Actor-Critic Algorithms

2019. 05. 24

Data Mining & Quality Analytics Lab.

이 영 재

DMQA

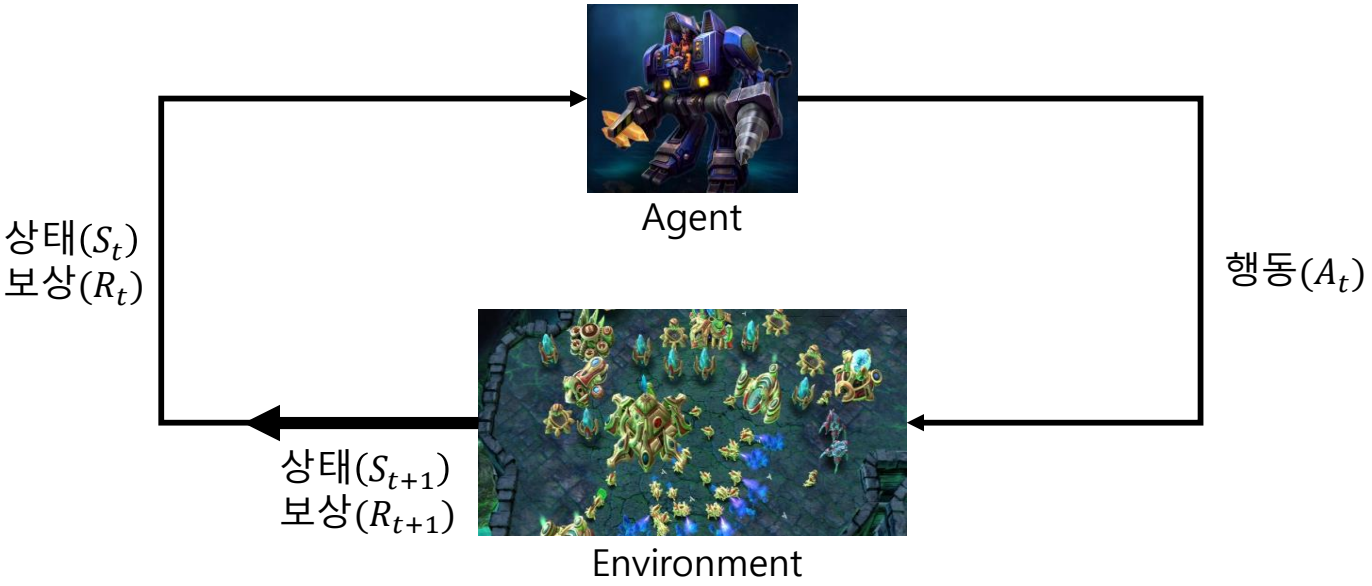
목차

DMQA

목차

Reinforcement Learning

❖ 환경과 상호작용하며 목표를 달성하는 에이전트를 다루는 학습 방법



✓ $0 \leq \gamma < 1$, 디스카운트 요소로 미래를 포함해 보상들을 반영하며 보상의 총합이 발산하지 않도록 하는 역할

DMQA

Reinforcement Learning

❖ 미래에 받을 보상의 합을 최대화하는 정책(Policy) $\boldsymbol{\pi(a|s)}$를 찾는 것

- $\boldsymbol{\pi(a|s)}$는 에이전트의 행동 함수로써 상태(s)에서 취할 행동(a)을 알려줌

Elements of RL

❖ 정책(Policy) - $\pi(a|s)$

- 주어진 상태에 대해 행동을 결정하는 역할

❖ 가치 함수(Value Function) - $v_\pi(s), q_\pi(s,a)$

- 미래 보상의 총합

Value Function

❖ 상태 가치 함수(State-Value Function)

- $v_\pi(s) = E_\pi[G_t|S_t = s]$

$$= E_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$$

$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

❖ 행동 가치 함수(Action-Value Function)

- $q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$

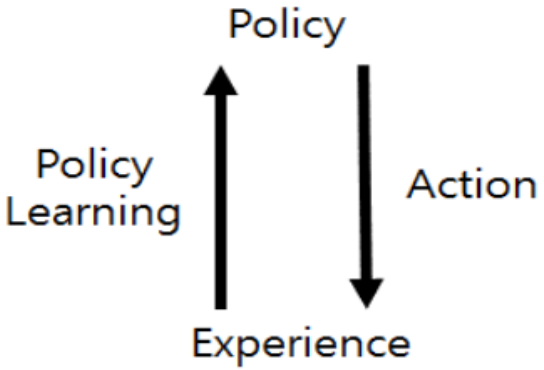
$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, A_t = a]$$

$$= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

Q-Leaning

❖ Q-함수(Q Function)

- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
- 에이전트는 특정 상태(s)에서 자신이 취할 수 있는 행동(a) 중 이득이 되는 방향으로 학습하는 방법

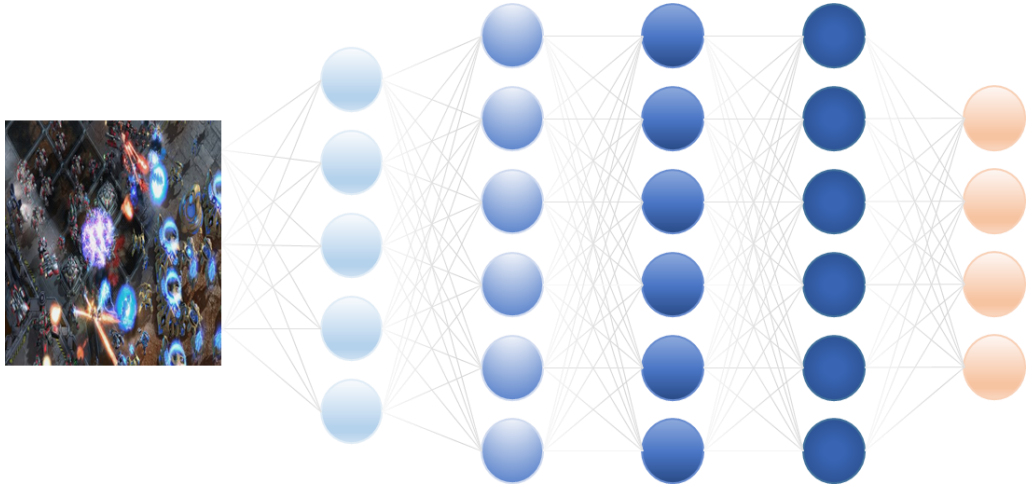


DMQA

Deep Q-Network

❖ 아이디어

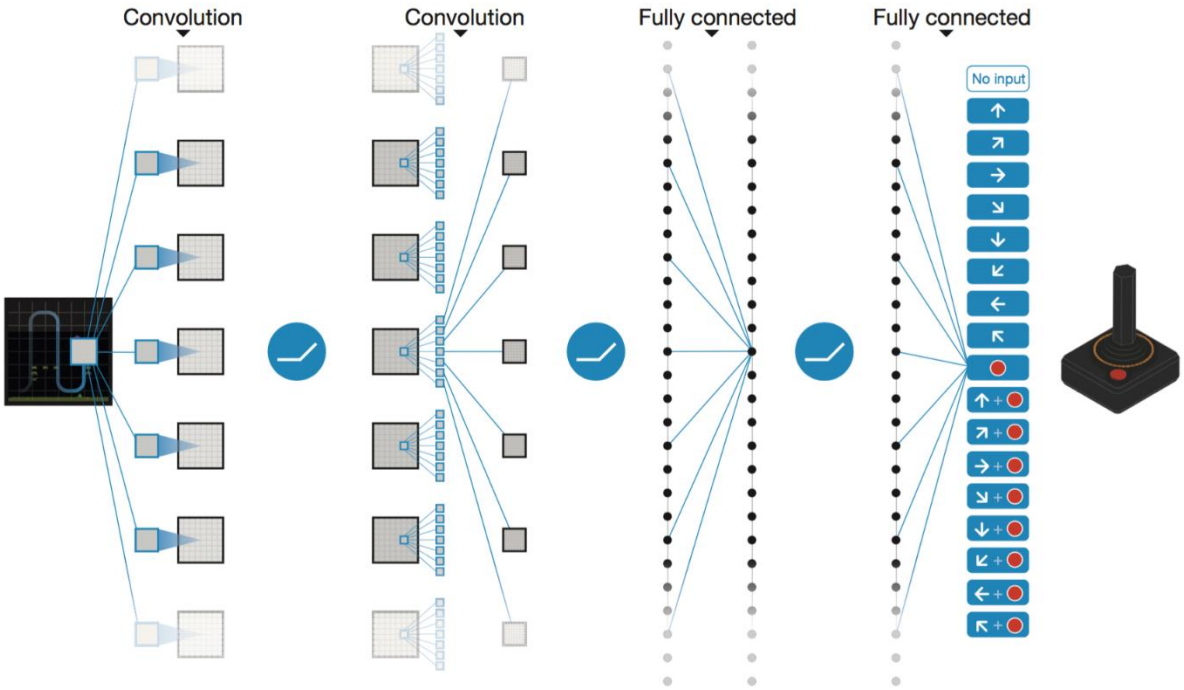
- 딥 러닝을 이용한 강화학습
- Q-table에 저장하지 않고 상태(s)와 행동(a)을 입력으로 하여 보상(r) 출력



Deep Q-Network

❖ Deep Q-Network(DQN)

- 상태(s)를 입력으로 여러 개의 Q값을 출력으로 하여 가장 큰 Q값에 대응하는 행동(a)를 선택



DMQA

Deep Q-Network

❖ Issues

- 순차적인 상태(s) 데이터 간의 높은 상관관계(Correlation between samples)
- Target y값이 움직이는 현상(Non-stationary targets)

❖ Solutions

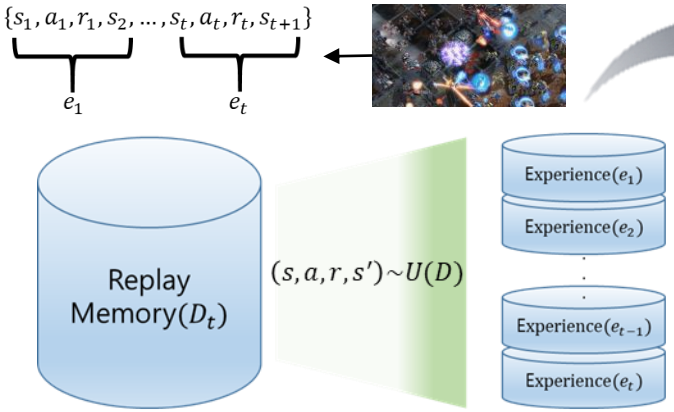
- Experience replay(Replay memory)
- Fixed Q-targets

Deep Q-Network

❖ Experience Replay & Fixed Q-target

- w_i^-는 매 C Step마다 w_i로 업데이트(논문에서 C=10000 설정)
- C번의 Iteration동안 Q-Learning 업데이트 시 Target 움직임 방지

$$\nabla_{w_i}\mathcal{L}(w_i) = E[(r + \gamma \max_{a'} Q(s', a', w_{i-1}) - Q(s, a, w_i))\nabla_{w_i} Q(s, a, w_i)]$$



$$\mathcal{L}_i(w_i) = E_{(s,a,r,s')\sim U(D)}\left[(r + \gamma \max_{a'} Q(s', a'; w_i) - Q(s, a; w_i))^2\right]$$

C-step 마다

$$\mathcal{L}_i(w_i) = E_{(s,a,r,s')\sim U(D)}\left[(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i))^2\right]$$

Target y_i

DMQA

Deep Q-Network

❖ 단점

- Replay Memory 사용 → 많은 메모리의 사용
- 느린 학습 속도
- 불안정한 학습 과정
- Starcraft II와 같은 수 많은 상태(s)와 행동(a)가 존재하는 게임에는 학습이 어려움

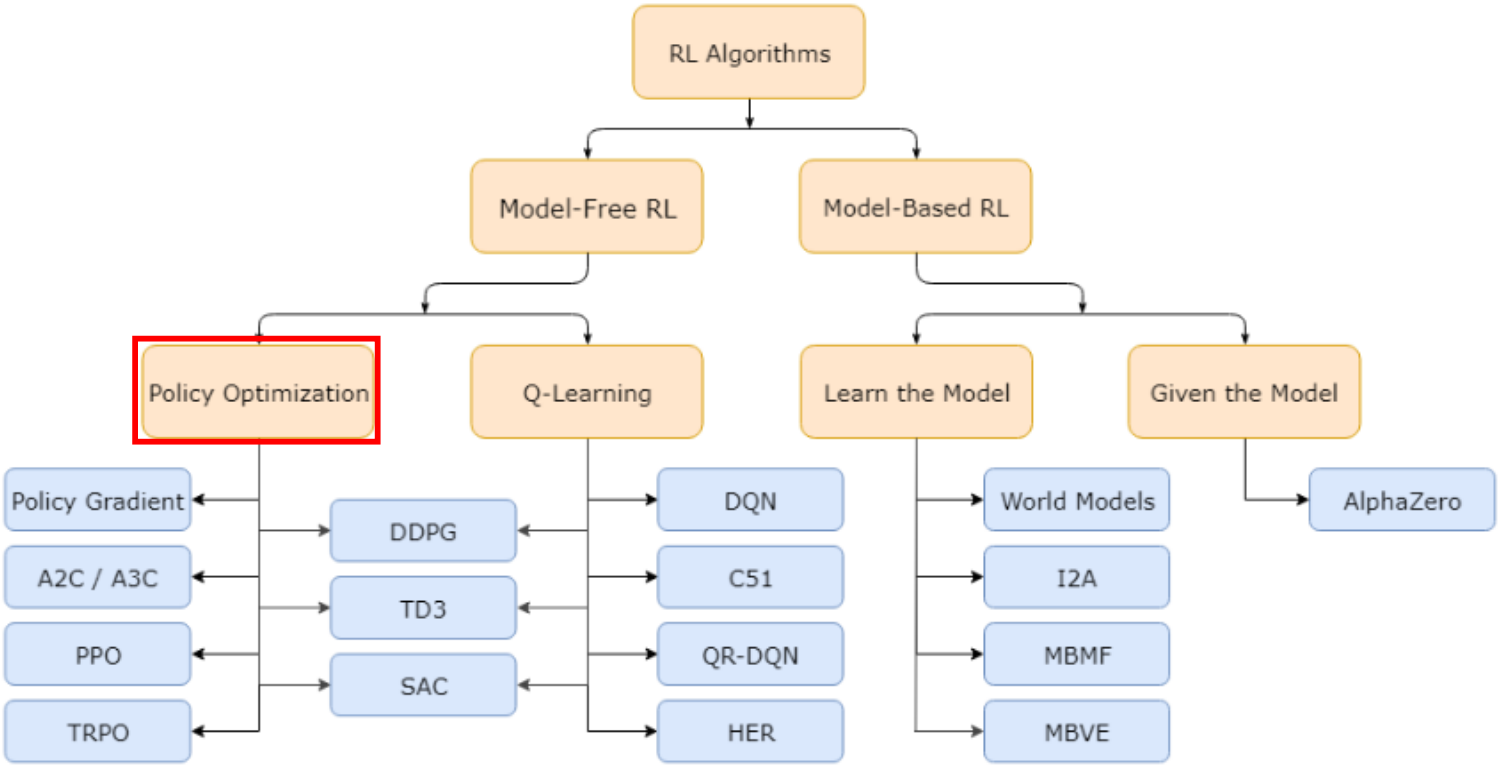
❖ Asynchronous Advantage Actor-Critic(A3C)

- 데이터 사이의 상관관계를 비동기적 업데이트로 해결(Asynchronous)
- 빠른 학습 속도
- Replay Memory를 사용하지 않고, Policy Gradient 알고리즘을 더함
- Starcraft II와 같은 게임에서 학습이 잘 되는 장점

목차

DMQA

Policy-based Method

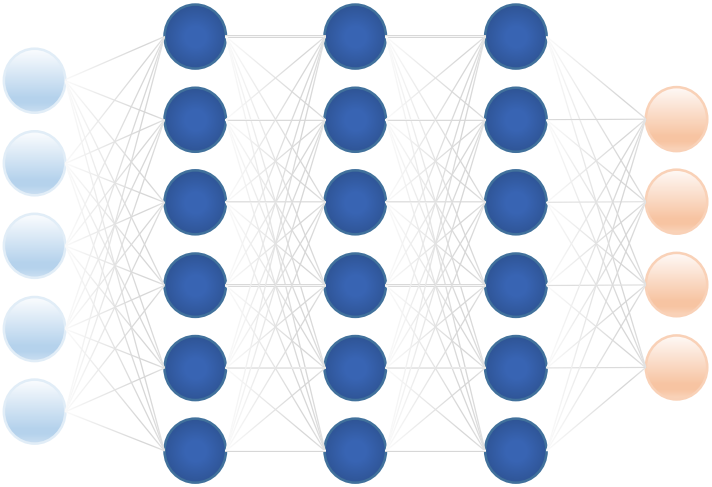


출처 : https://mclearninglab.tistory.com/entry/OpenAI-Spinning-Up

Policy Gradient

❖ Policy Gradient

- 파라미터 θ에 대한 정책(Policy) $\pi(a|s;\theta)$를 직접 학습

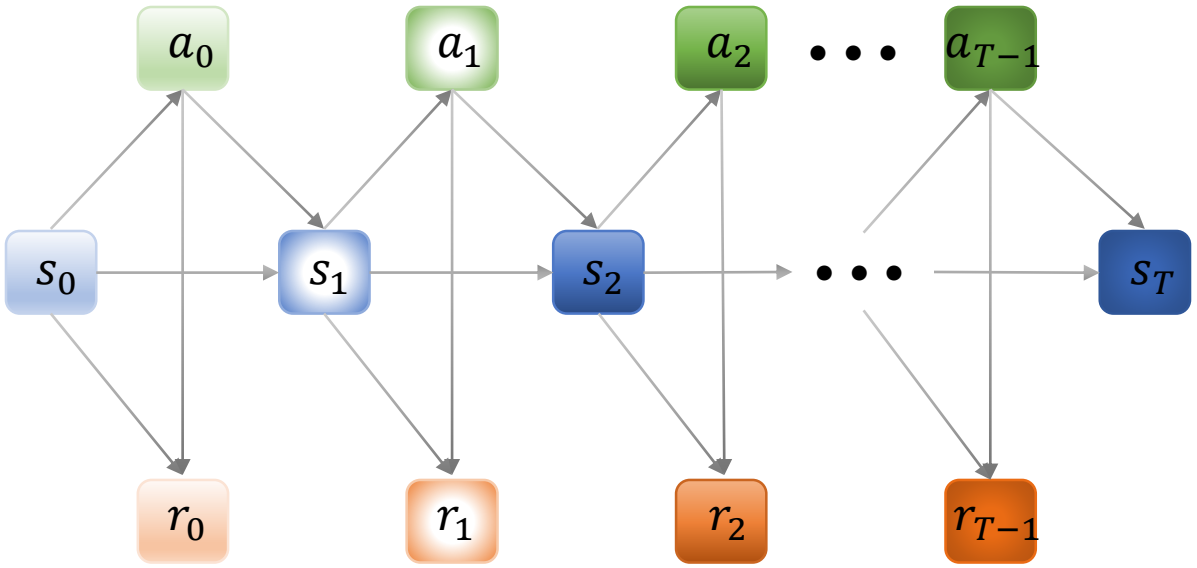


$$P(A = a_1|S = s)$$
$$P(A = a_2|S = s)$$
$$P(A = a_3|S = s)$$
$$P(A = a_4|S = s)$$

Policy Gradient Algorithms

❖ REINFORCE

- Episode$(\tau) = s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots, s_T$
- $J(\theta) = E[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta] = E[r_1 + r_2 + r_3 + \cdots + r_\tau | \pi_\theta]$



DMQA

Policy Gradient Algorithms

❖ REINFORCE

- $J(\theta) = E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = E[r_1 + r_2 + r_3 + \cdots + r_\tau|\pi_\theta]$
- $\theta' = \theta + \alpha\nabla_\theta J(\theta)$, $\nabla_\theta J(\theta)$ = Policy Gradient

$$\nabla_\theta E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = E_\tau \nabla_\theta [\sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t) r_{t+1}]$$

$$\approx E_\tau[\nabla_\theta \sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t) G_t],$$

$$where\ G_t = \sum_{t=0}^{T-1} \gamma^t r_{t+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-1} r_T$$

Discounted G_t → 단순 보상의 합 발산 방지

목차

DMQA

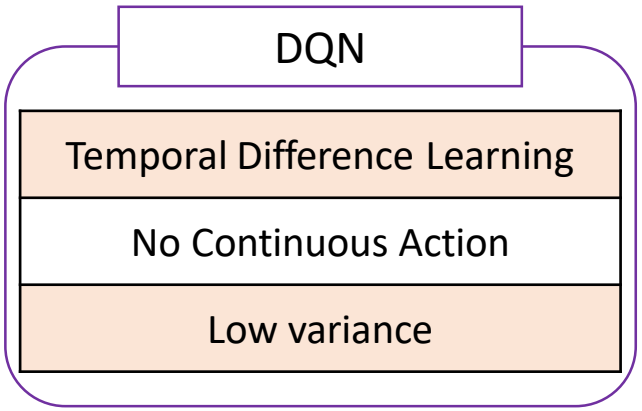
DQN vs REINFORCE

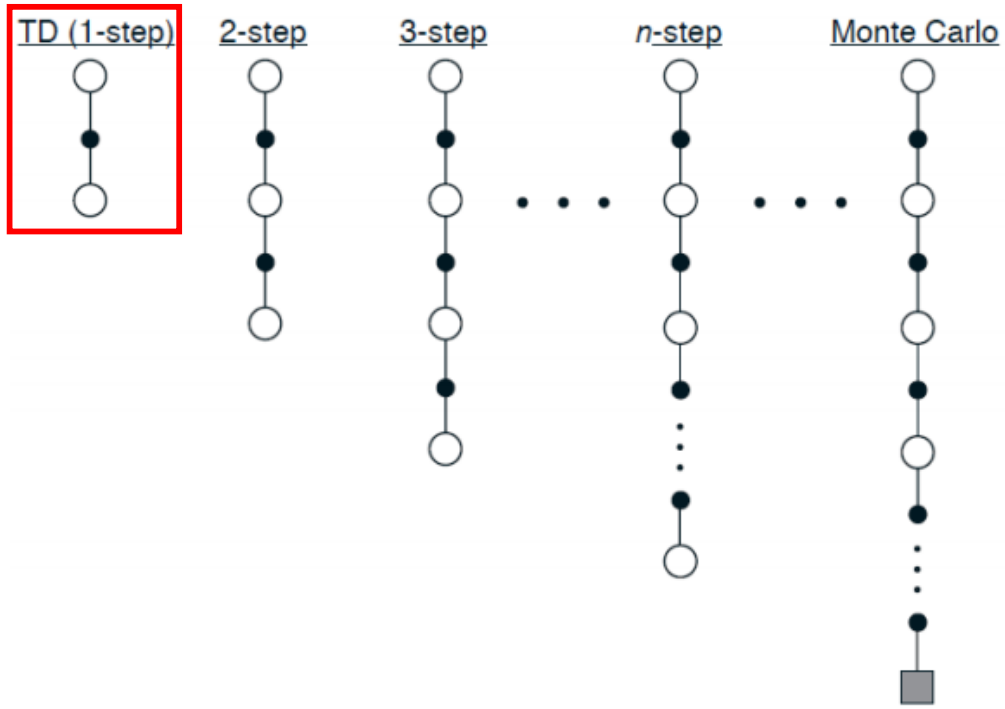
❖ DQN

- TD : Epsiode(S, A, R, Update, S, A, R, Update, S, A, R, Update, S, A, R, Terminate, Update)

  → Low Variance, High Bias

- Continuous Action 학습이 어려움



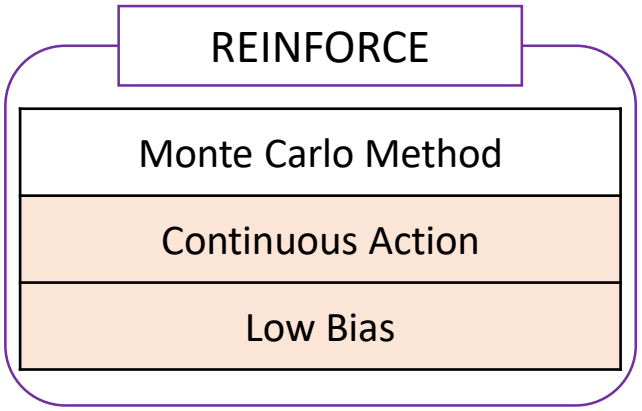


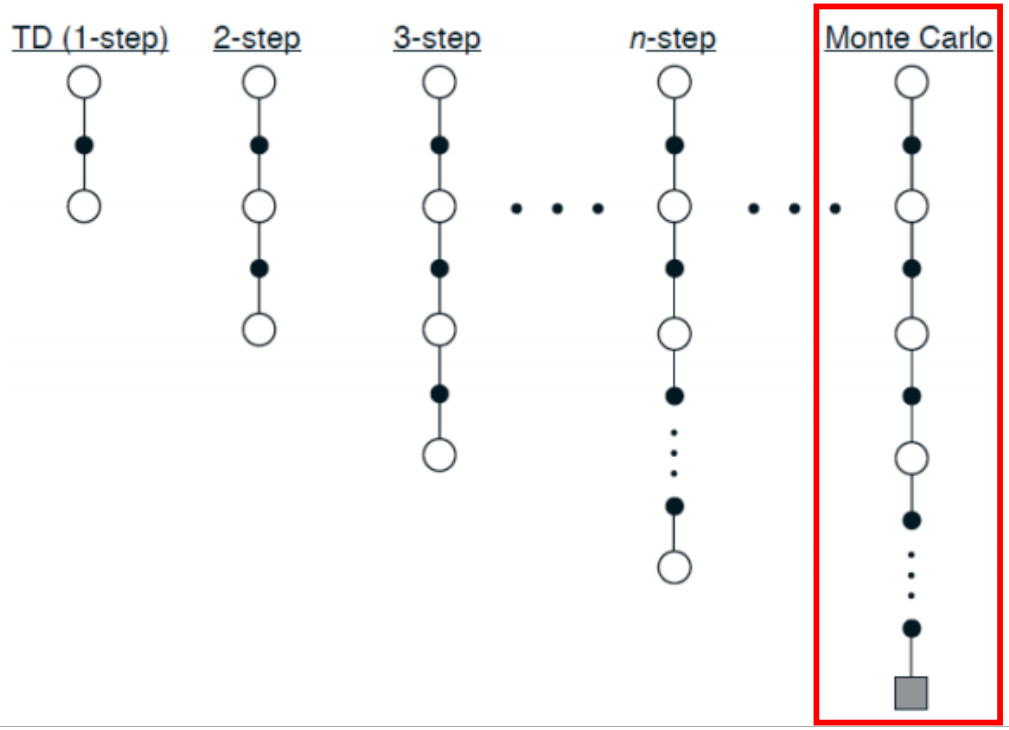
DMQA

DQN vs REINFORCE

❖ REINFORCE

- MC : Episode(S, A, R, S, A, R, S, A, R, S, A, R, Terminate, Update) → High Variance, Low Bias
- Continuous Action 학습 가능



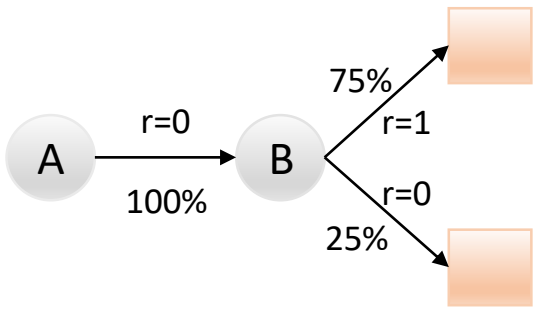


DMQA

DQN vs REINFORCE

❖ MC Method vs TD Learning

- Example
- 상태(A, B) ; $\gamma = 1$; 1-Step



$$TD(0): V(A) = V(A) + 1 * \big(0 + 1 * V(B) - V(A)\big) = 3/4$$

$$MC: V(A) = V(A) + 1 * \big(G(A) - V(A)\big) = 0$$

- A에서 B로 100%의 확률로 전이되므로 V(A)=V(B) 임을 알 수 있지만 MC는 하나의 에피소드 단위로 만 학습되기 때문에 이를 반영하지 못함

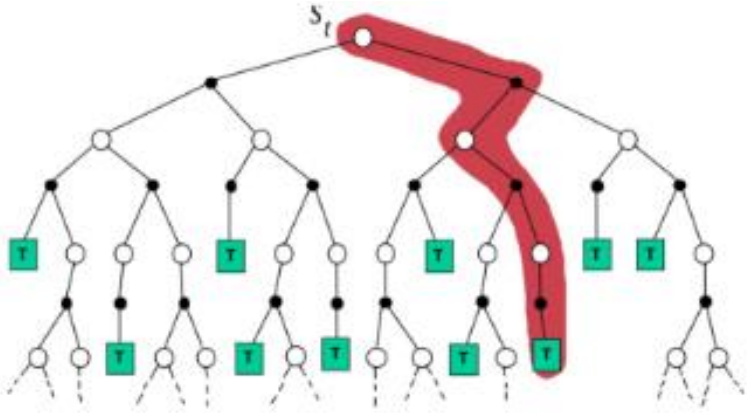
DQN vs REINFORCE

❖ MC Method vs TD Learning

- MC Method는 Episode 단위로 학습 → High Variance, Low Bias
- TD Learning은 Episode내의 1-Step 단위로 학습 → Low Variance, high Bias

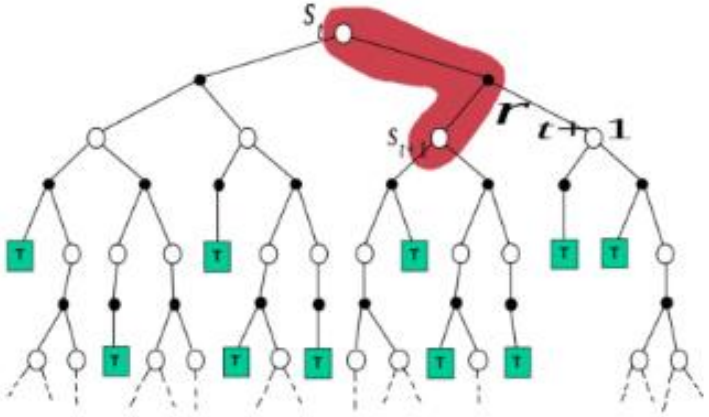
Monte Carlo Method

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)]$$



Temporal Difference Learning

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

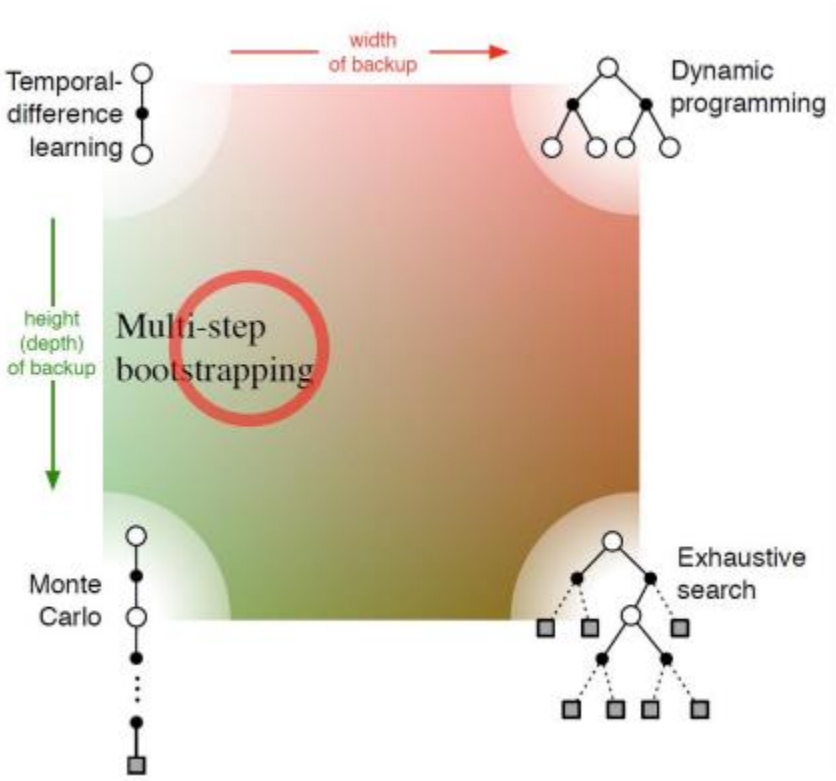


출처 : https://www.slideshare.net/carpedm20/reinforcement-learning-an-introduction-64037079

DQN vs REINFORCE

❖ Multi-Step Bootstrapping

- 1-Step TD Learning과 Episode단위로 학습하는 MC Method의 중간지점
- 최선의 지점

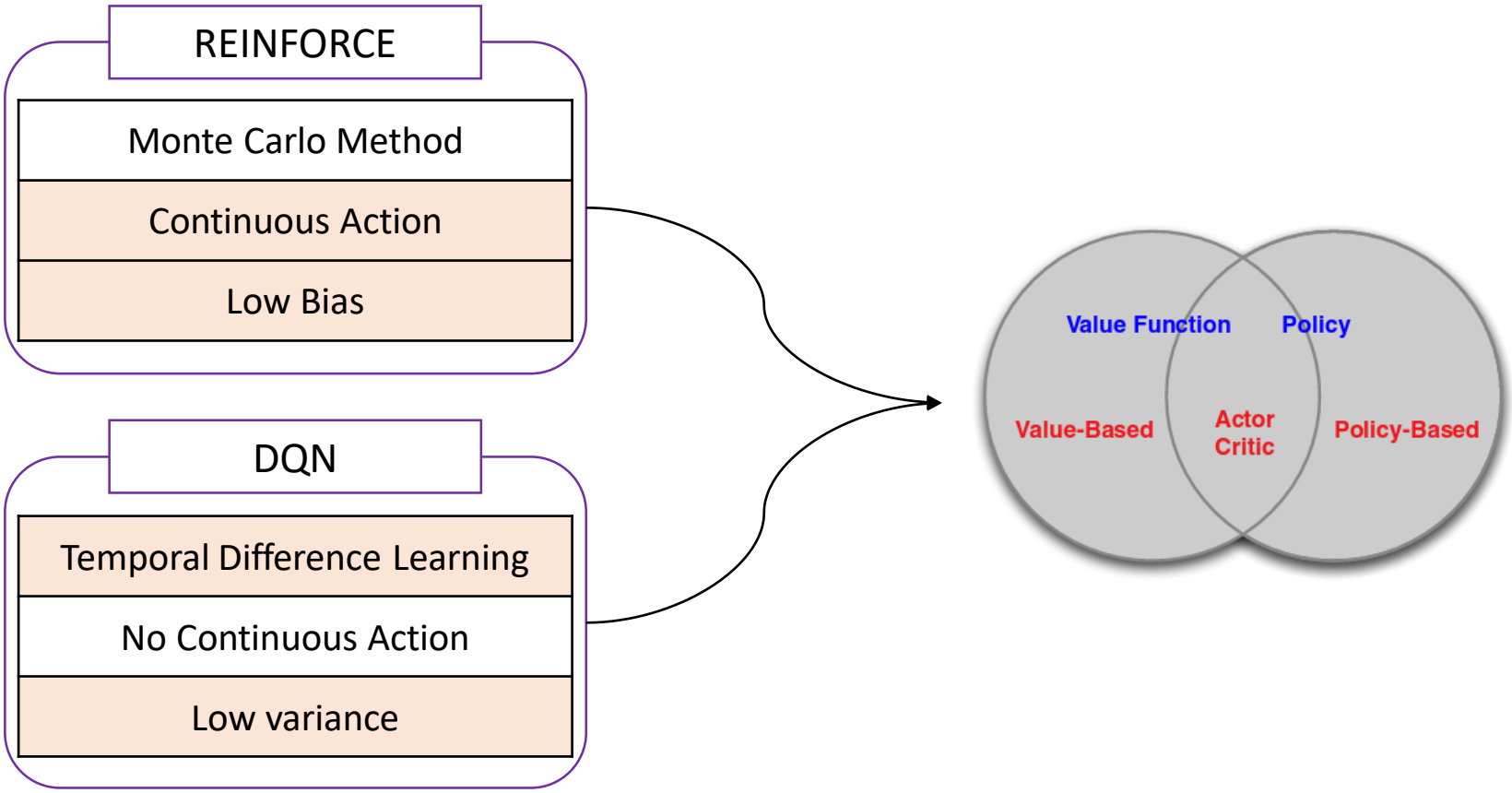


DMQA

Actor-Critic Method

❖ Actor-Critic

- REINFORCE의 장점과 DQN 장점의 조합

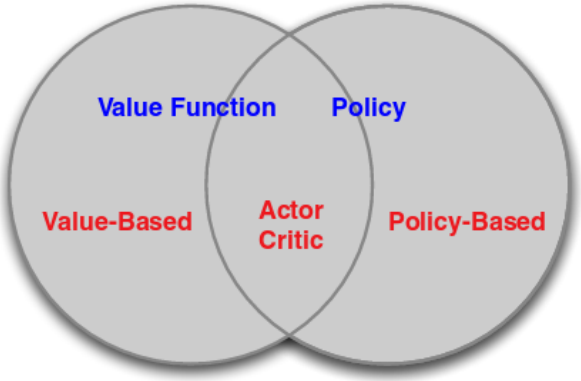


DMQA

Actor-Critic Method

❖ Actor-Critic

- Value-Based & Policy-Based
- Actor : Policy Network & Critic : Value Network



$$E_\tau[\nabla_\theta \sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t)G_t] \approx \sum_{t=0}^{T-1} E_{s_o,a_0,\ldots,s_t,a_t}[\nabla_\theta log\pi_\theta(a_t|s_t)]E_{r_{t+1},s_{t+1},\ldots,s_T,r_T}[G_t]$$

$$= \sum_{t=0}^{T-1} E_{s_o,a_0,\ldots,s_t,a_t}[\nabla_\theta log\pi_\theta(a_t|s_t)]Q_{\pi_\theta}(s_t,a_t)$$

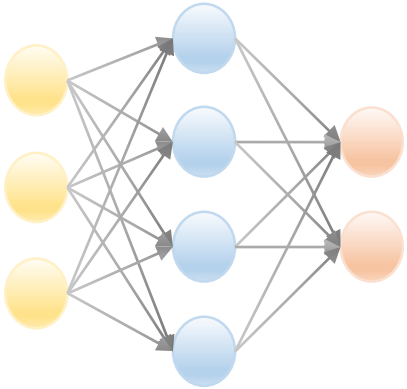
DMQA

Actor-Critic Method

❖ Actor-Critic

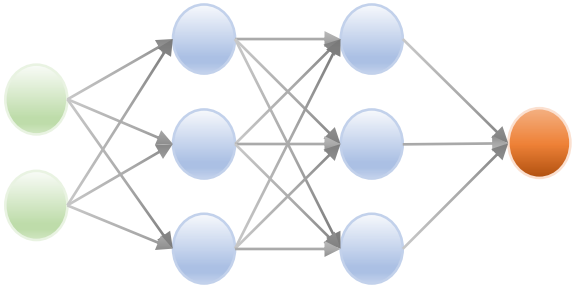
- Actor : Policy Network & Critic : Value Network
- $Q(s_t, a_t) \sim Q_v(s_t, a_t)$ 이용

Actor



- Policy Network에서 Action을 선택

Critic



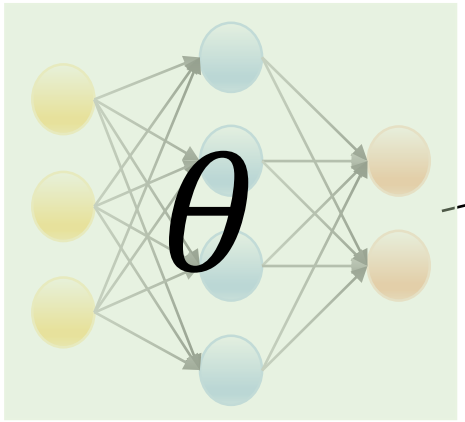
- 선택된 Action을 평가하는 역할

Actor-Critic Method

❖ Actor-Critic

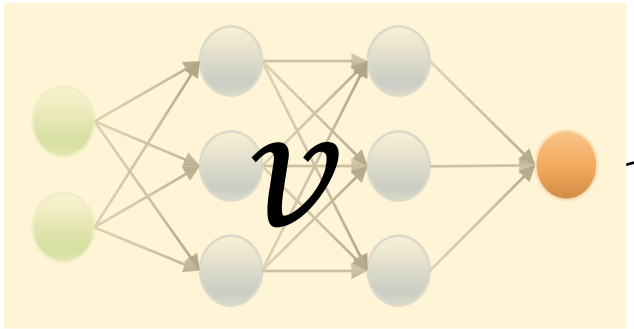
- Actor : Policy Network & Critic : Value Network
- $Q(s_t, a_t) \sim Q_v(s_t, a_t)$ 이용

Actor



$$\theta \leftarrow \theta + \alpha \nabla_\theta log \pi_\theta(a_t|s_t)\big(Q_v(s_t, a_t)\big)$$

Critic



$$v \leftarrow v + \beta\big(r_{t+1} + \gamma\, Q_v(s_{t+1}, a_{t+1}) - Q_v(s_t, a_t)\big)\nabla_v Q_v(s_t, a_t)$$

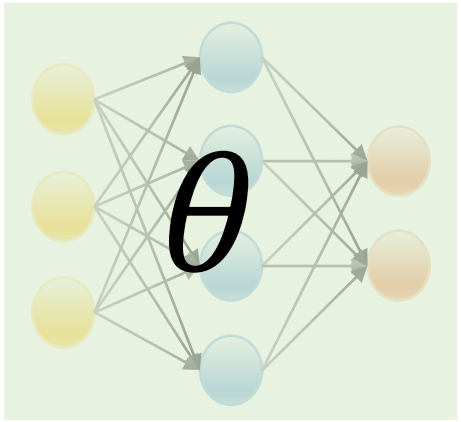
DMQA

Actor-Critic Method

❖ Actor-Critic

- Actor : Policy Network & Critic : Value Network
- $Q(s_t, a_t) \sim Q_v(s_t, a_t)$ 이용

Actor

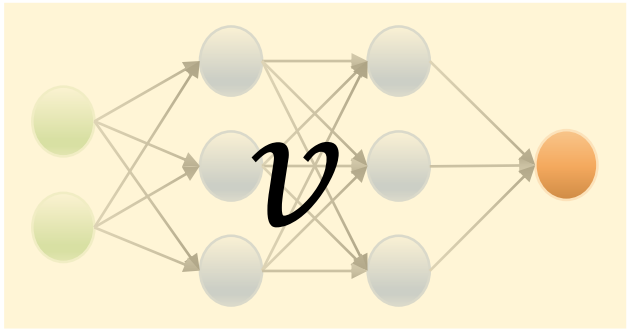


$$\theta \leftarrow \theta + \nabla_\theta log\pi_\theta(a_t|s_t)\left(r_{t+1} + \gamma Q_v(s_{t+1}, a_{t+1}) - Q_v(s_t, a_t)\right)$$

score

- Critic에서 파라미터 v를 업데이트
- Critic이 제안하는 방향으로 θ를 업데이트

Critic

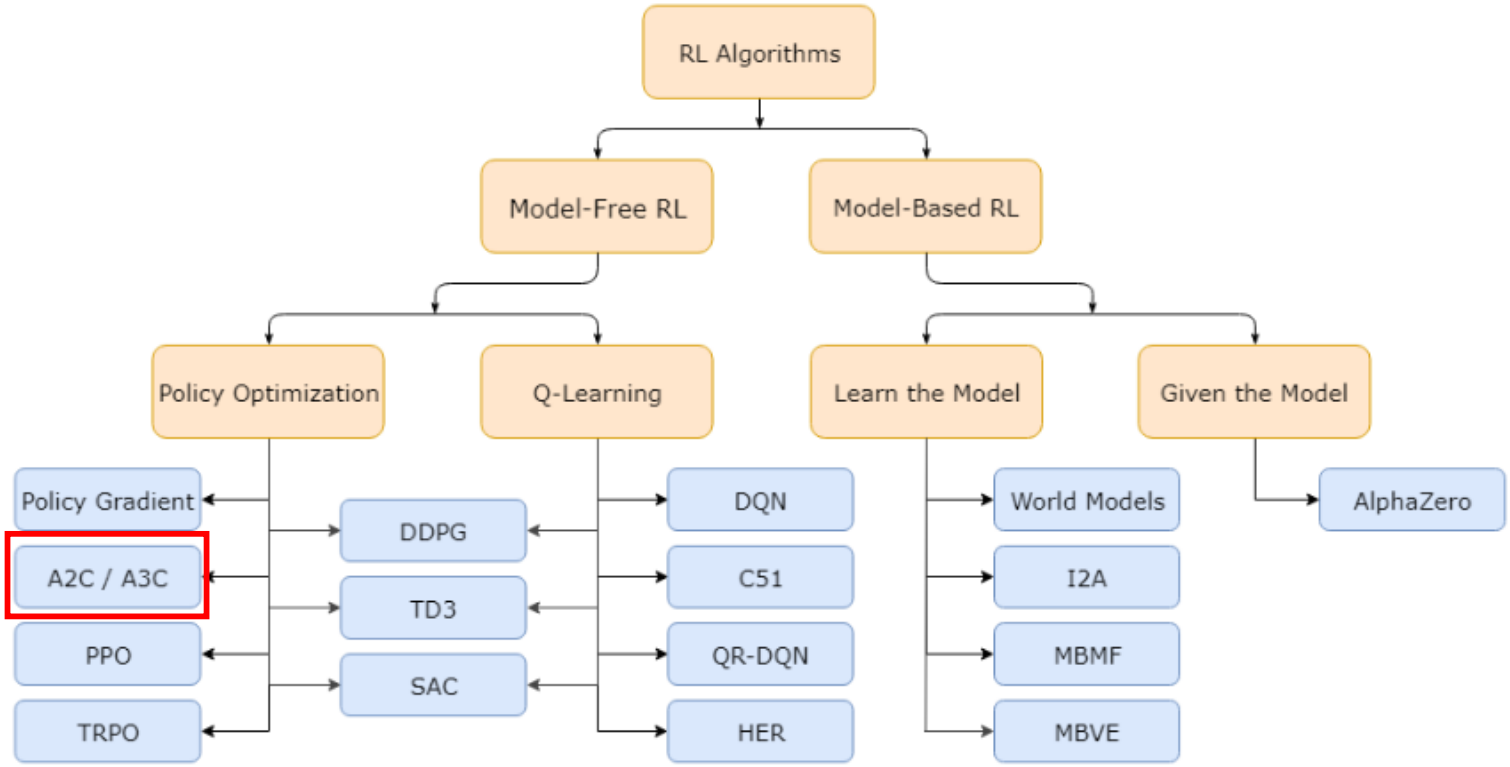


DMQA

목차

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)



출처 : https://mclearninglab.tistory.com/entry/OpenAI-Spinning-Up

DMQA

A3C

❖ 논문

- Google DeepMind

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners have a stabilizing effect on training allowing all four methods to successfully train neural network controllers. The best performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core CPU instead of a GPU. Furthermore, we show that asynchronous actor-critic succeeds on a wide variety of continuous motor control problems as well as on a new task of navigating random 3D mazes using a visual input.

line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to off-policy reinforcement learning algorithms.

Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

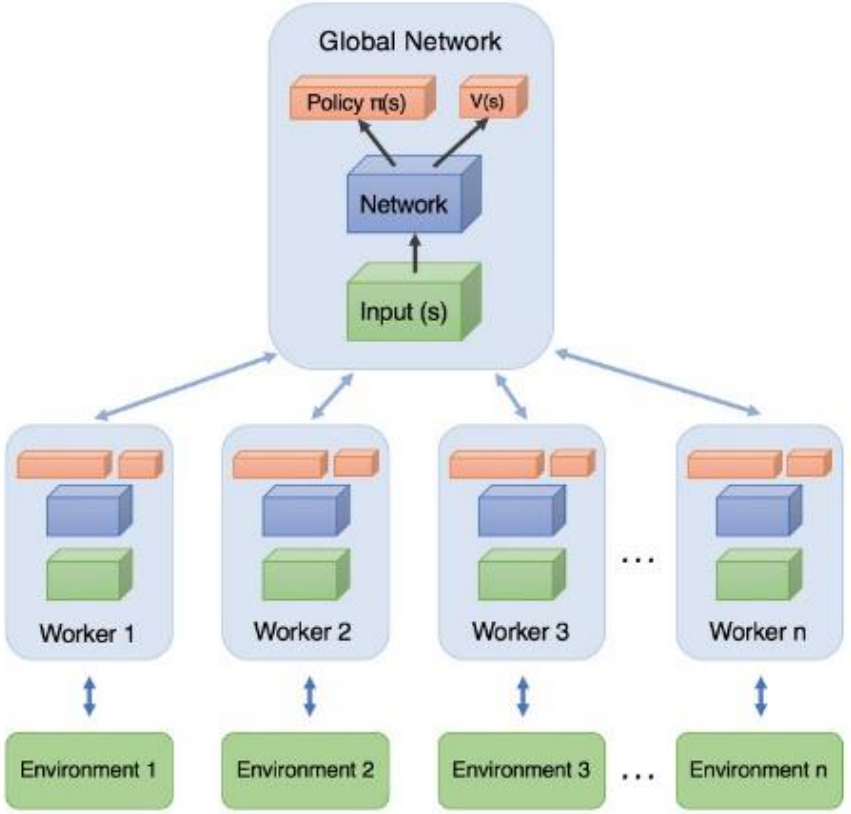
In this paper we provide a very different paradigm for deep reinforcement learning. Instead of experience replay, we asynchronously execute multiple agents in parallel, on multiple instances of the environment. This parallelism also decorrelates the agents' data into a more stationary process,

DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- 다수의 Actor-Critic
- Advantage : $A = Q(s,a) - V(s)$
- Multi-Core CPU



DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- Advantage : $A = Q(s,a) - V(s)$
- Advantage Estimate : $A = R - V(s)$, where $\mathrm{R} = \mathrm{r} + \gamma V(s')$

Actor-Critic

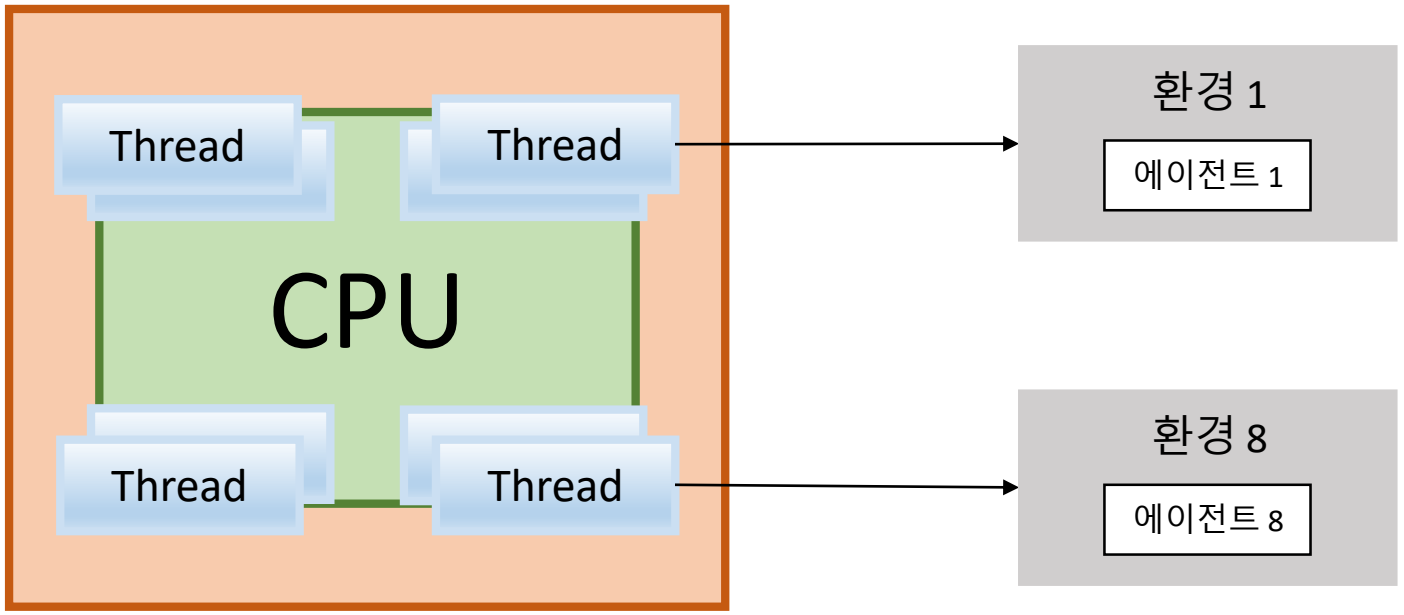
$$\theta \leftarrow \theta + \nabla_\theta log\pi_\theta(a_t|s_t)\left(r_{t+1} + \gamma Q_v(s_{t+1}, a_{t+1}) - Q_v(s_t, a_t)\right)$$

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

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- Multi-Core CPU
- Agent의 최대 개수 = Thread 수

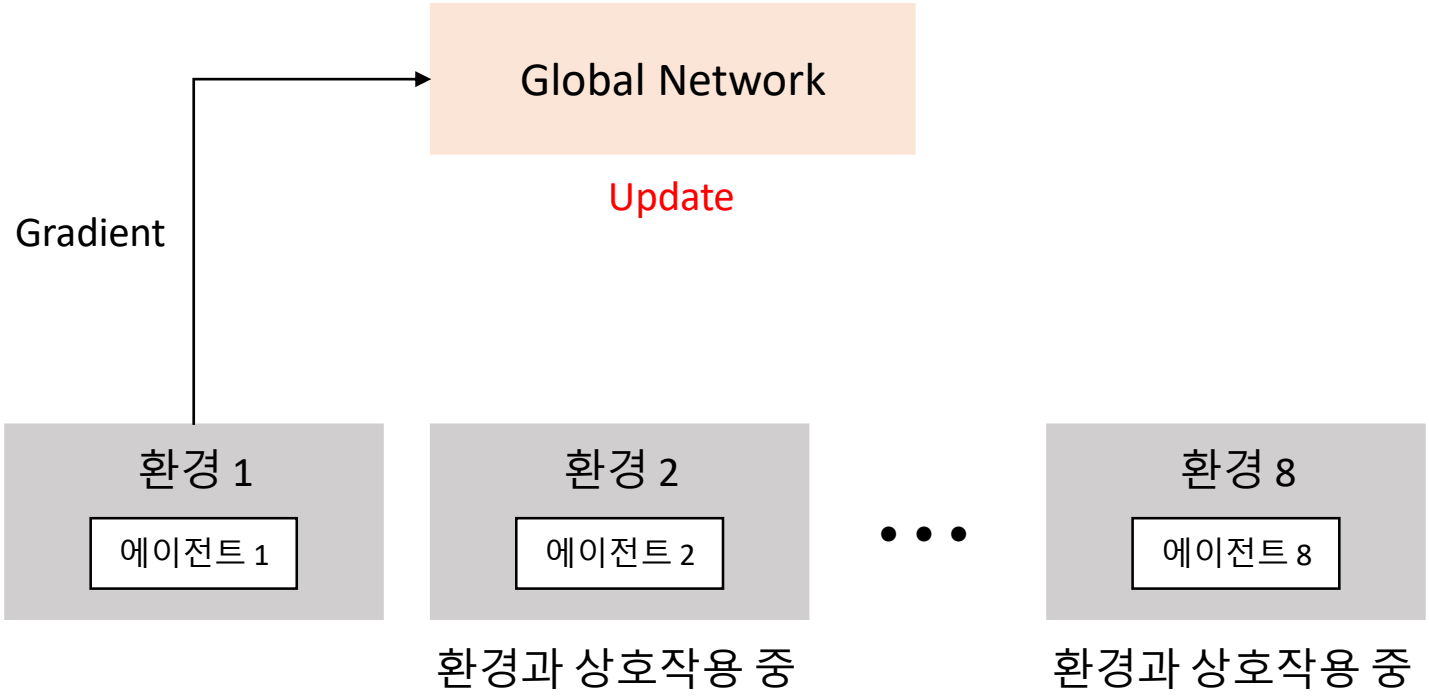


DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- Global Network를 업데이트
- Global Network로 에이전트의 신경망 업데이트

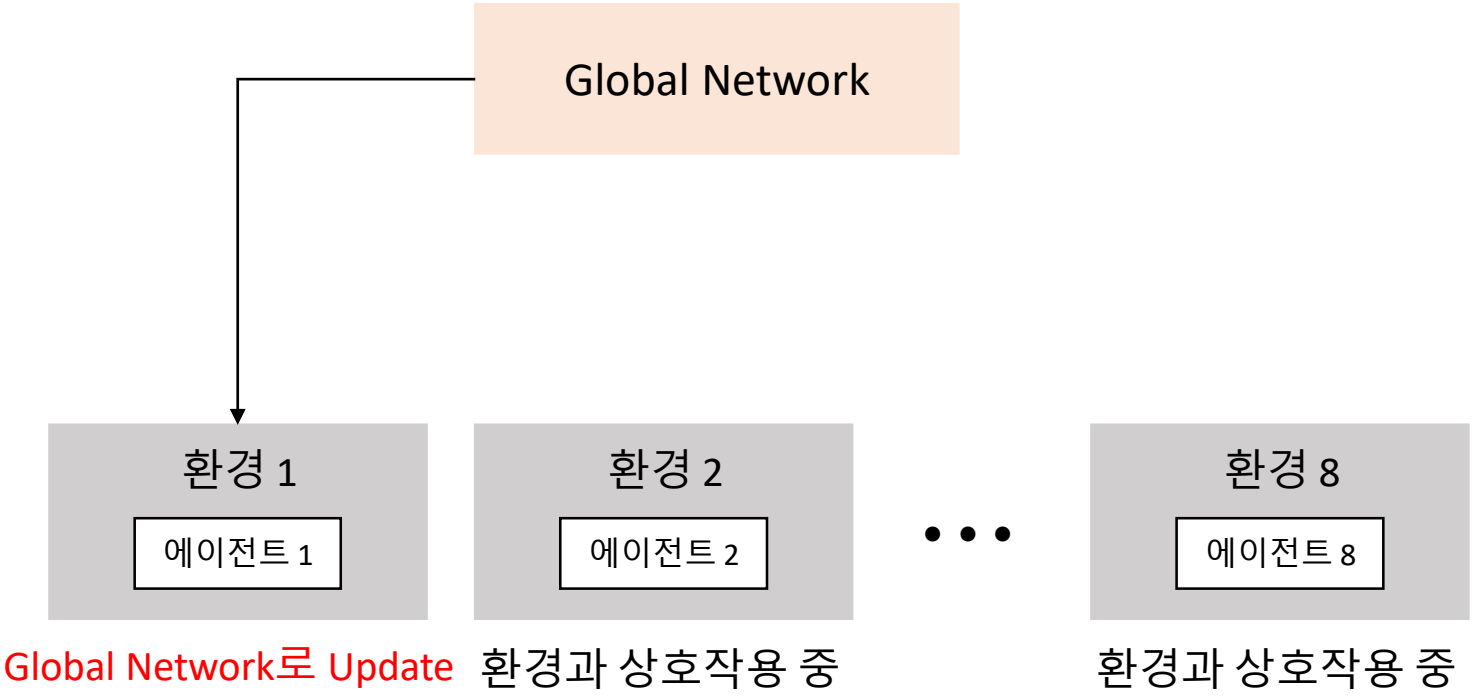


DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- Global Network를 업데이트
- Global Network로 에이전트의 신경망 업데이트

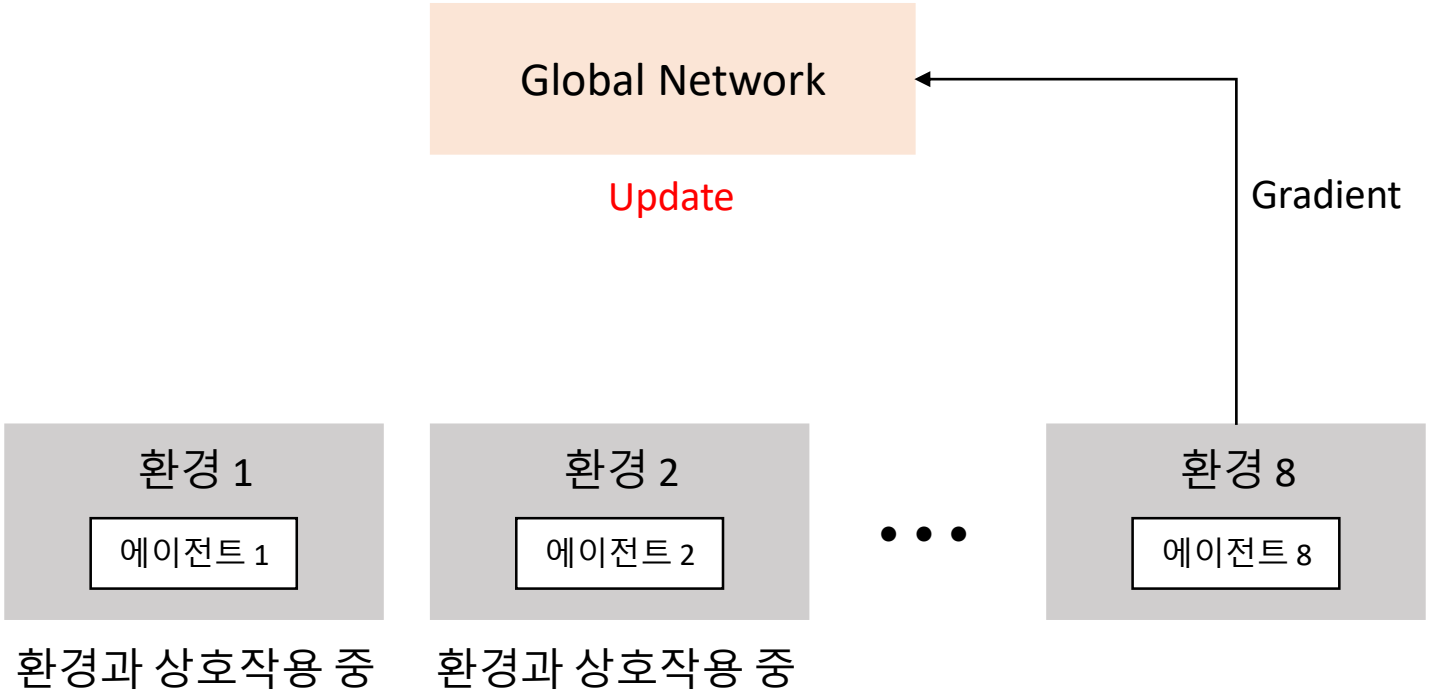


DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- Global Network를 업데이트
- Global Network로 에이전트의 신경망 업데이트

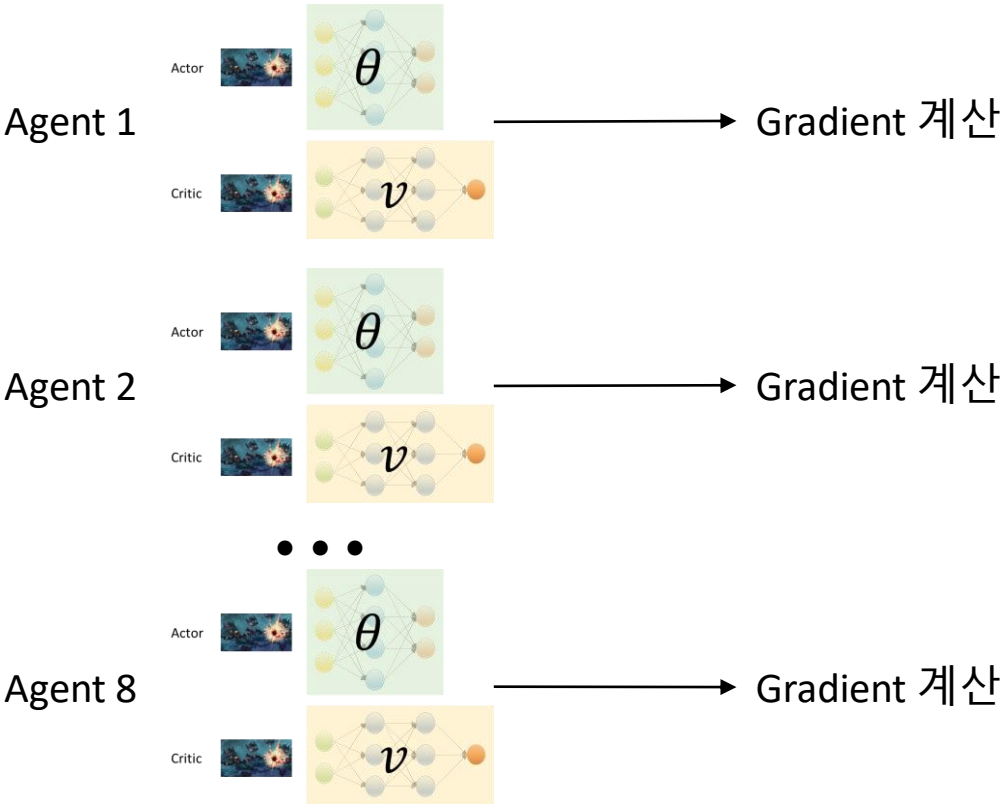


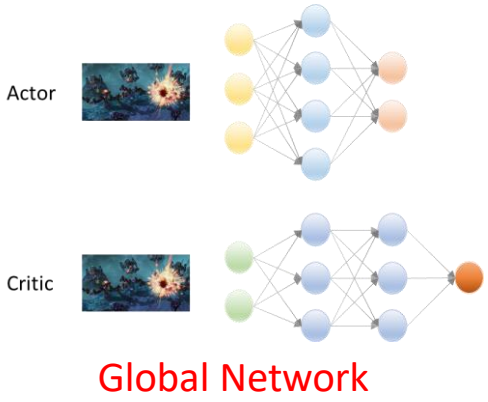
DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- 다양한 환경에서 다수 에이전트 활용
- 에이전트의 독립적 경험 & 다양한 경험 학습 & No Replay Memory

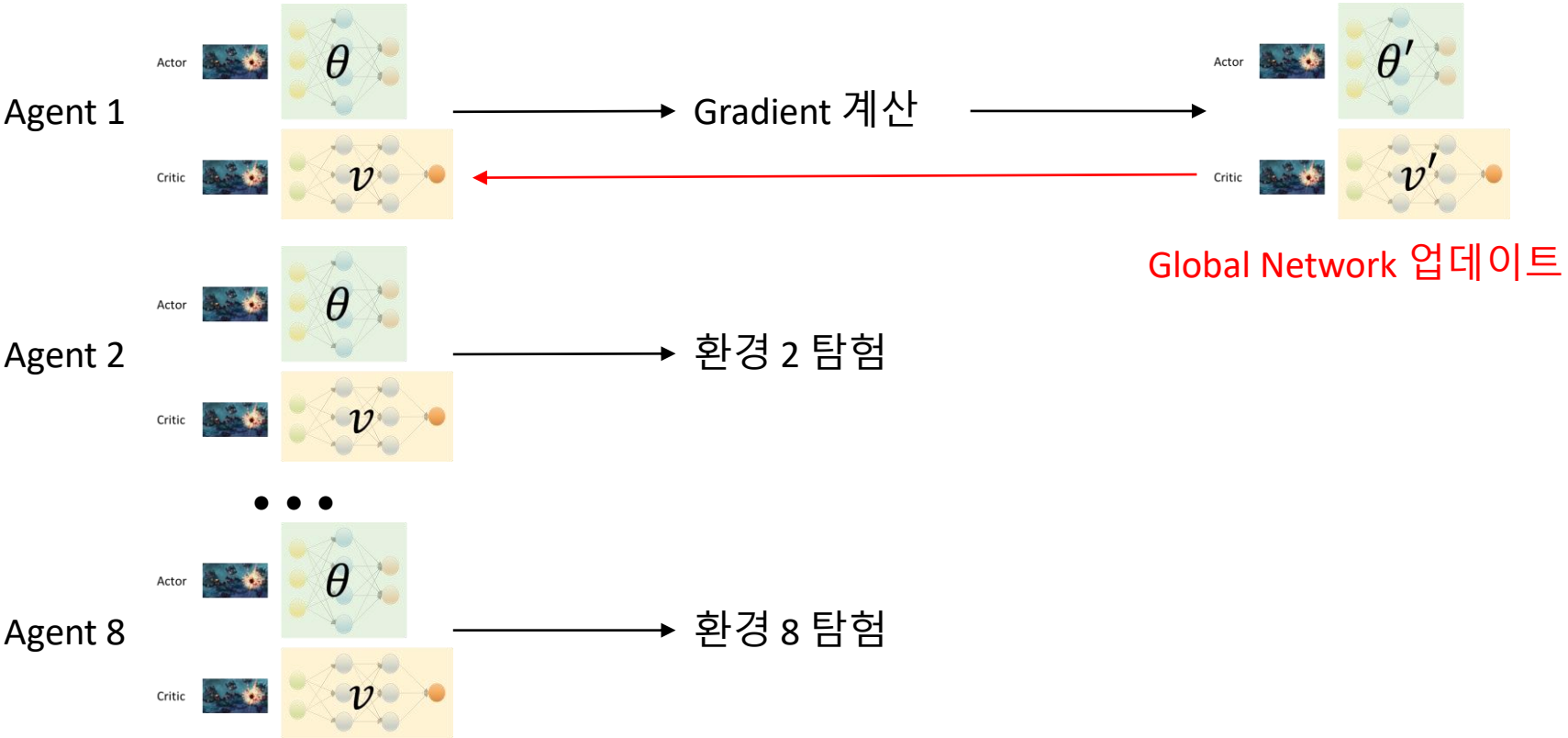




A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

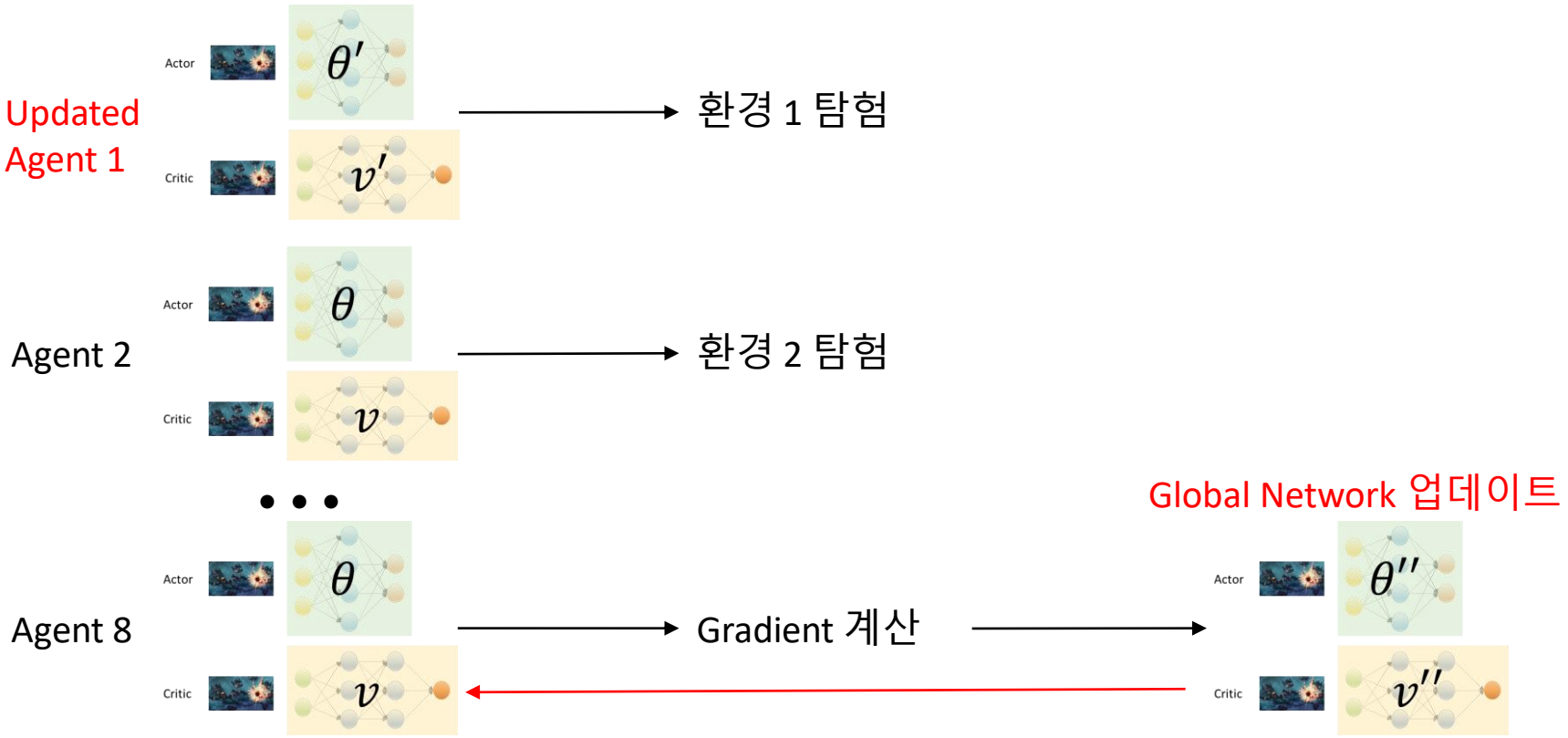
- 다양한 환경에서 다수 에이전트 활용
- 에이전트의 독립적 경험 & 다양한 경험 학습 & No Replay Memory



A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- 다양한 환경에서 다수 에이전트 활용
- 에이전트의 독립적 경험 & 다양한 경험 학습 & No Replay Memory

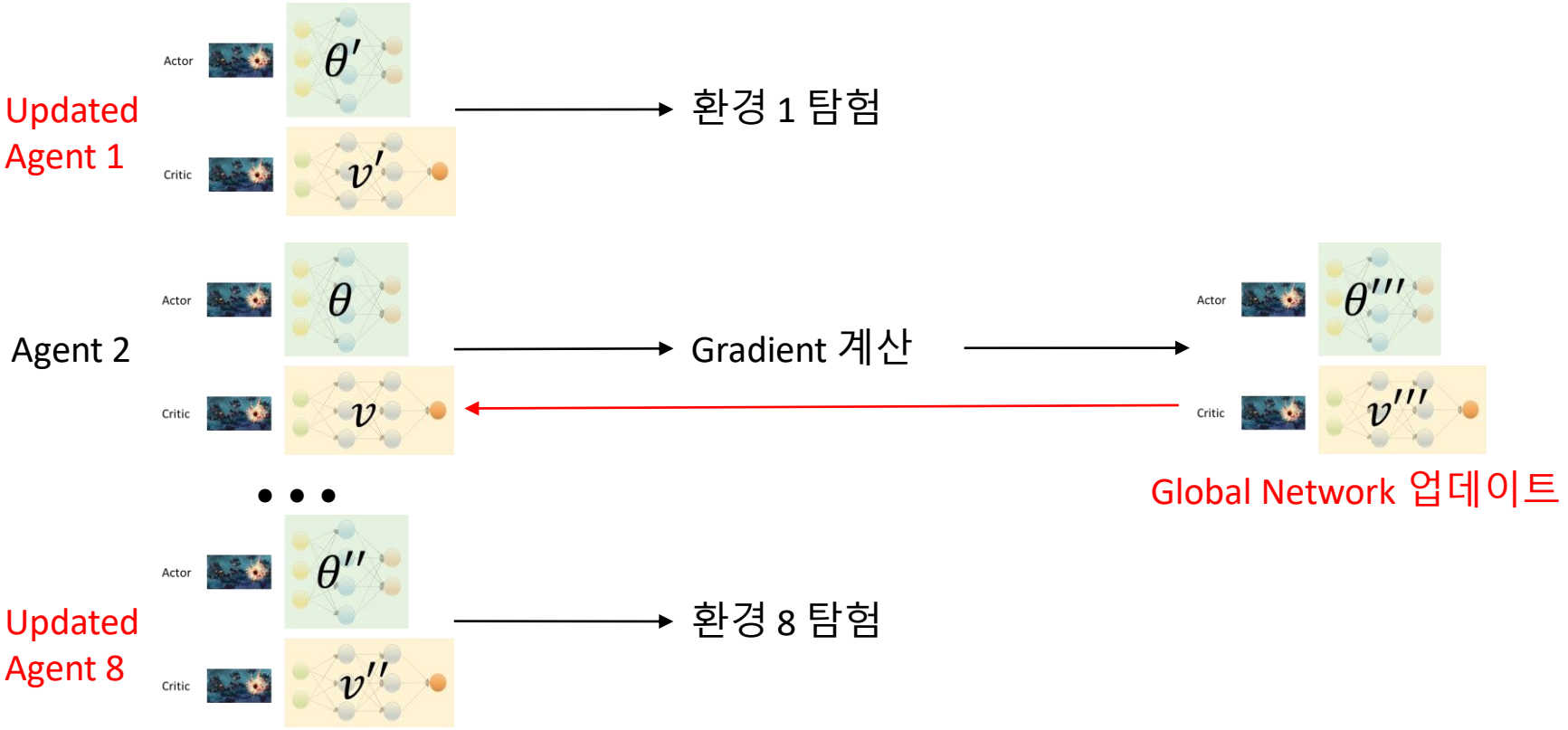


DMQA

A3C

❖ Asynchronous Advantage Actor-Critic(A3C)

- 다양한 환경에서 다수 에이전트 활용
- 에이전트의 독립적 경험 & 다양한 경험 학습 & No Replay Memory



DMQA

A3C

❖ Results

- 5 Atari Games

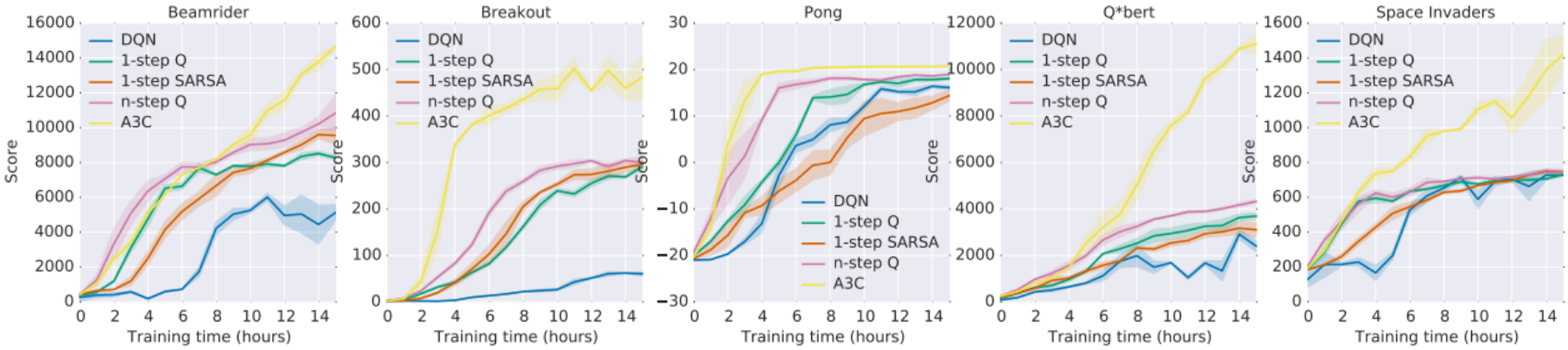


Figure 1. Learning speed comparison for DQN and the new asynchronous algorithms on five Atari 2600 games. DQN was trained on a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores. The plots are averaged over 5 runs. In the case of DQN the runs were for different seeds 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.

DMQA

A3C

❖ Results

- 57 Atari Games

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table SS3 shows the raw scores for all games.

목차

DMQA

Conclusion

❖ Conclusions

- DQN의 단점을 보완하기 위해 고안된 알고리즘
- A3C는 수 많은 상태, 행동이 존재하는 게임 환경에 적합
- 다양한 에이전트를 활용하여 비동기적 업데이트 진행
- 하나의 에이전트인 DQN보다 단축된 시간과 학습 성능이 뛰어남

Q & A

감사합니다

Appendix

❖ 미래에 받을 보상의 합

- γ : 미래에 받을 보상에 대한 디스카운트 요소(발산 방지)

$$v(s_t) = E[r_{t+1}|s_t]$$

$$v(s_t) = E[r_{t+1} + \gamma v(s_{t+1})|s_t]$$

DMQA

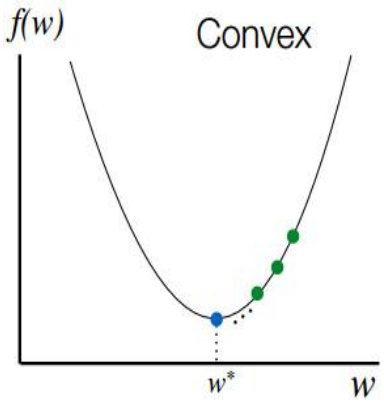
Appendix

❖ Fixed Q-Target

- 지도학습 vs 강화학습

지도학습

Training Data	(x, y)
Model	$\hat{y} = f(x)$
Objective Function	$Loss = E[(y_i - \hat{y}_i)^2$

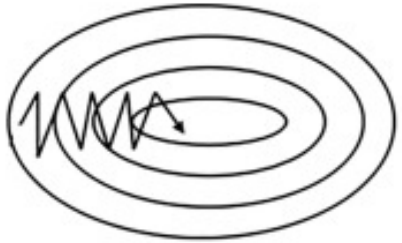


- 고정된 Target y 값
- 수렴 보장(O)

강화학습

Training Data	S, A, R, S, A, R, …
Model	$\pi(a\|s)$
Objective Function	

$$Loss = E[(r + \gamma \max_{a'} Q(s', a', w^{-}) - Q(s, a, w_i))^2]$$



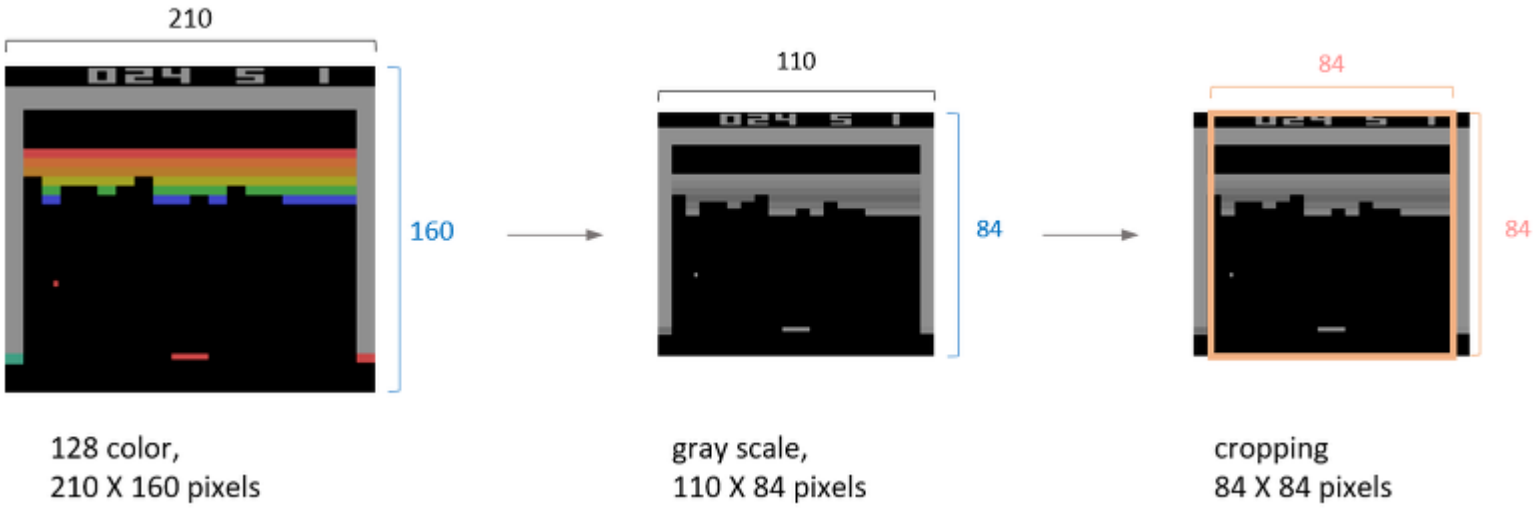
- 학습 시 Target y값이 움직임 (Oscillation 현상)
- 수렴 보장(X)

DMQA

Appendix

❖ Preprocessing(DQN)

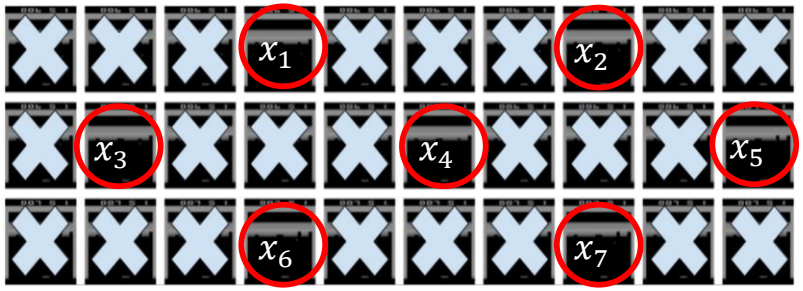
- Atari 게임에서 down-sizing과 gray scale함
- 정사각형 이미지로 변환 후, 4개의 프레임을 쌓음(RGB 채널 개념)



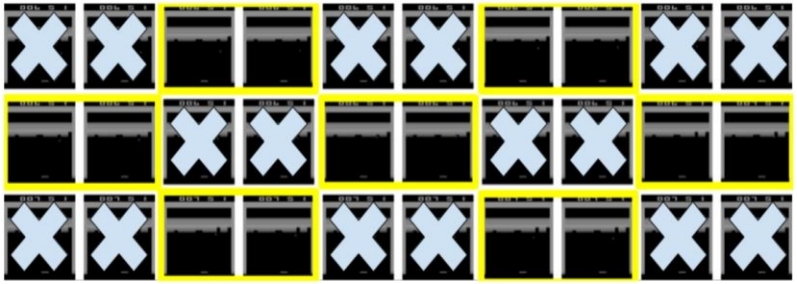
Appendix

❖ Preprocessing(DQN)

- Atari 게임에서 down-sizing과 gray scale함
- 정사각형 이미지로 변환 후, 4개의 프레임을 쌓음(RGB 채널 개념)



K번째 위치한 이미지만 선택



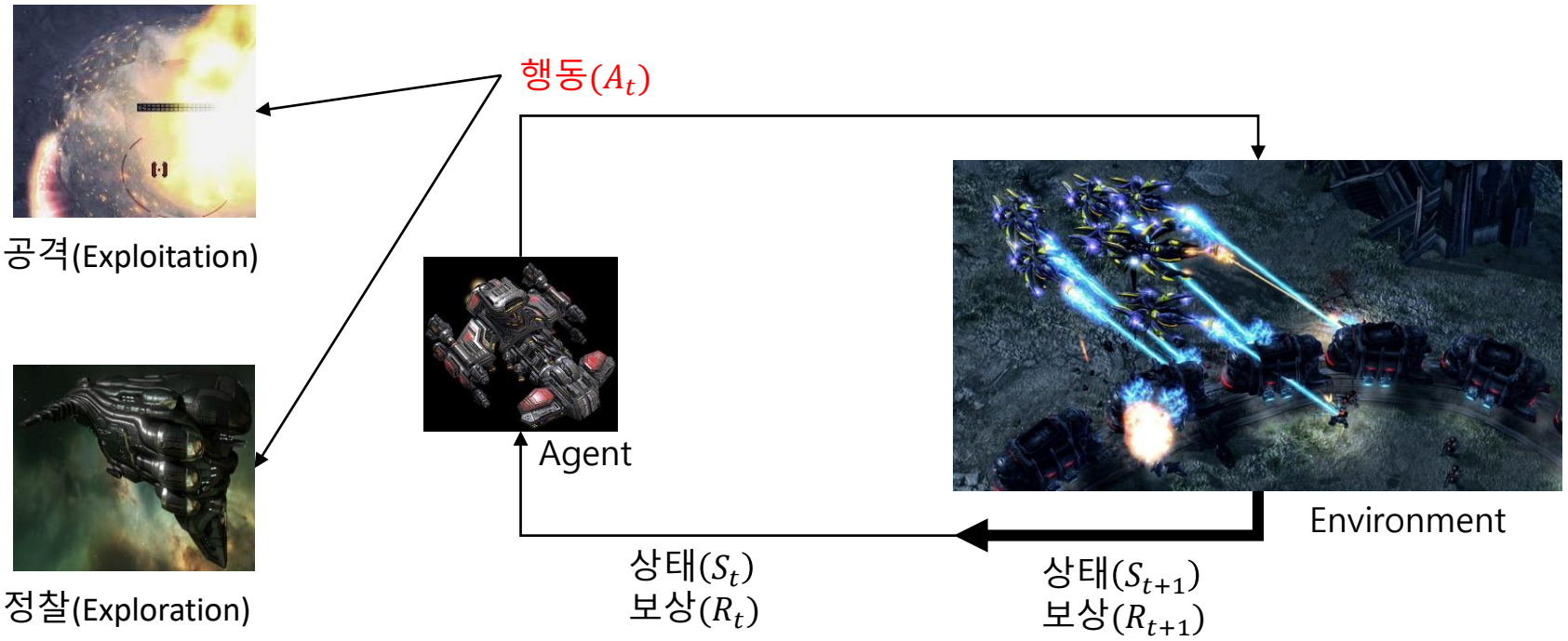
선택된 이미지 앞의 이미지와 pixel-wise maximum

$$s_1 = (x_1, x_2, x_3, x_4)$$
$$s_2 = (x_2, x_3, x_4, x_5)$$
$$\vdots$$

Appendix

❖ 탐험(Exploration)과 착취(Exploitation)

- 더 높은 보상을 받기 위해서는 주어진 상황에서 더 적절한 행동을 선택(Exploitation)
- 각 행동들의 가치에 대해 알기 위해서는 사전에 탐험이 필요(Exploration)



DMQA

Appendix

❖ On Policy

- 모델 학습을 하기 위해 Target Policy의 결과들을 직접 사용

❖ Off Policy

- Target Policy와 Behavior Policy로 나뉨
- Target Policy는 더 높은 보상을 받기 위한 행동을 학습(Exploitation)
- Behavior Policy는 행동의 가치를 탐색(Exploration)

Appendix

❖ 수식 유도

- $J(\theta) = E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = E[r_1 + r_2 + r_3 + \cdots + r_\tau|\pi_\theta]$
- $\theta' = \theta + \alpha\nabla_\theta J(\theta)$, $\nabla_\theta J(\theta)$ = Policy Gradient

$$\nabla_\theta E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = \nabla_\theta \sum_{t=0}^{T-1} P(s_t, a_t|\tau) r_{t+1}$$

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

미분 불가능

Appendix

❖ 수식 유도

- $J(\theta) = E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = E[r_1 + r_2 + r_3 + \cdots + r_\tau|\pi_\theta]$
- $\theta' = \theta + \alpha\nabla_\theta J(\theta)$, $\nabla_\theta J(\theta)$ = Policy Gradient

$$\nabla_\theta E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = \nabla_\theta \sum_{t=0}^{T-1} P(s_t, a_t|\tau) r_{t+1}$$

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

미분 가능

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

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

$$= E_\tau[\sum_{t=0}^{T-1} \nabla_\theta log P(s_t, a_t|\tau) r_{t+1}]$$

Appendix

❖ 수식 유도

- $P(s_t, a_t|\tau) = P(s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_t, a_t|\theta)$
- $P(s_0)\pi_\theta(a_0|s_0)P(s_1|s_0, a_0)\pi_\theta(a_1|s_1)P(s_2|s_1, a_1) \dots$
- $logAB = logA + logB$

$$P(\tau|\theta) = P(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t, a_t)$$

$$logP(\tau|\theta) = logP(s_0) + \sum_{t=0}^{T-1}[log\pi_\theta(a_t|s_t) + logP(s_{t+1}|s_t, a_t)]$$

θ에 대한 미분 → 상태 전이 확률 미분 시 상수 제거

Appendix

❖ 수식 유도

- $P(s_t, a_t|\tau) = P(s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_t, a_t|\theta)$
- $P(s_0)\pi_\theta(a_0|s_0)P(s_1|s_0, a_0)\pi_\theta(a_1|s_1)P(s_2|s_1, a_1) \dots$
- $logAB = logA + logB$

$$P(\tau|\theta) = P(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t, a_t)$$

$$logP(\tau|\theta) = logP(s_0) + \sum_{t=0}^{T-1}[log\pi_\theta(a_t|s_t) + logP(s_{t+1}|s_t, a_t)]$$

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

Appendix

❖ 수식 유도

- $P(s_t, a_t|\tau) = P(s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_t, a_t|\theta)$
- $P(s_0)\pi_\theta(a_0|s_0)P(s_1|s_0, a_0)\pi_\theta(a_1|s_1)P(s_2|s_1, a_1)\dots$
- $logAB = logA + logB$

$$\nabla_\theta logP(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t) \xrightarrow{\text{대입}} E_\tau[\sum_{t=0}^{T-1} \nabla_\theta logP(s_t, a_t|\tau) r_{t+1}]$$

$$\nabla_\theta E[\sum_{t=0}^{T-1} r_{t+1}|\pi_\theta] = E_\tau \nabla_\theta [\sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t) r_{t+1}]$$

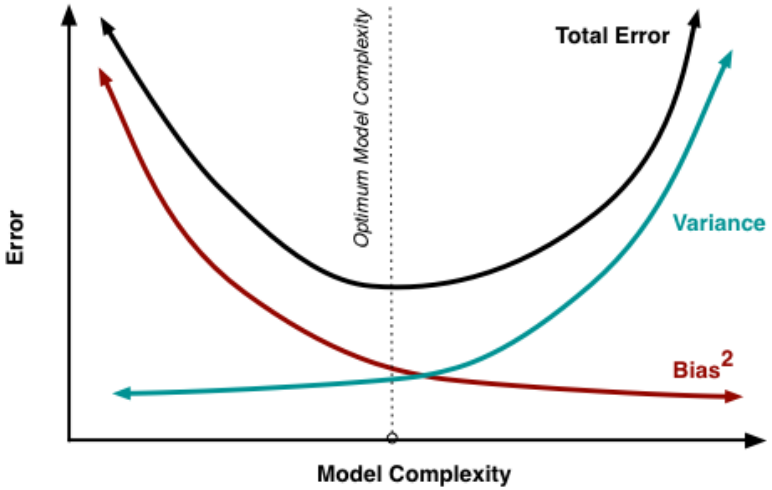
$$\approx E_\tau[\nabla_\theta \sum_{t=0}^{T-1} log\pi_\theta(a_t|s_t) G_t],\ where\ G_t = \sum_{t=0}^{T-1} \gamma^t r_{t+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-1} r_T$$

Discounted G_t → 단순 보상의 합 발산 방지

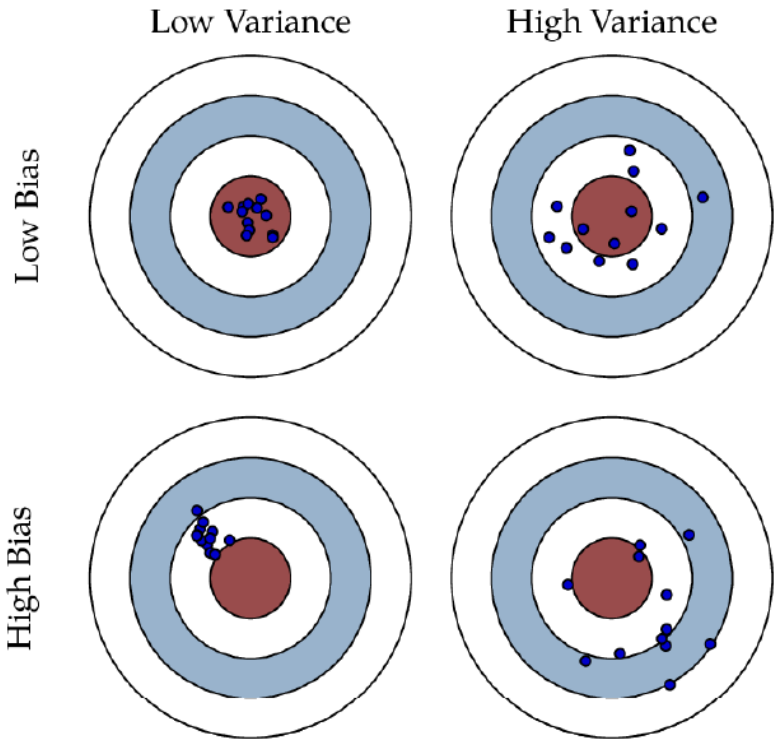
Appendix

❖ Bias and Variance Tradeoff

- Bias : 실제 값과 예측 값의 차이
- Variance : 예측 값과 예측 값들의 평균과의 차이



출처 : http://scott.fortmann-roe.com/docs/BiasVariance.html



Appendix

❖ Asynchronous Advantage Actor-Critic(A3C)

- Pseudo Code

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 $T > T_{max}$