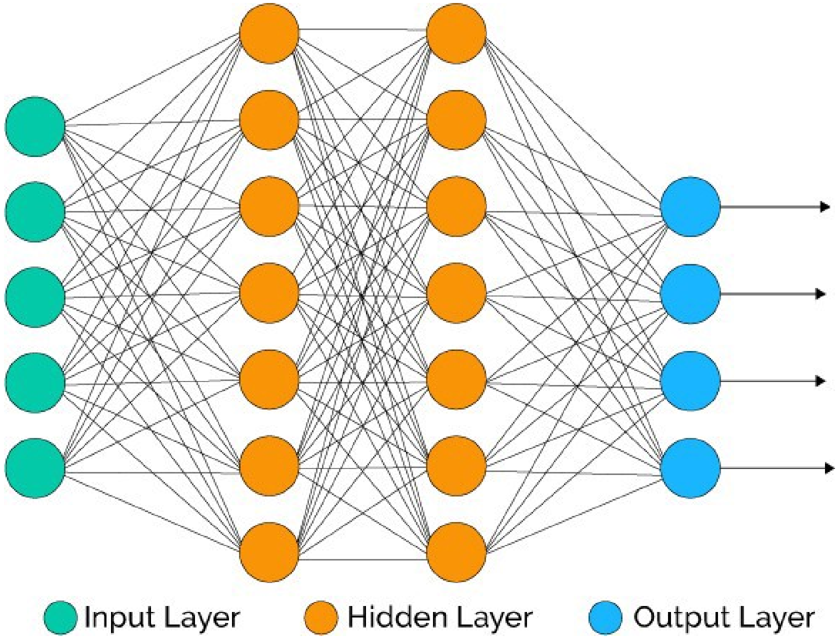


Deep Q Network (DQN)

❖ DQN = Q-Learning + Deep Neural Network

- 신경망 모델로 Q Table을 대신 (Q-learning)
- Off-policy, Value-based method
- Replay Memory, Target Network



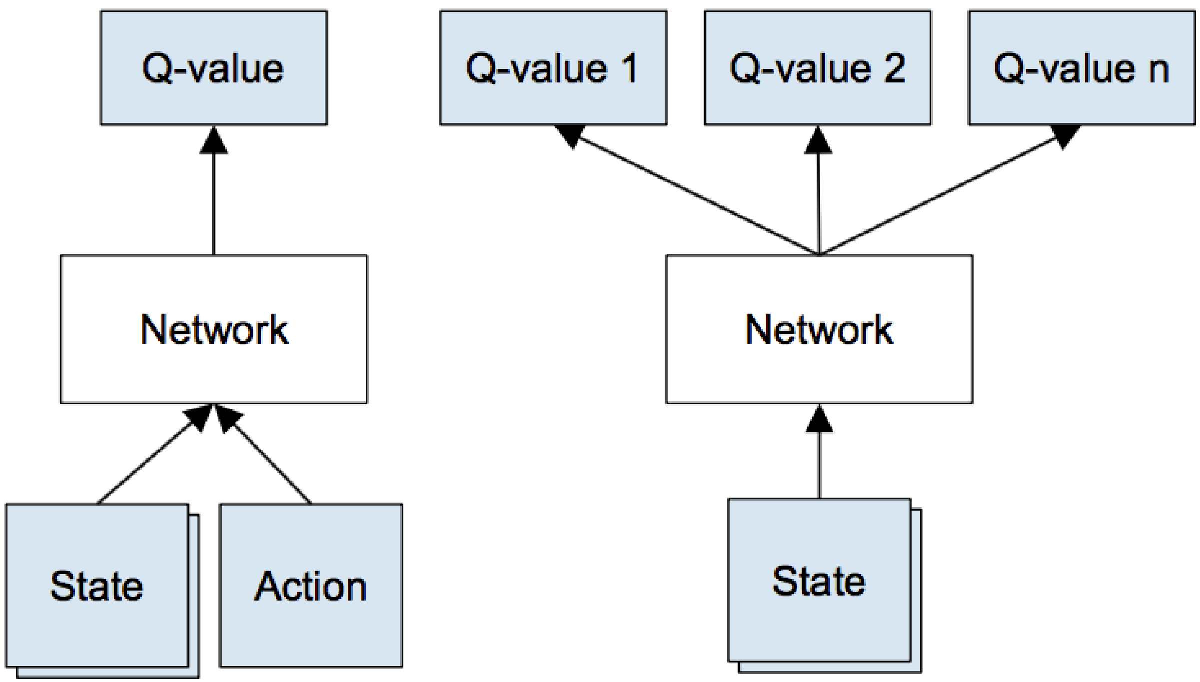


$$Q(S, A|\theta)$$

Deep Q Network (DQN)

❖ Q Network 어떻게 만들어야 하나

$$Q(s,a) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, a_0 = a]$$



DQN의 학습

❖ Q-Learning

- S, A, R, S, A, R, *Update*, S, A, R, *Update*, S, A, R, Terminate, *Update*
- S, A, R, S, A, R, Terminate, *Update*

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

❖ DQN

- Q 네트워크 학습: Temporal Difference (TD) Learning

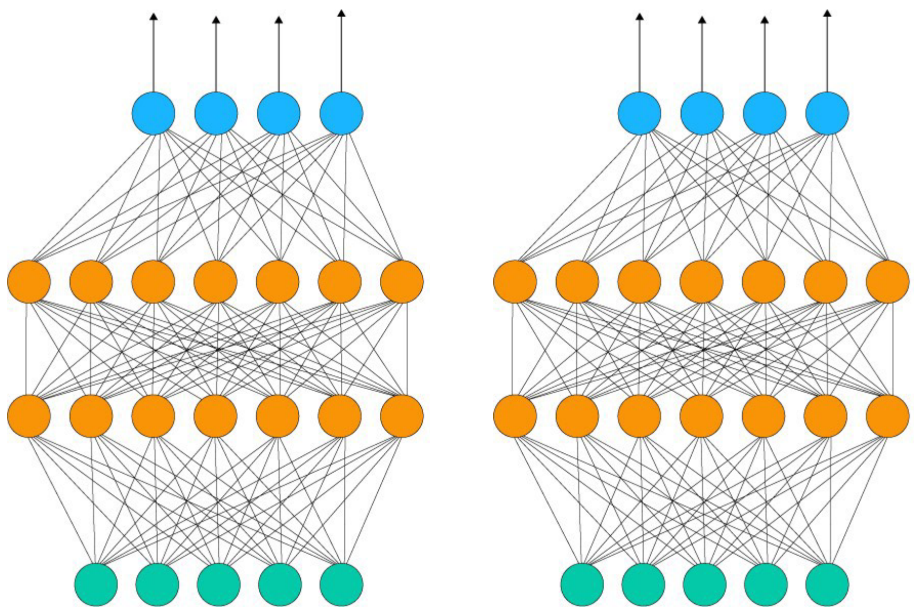
$$\left(\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right] - Q(S_t, A_t)\right)^2 ; MSE$$

DQN의 학습

❖ Target Network

- Target Network는 고정하고 Q-network만 학습

$$\left(\left[R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a)\right] - Q(S_t, A_t)\right)^2$$



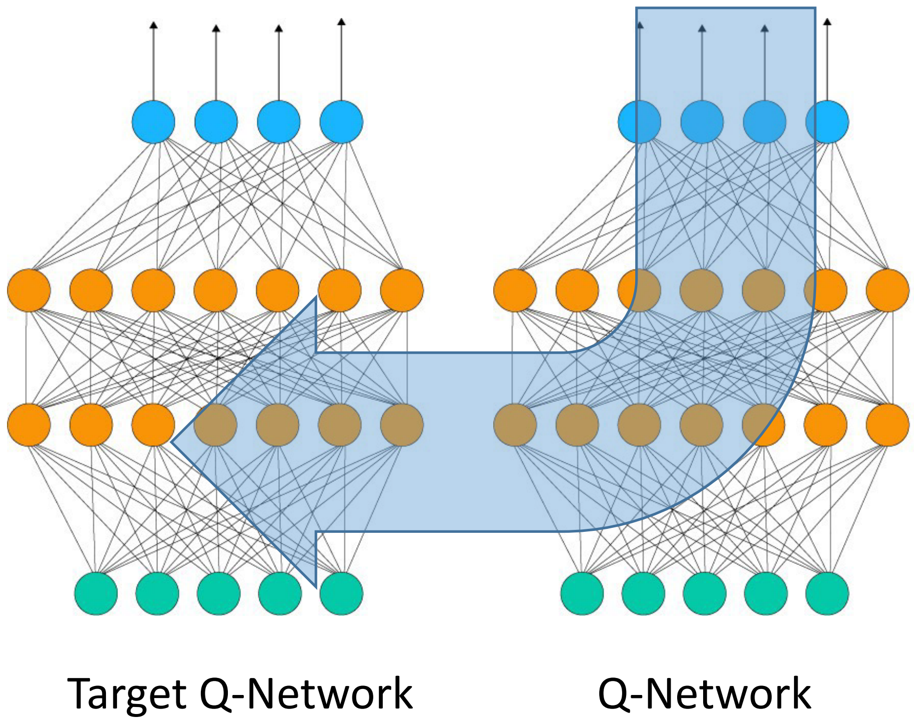
Target Q-Network Q-Network

DQN의 학습

❖ Target Network

- Target Network는 고정하고 Q-network만 학습

$$\left(\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right] - Q(S_t, A_t)\right)^2$$



DMQA

DQN의 학습

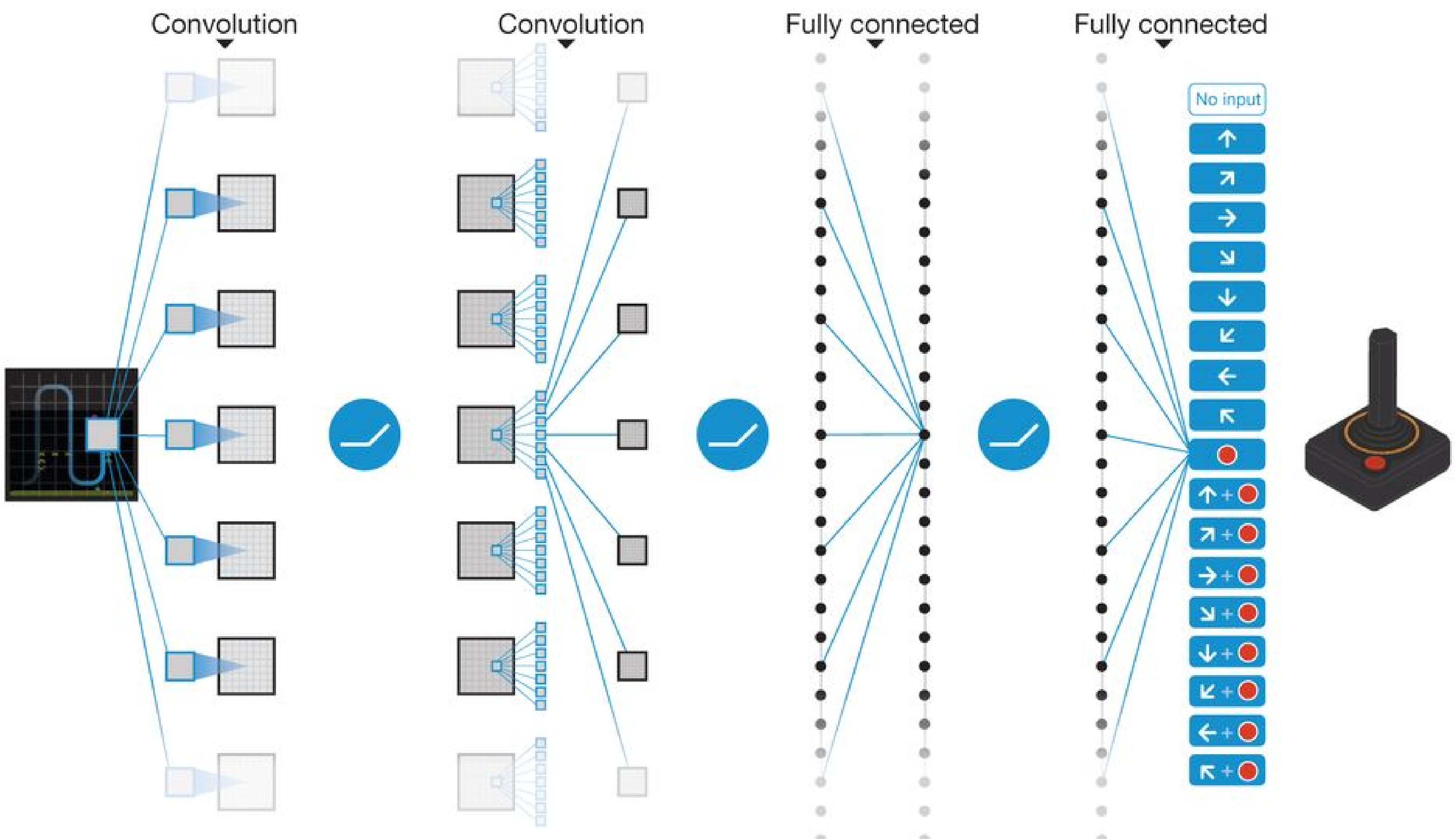
❖ Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DQN 성능

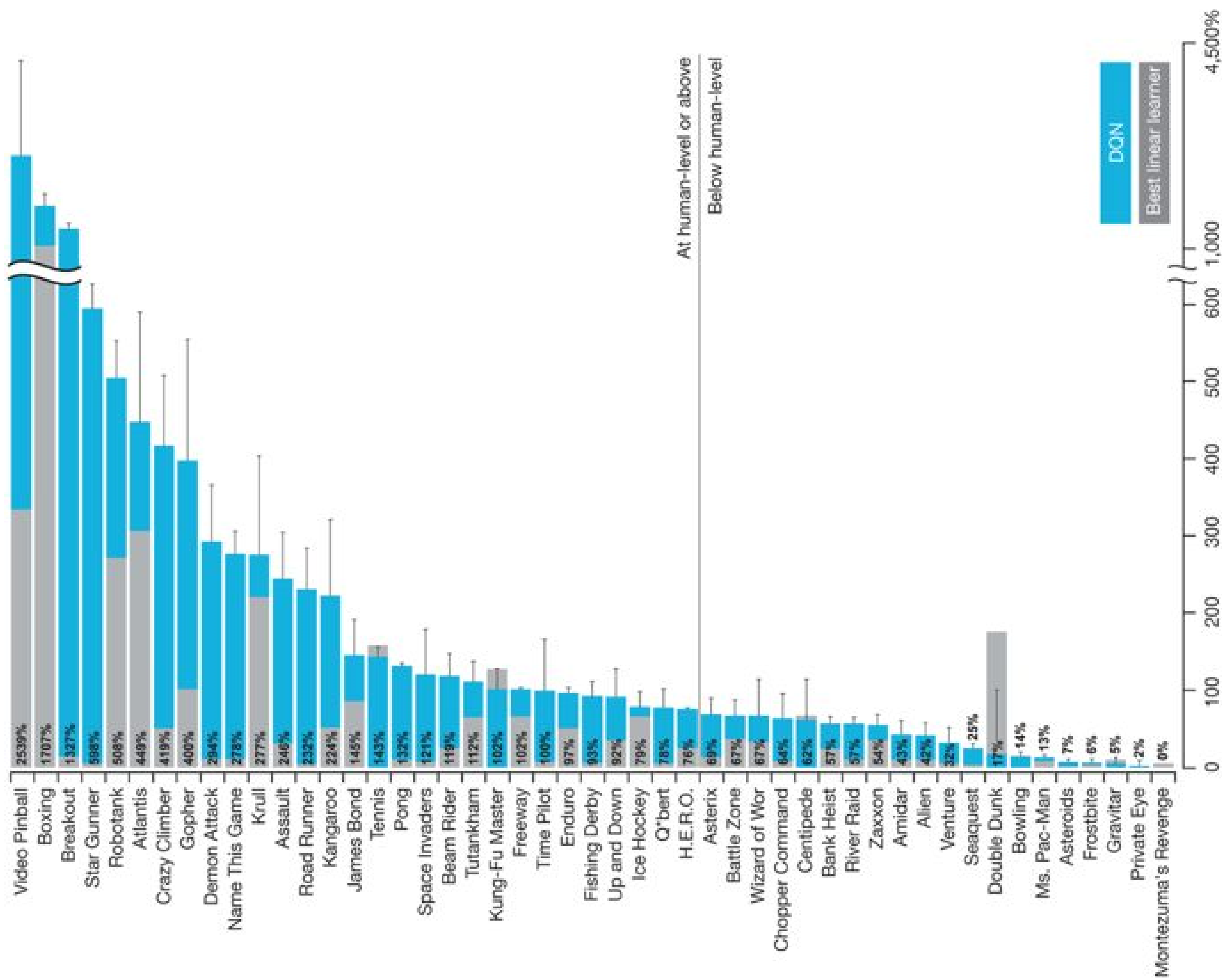
❖ 비디오 게임 (Atari)



DQN 성능

❖ 비디오 게임 (Atari)

DQN 성능



DQN
Best linear learner
At human-level or above
Below human-level
0
100
200
300
400
500
600
1,000
4,500%
Video Pinball
2539%
Boxing
1707%
Breakout
1327%
Star Gunner
598%
Robotank
508%
Atlantis
449%
Crazy Climber
419%
Gopher
400%
Demon Attack
294%
Name This Game
278%
Krull
277%
Assault
246%
Road Runner
232%
Kangaroo
224%
James Bond
145%
Tennis
143%
Pong
132%
Space Invaders
121%
Beam Rider
119%
Tutankham
112%
Kung-Fu Master
102%
Freeway
102%
Time Pilot
100%
Enduro
97%
Fishing Derby
93%
Up and Down
92%
Ice Hockey
79%
Q*bert
78%
H.E.R.O.
76%
Asterix
69%
Battle Zone
67%
Wizard of Wor
67%
Chopper Command
64%
Centipede
62%
Bank Heist
57%
River Raid
57%
Zaxxon
54%
Amidar
43%
Alien
42%
Venture
32%
Seaquest
25%
Double Dunk
17%
Bowling
14%
Ms. Pac-Man
13%
Asteroids
7%
Frostbite
6%
Gravitar
5%
Private Eye
2%
Montezuma's Revenge
0%

강화학습 문제를 푸는 방식

- ❖ **Action-Value 기법**

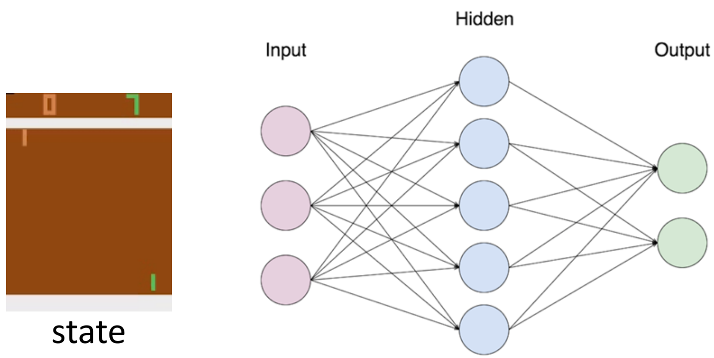
  Value-Based RL

- ❖ **그래디언트 기반 기법**

  Policy-Based RL

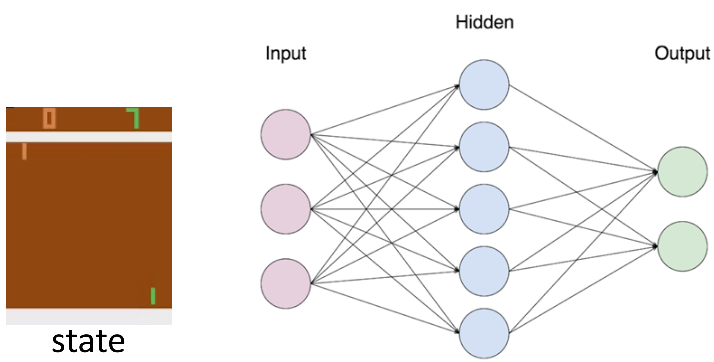
Policy Gradient & Q-Learning

❖ Policy gradient



$$P(a = a_1 | S = s)$$
$$P(a = a_2 | S = s)$$

❖ Q-learning



$$Q(a = a_1 | S = s)$$
$$Q(a = a_2 | S = s)$$

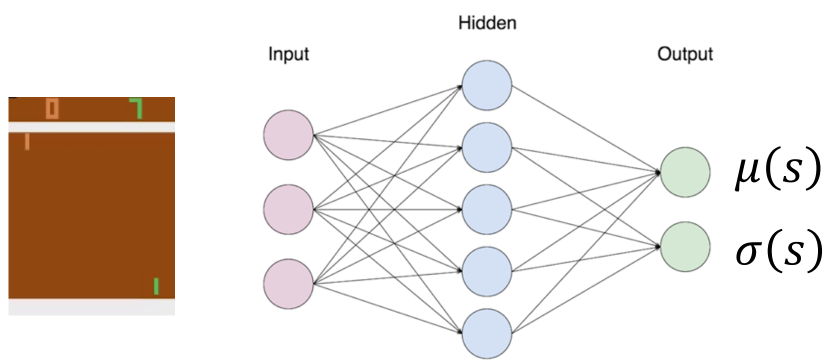
$Q(s, a) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s, a_0 = a]$: Q-function

Policy Gradient for Continuous Action

- DQN cannot solve continuous action tasks
- Policy gradient can simply solve the tasks (Gaussian policy parameterization)

$$\pi(s) = N\big(\mu(s), \sigma(s)\big)$$

$$a \sim N\big(\mu(s), \sigma(s)\big)$$

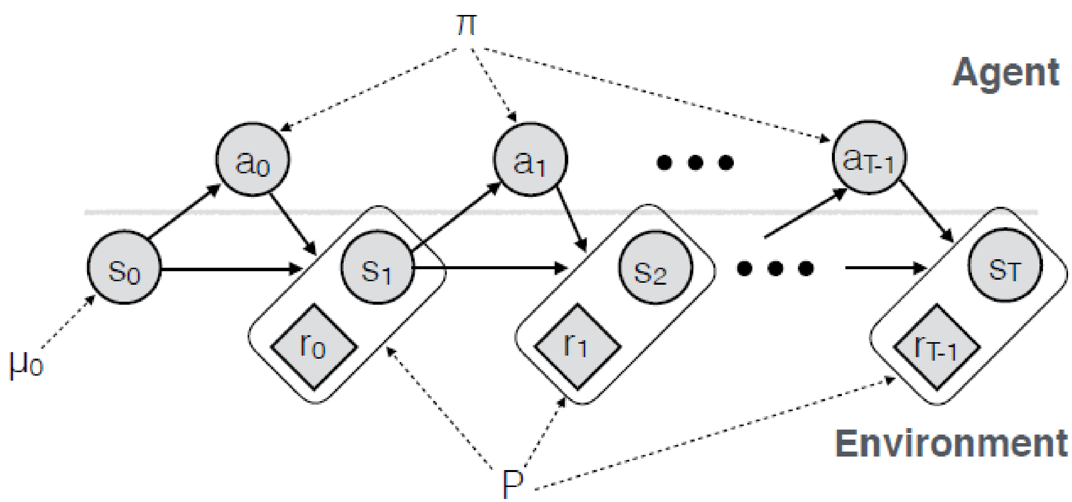


Sutton, R. S., et al., (2018). Reinforcement Learning: An Introduction, 338p

Training Policy Gradient

❖ Episodic update: REINFORCE

$$\max E[G|\pi_\theta] \qquad where\ G = \sum_{t=0}^{T-1} r_t$$



Gradient Estimator (optional)

❖ $E_{x \sim p(x|\theta)}[f(x)]$

$$\nabla_\theta E_x[f(x)] = \nabla_\theta \int p(x|\theta) f(x) dx$$

$$= \int \nabla_\theta p(x|\theta) f(x) dx$$

$$= \int \frac{\nabla_\theta p(x|\theta)}{p(x|\theta)} p(x|\theta) f(x) dx$$

$$= \int \nabla_\theta \log p(x|\theta)\, p(x|\theta) f(x) dx$$

$$= E_x[f(x) \nabla_\theta \log p(x|\theta)]$$

Sample $x_i \sim p(x|\theta)$, and compute $\hat{g}_i = f(x_i) \nabla_\theta \log p(x_i|\theta)$

Policy Gradient Estimator

- Now random variable x is a whole trajectory

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$$

$$\nabla_\theta E_\tau[G(\tau)] = E_\tau[G(\tau)\nabla_\theta \log p(\tau|\theta)]$$

- Just need to write out $p(\tau|\theta)$

$$p(\tau|\theta) = \mu(s_o)\prod_{t=0}^{T-1}[\pi(a_t|s_t,\theta)P(s_{t+1}, r_t|s_t, a_t)]$$

$$\log p(\tau|\theta) = \log \mu(s_o) + \sum_{t=0}^{T-1}[\log \pi(a_t|s_t,\theta) + \log P(s_{t+1}, r_t|s_t, a_t)]$$

$$\nabla_\theta \log p(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t,\theta)$$

$$\nabla_\theta E_\tau[G] = E_\tau\left[G\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t,\theta)\right]$$

Policy Gradient Methods

- REINFORCE (*MC policy gradient*)

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(s)\, G_t$$

- Actor Critic Policy Gradient (*mix*)

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(s)\, Q(s,a)$$

- Advantage Actor-Critic Policy Gradient (*mix*)

$$\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(s)\, [Q(s,a) - V(s)]$$

Policy Gradient Algorithm (REINFORCE)

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

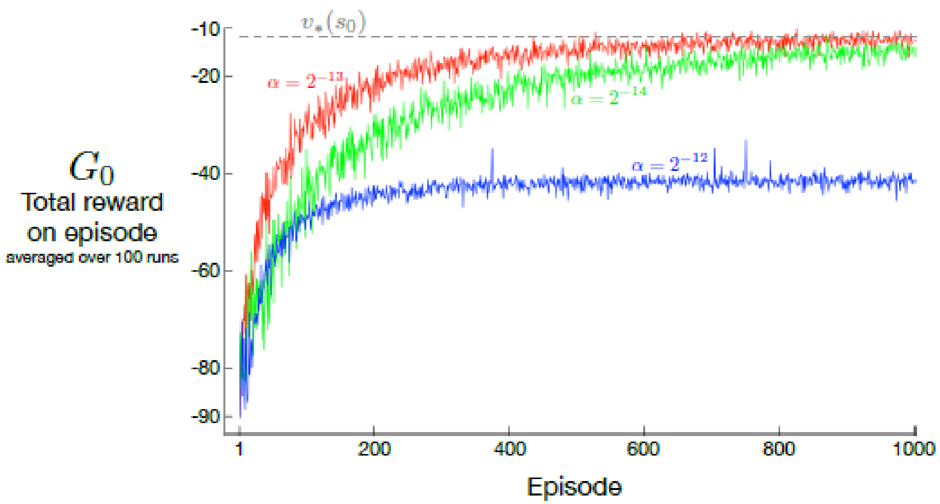


Figure 13.1: REINFORCE on the short-corridor gridworld (Example 13.1). With a good step size, the total reward per episode approaches the optimal value of the start state.

Connection Between Reinforcement Learning

❖ Action-value methods

- **Value-based RL**
- DQN

❖ Gradient-based methods

- **Policy-based RL**
- Policy Gradient

❖ Action-value methods & Gradient-based methods

- **Actor-Critic methods** (actor: policy-based / critic: value-based)
- A2C, A3C,…

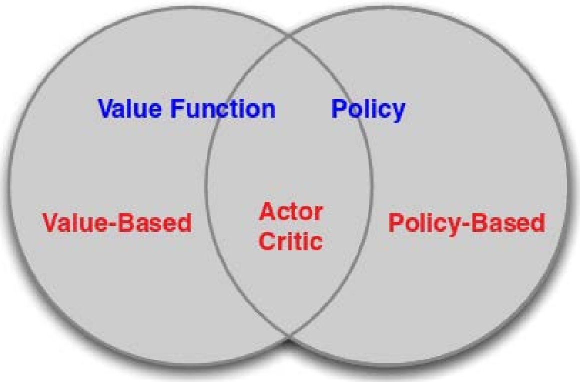
REINFORCE vs DQN

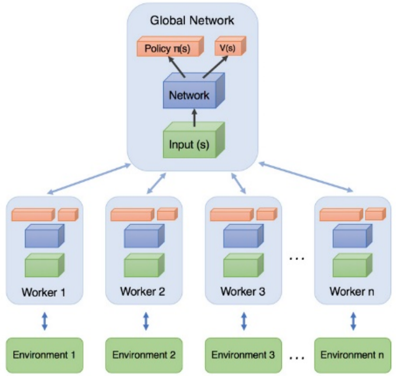
❖ Comparison between policy gradient and DQN

	REINFORCE	DQN
Estimation	Monte Carlo method	Bootstrapping (temporal difference)
Pros.	• Continuous action • Stochastic policies	• Low variance • Long episode
Cons.	• High variance • Local optimal	• No continuous action • Only deterministic policy

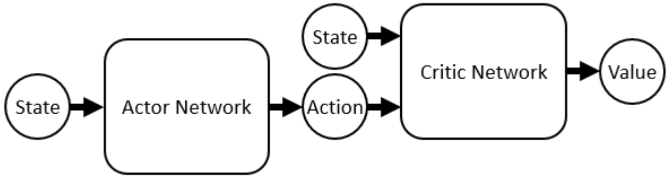
Actor-Critic Algorithms

- Actor-critic is hybrid of DQN & policy gradient
- Variants: A3C, DDPG, TRPO…





A3C



DDPG

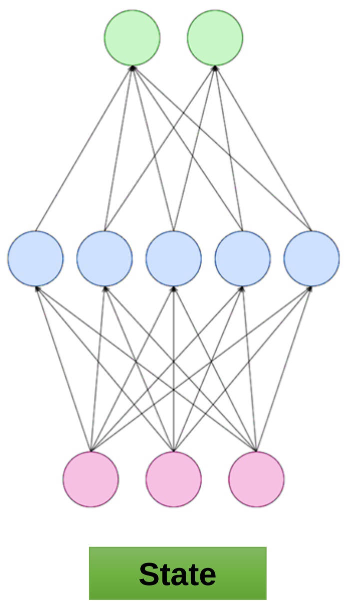
Actor-Critic Method

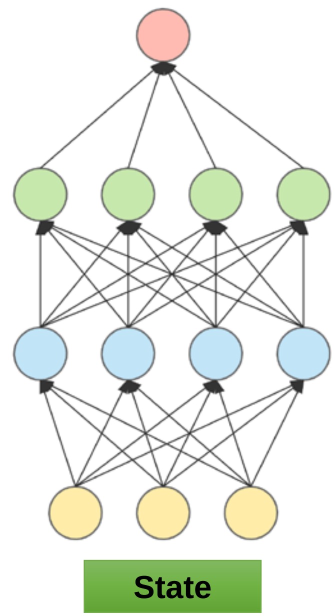
❖ Network architectures

❖ Actor: policy network / Critic: value network

$$\theta \leftarrow \theta + \alpha \mathbf{G} \nabla_\theta \log \pi(a_t|s_t, \theta)$$

Actor



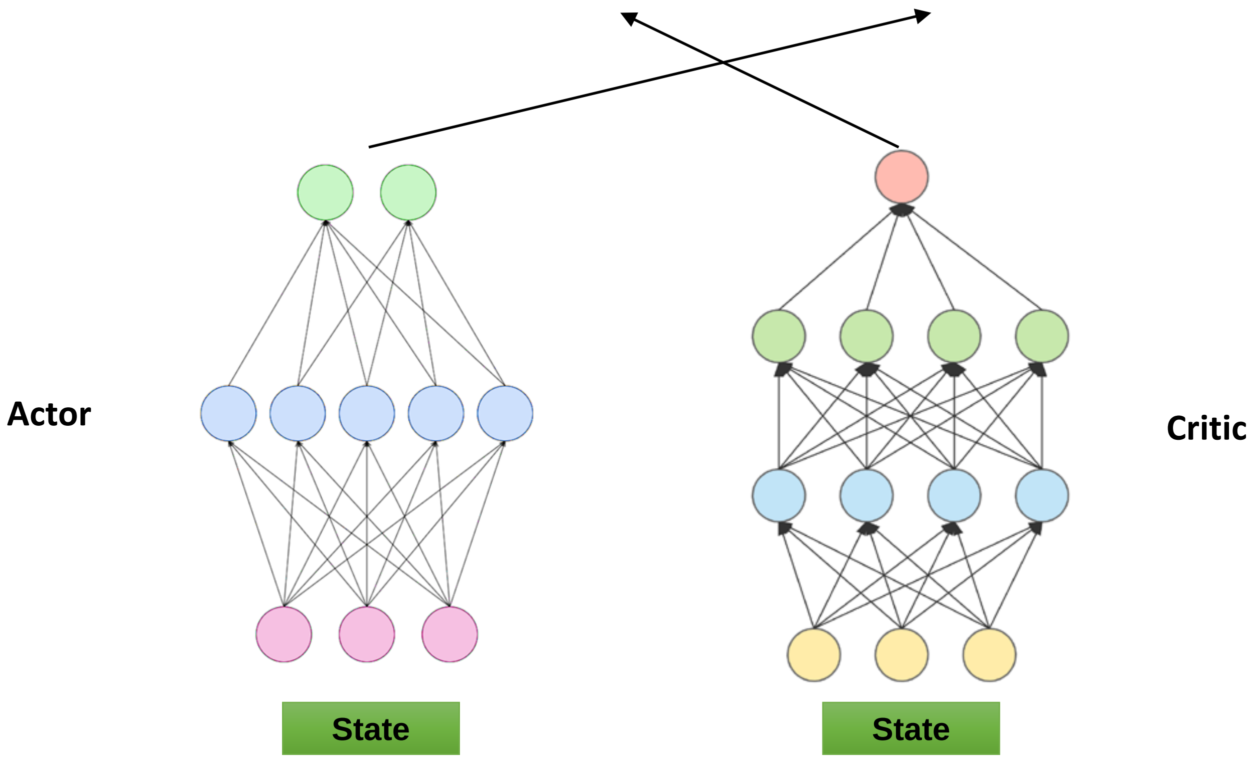


Critic

Actor-Critic Method

- Network architectures
- Actor: policy network / Critic: value network

$$\theta_a \leftarrow \theta_a + \alpha \boldsymbol{V(s|\theta_c)} \nabla_{\theta_a} \log \boldsymbol{\pi(a_t|s_t, \theta_a)}$$



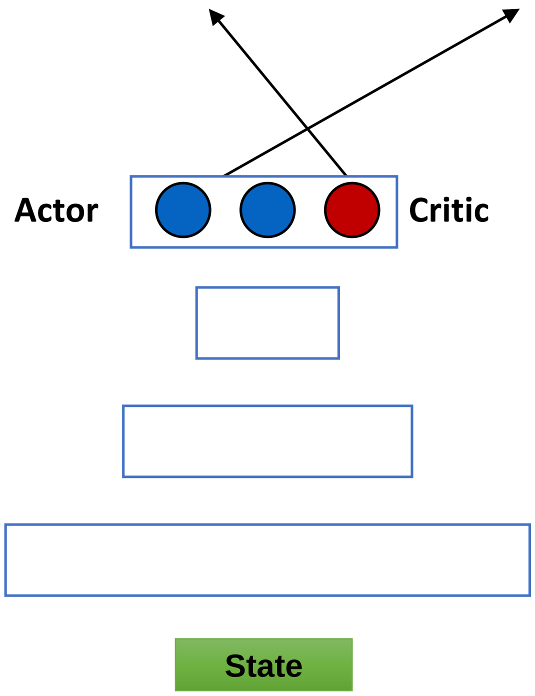
$V(s) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s\ ]$: V-function

Actor-Critic Method

❖ Network architectures

❖ Actor: policy network / Critic: value network

$$\theta_a \leftarrow \theta_a + \alpha \boldsymbol{V(s|\theta_c)} \nabla_{\theta_a} \log \boldsymbol{\pi(a_t|s_t, \theta_a)}$$



$V(s) = E[r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots | s_0 = s\,]$: V-function

Actor-Critic Method

- Network architectures
- Actor: policy network / Critic: value network
- Update actor

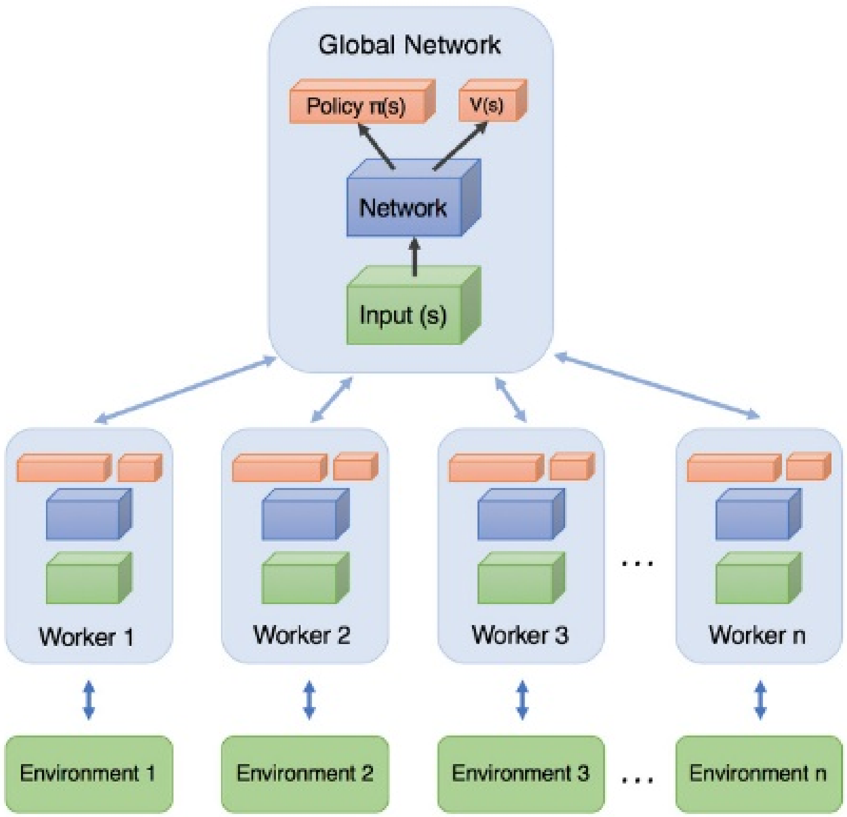
$$\theta_a \leftarrow \theta_a + \alpha \boldsymbol{Score} \nabla_{\theta_a} \log \boldsymbol{\pi(a_t | s_t, \theta_a)}$$

- $\boldsymbol{Score}$
  - G — *REINFORCE*
  - $Q(s,a)$ — *Q Actor-Critic*
  - $A(s,a) = Q(s,a) - V(s)$ — *Advantage Actor-Critic*
  - $\delta = r + \gamma V(s_{t+1}) - V(s_{t+1})$ — *TD Actor-Critic*

A3C

❖ Asynchronous Advantage Actor-Critic (A3C)

- Use CPU thread
- Multiple Actor-Critic
- Advantage
- Episodic update



A3C

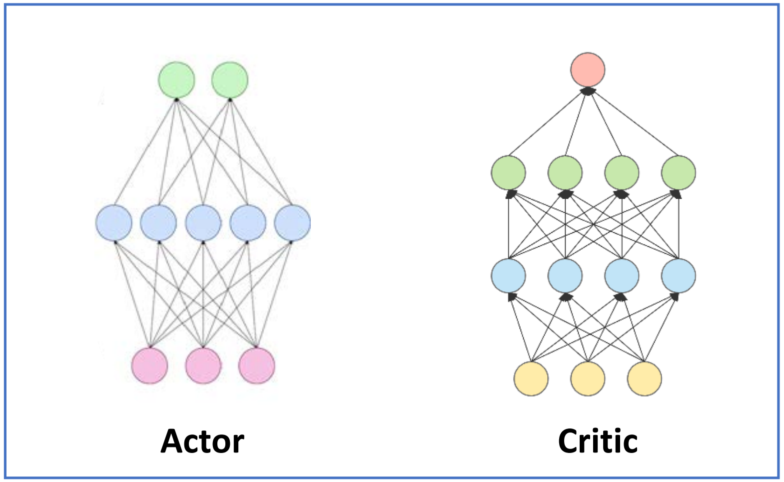
❖ No replay memory, On policy

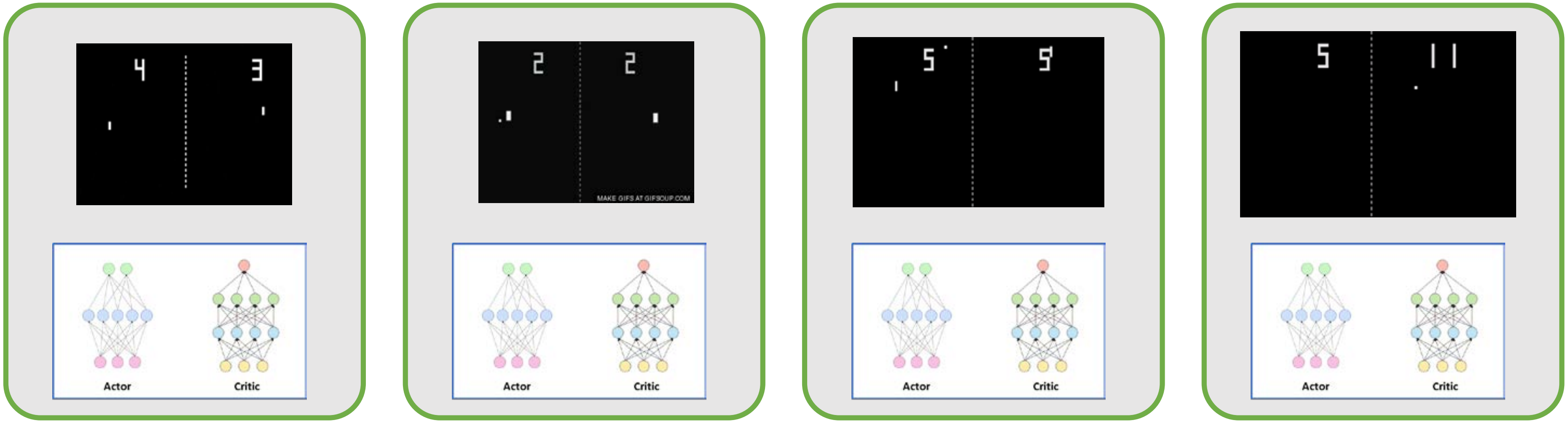
Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Asynchronous Update

❖ Uncorrelated experience

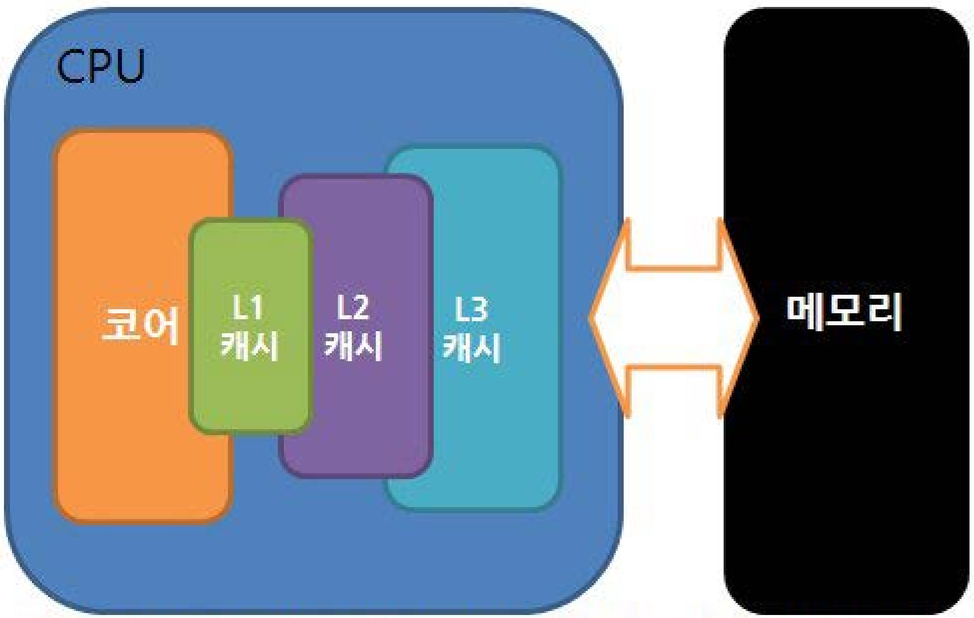




DMQA

Asynchronous Update

❖ Fast communication speed between CPU & memory



DMQA

A3C Performance

❖ 5 Atari games

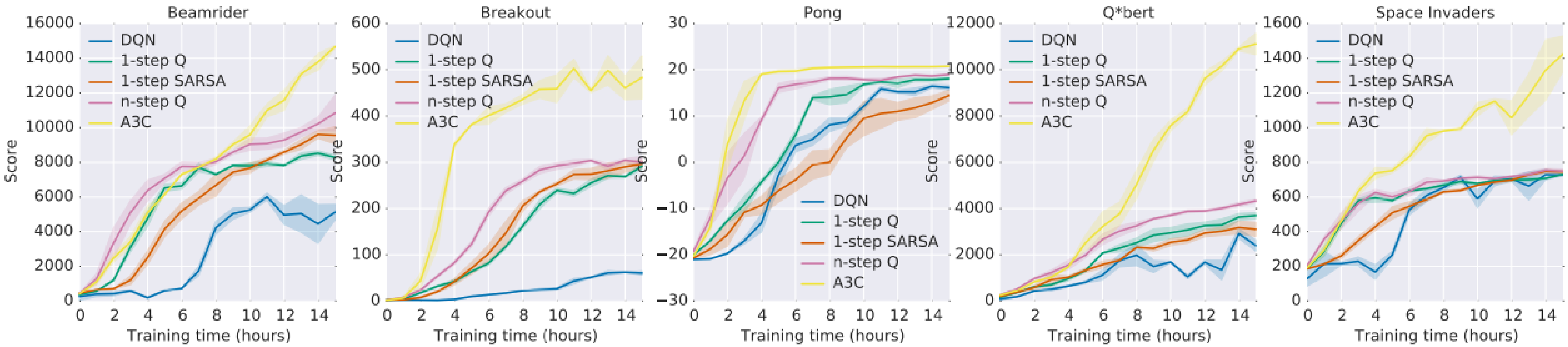


Figure 1. Learning speed comparison for DQN and the new asynchronous algorithms on five Atari 2600 games. DQN was trained on a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores. The plots are averaged over 5 runs. In the case of DQN the runs were for different seeds with fixed hyperparameters. For asynchronous methods we average over the best 5 models from 50 experiments with learning rates sampled from $LogUniform(10^{-4}, 10^{-2})$ and all other hyperparameters fixed.

Deterministic Policy for Continuous Action

- In Deep RL, policy is parametrized by neural networks (θ)

- Deterministic policy

$$\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{S})$$

- Stochastic Policy

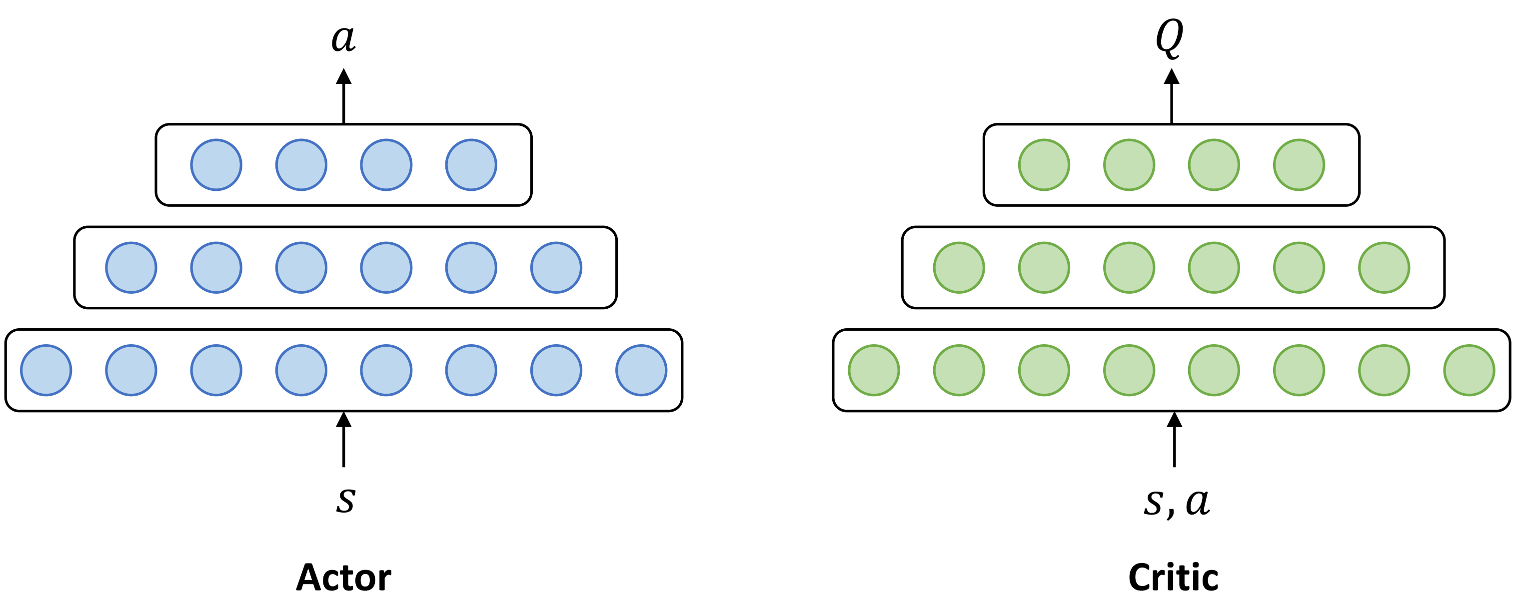
$$\boldsymbol{a} \sim \boldsymbol{\pi}(\boldsymbol{S})$$

- REINFORCE

$$\nabla_\theta E_\tau[G] = E_\tau\left[G \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t | s_t, \theta)\right]$$

Actor-Critic 기법

- ❖ Actor는 action을 결정 $a = \pi(s;\theta_a)$
- ❖ Critic은 actor를 평가 $Q(s,a;\theta_c) \approx \mathbb{E}_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \,|S_t = s, A_t = a]$,
- ❖ A3C[1], DDPG[2], ACER[3] 등 다양한 알고리즘이 있음

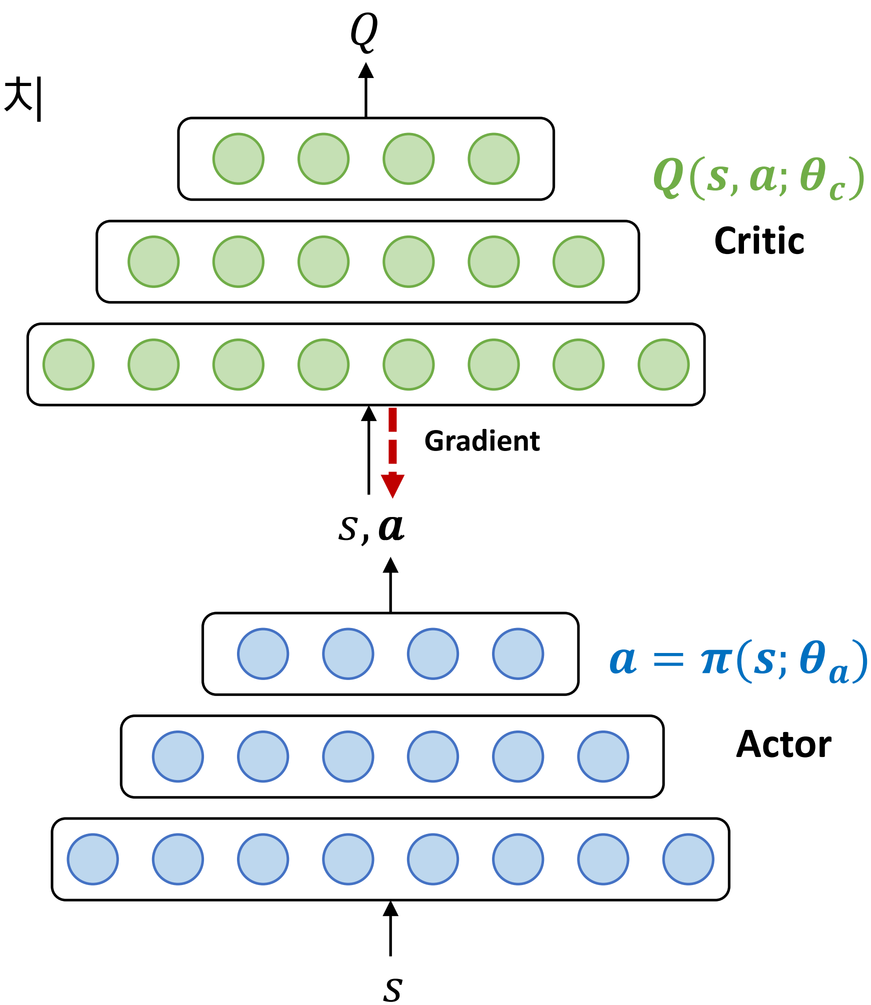


Deep Deterministic Policy Gradient (DDPG)

- Actor-Critic 기법
- Actor와 Critic 신경망이 순차적으로 배치
- Critic Network의 아웃풋 Q를 최대화 하는 policy $\boldsymbol{\pi(s;\theta_a)}$ 학습

$$\nabla_{\theta_a} E_\tau[Q(s,a)] = E_\tau\left[\nabla_{\theta_a} Q(s,\pi(s;\theta_a);\theta_c)\right]$$

$$= \frac{\partial Q}{\partial a}\frac{\partial \pi}{\partial \theta_a}$$



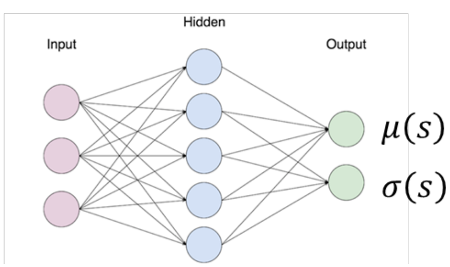
DDPG

- Deep deterministic policy gradient (DDPG)

  Deterministic policy gradient† + DQN (replay memory)

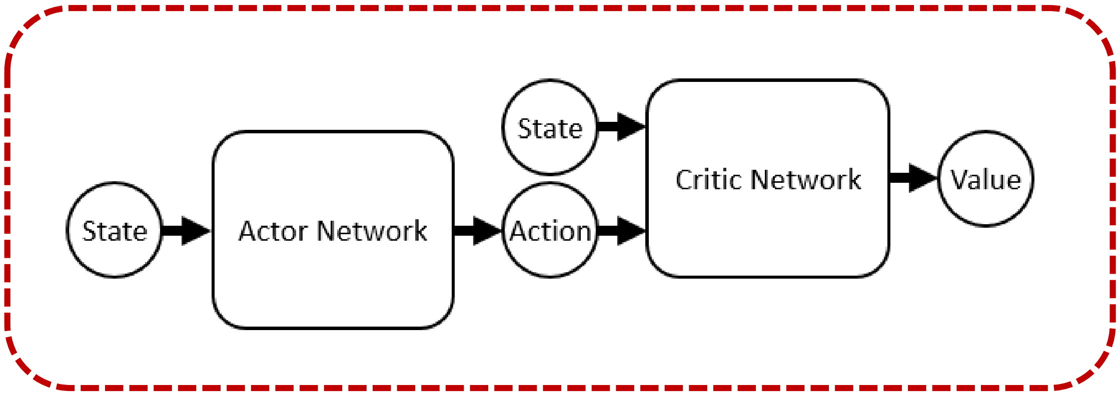
- Continuous action (categorical action)
- Continuous update (not episodic update)

$$\pi(s) = N(\mu(s), \sigma(s))$$
$$a \sim N(\mu(s), \sigma(s))$$



Gaussian policy

$$\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{S})$$



DDPG

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
† Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014, June). Deterministic policy gradient algorithms. In *ICML*.

DDPG

❖ Incorporate replay memory and target network ideas from DQN for increased stability

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

DDPG Performance

❖ Continuous control examples

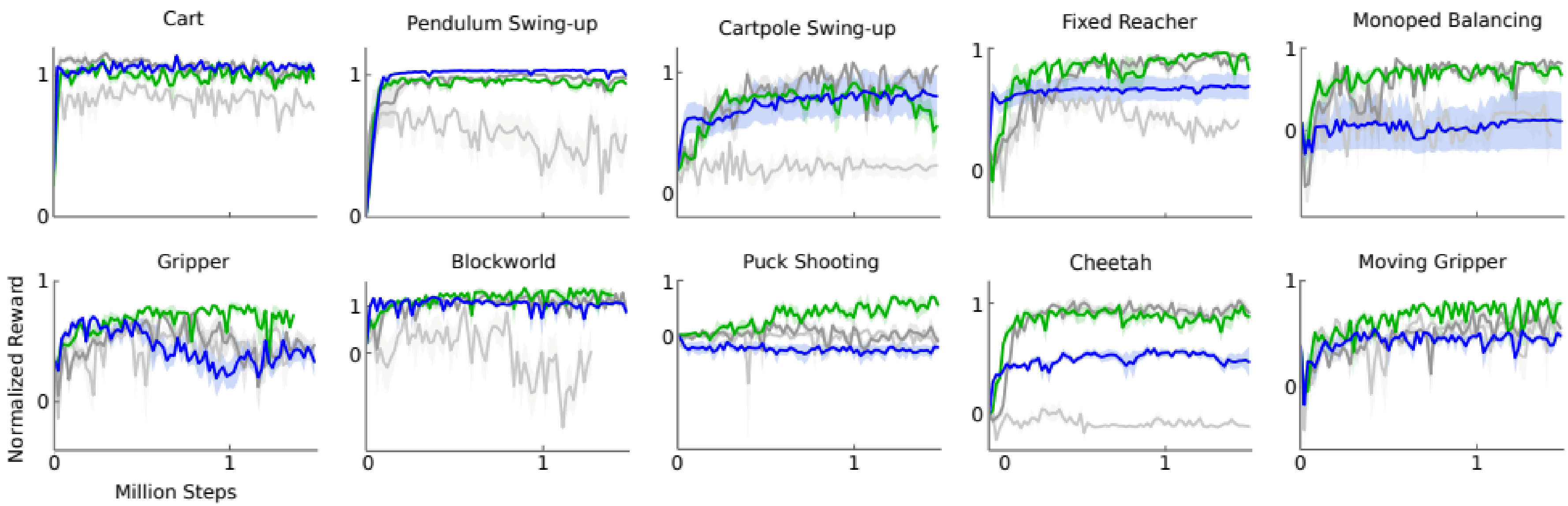


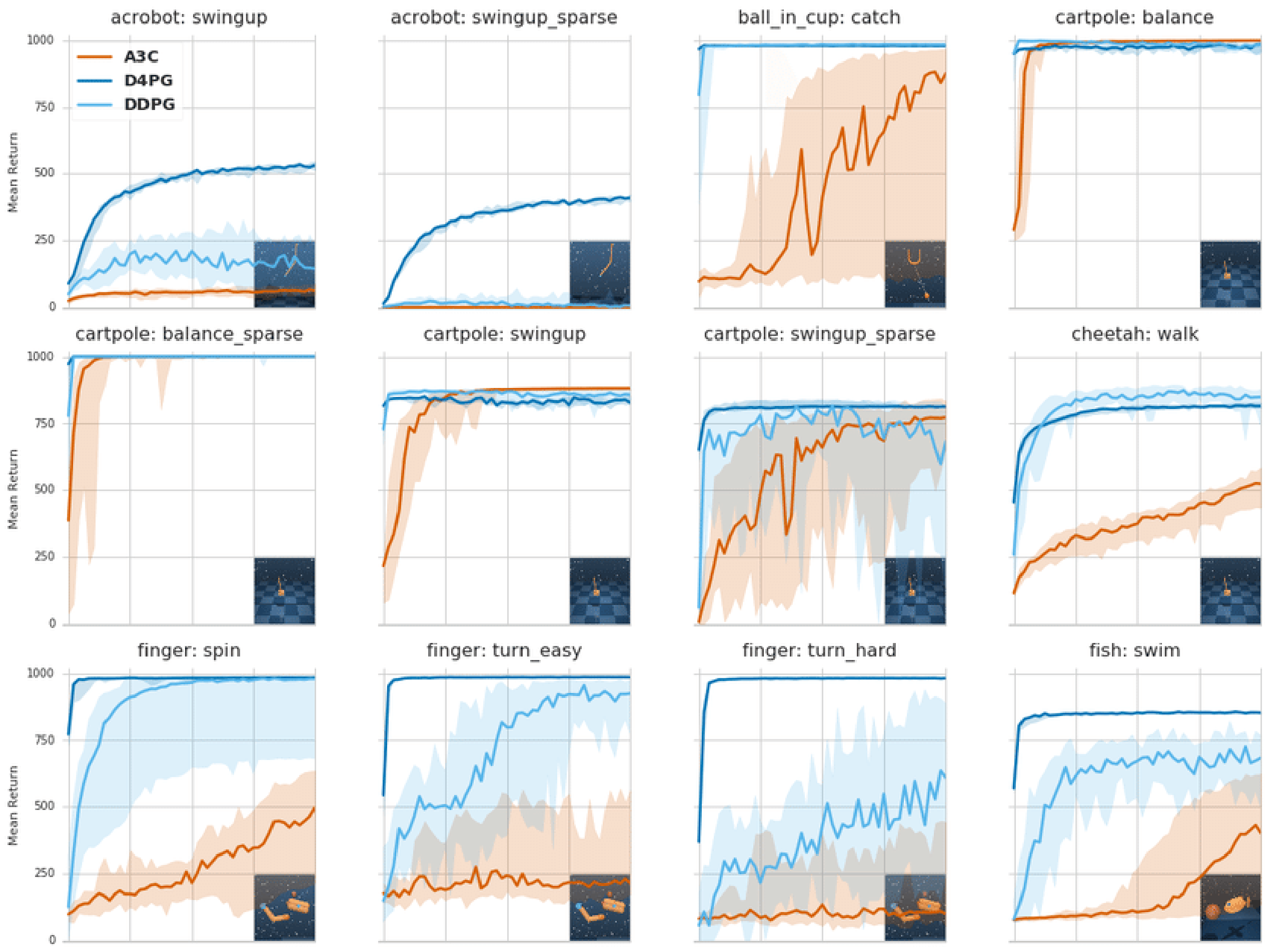
Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

DDPG Performance

- Continuous control examples

DDPG vs A3C

❖ Continuous control examples



Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. D. L., ... & Lillicrap, T. (2018). DeepMind Control Suite. *arXiv preprint arXiv:1801.00690*.

Future of RL Algorithms

❖ Efficient Learning

❖ Robust Training

❖ Complex problem

- Hierarchical RL
- Multi-agent RL