# Q-Learning & DQN

Deep-Learning seminar

AILAB 김건 권명하

# CONTENTS

# 01

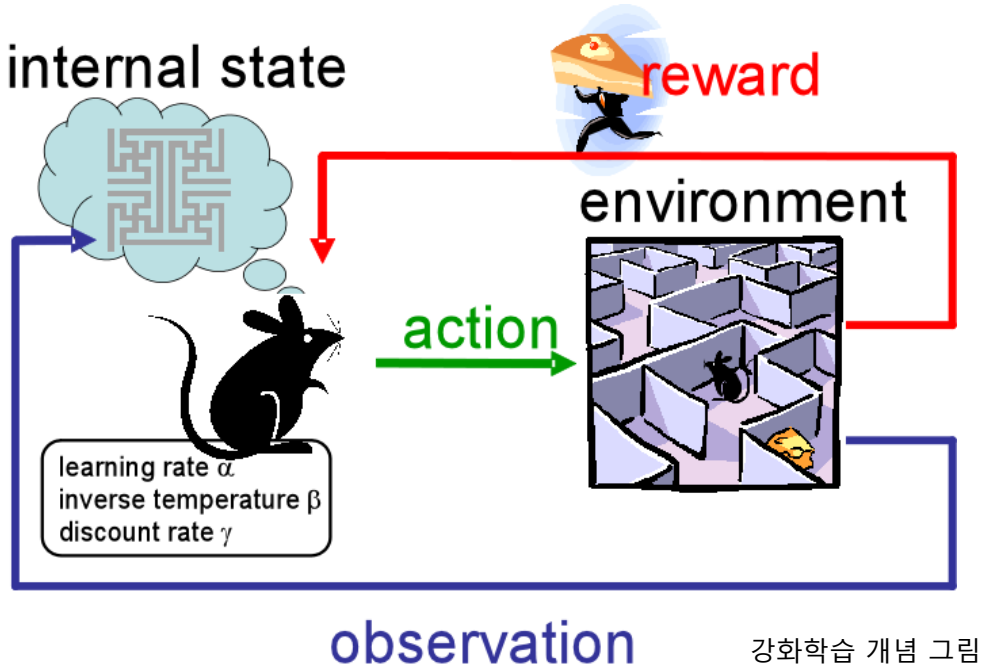
# Q-Learning

# 01. Q-Learning

## Q-learning?

강화 학습이란? 어떤 환경(Environment) 안에서 정의된 에이전트(Actor)가
현재의 상태(State)를 인식하여, 선택 가능한 행동들(Actions) 중
보상(Reward)을 최대화하는 행동 혹은 행동 순서를 선택(Policy)하는 방법이다. (Wikipedia)

강화 학습 기법 중 하나인 Q-러닝은  Q 함수(state-action value function)을 업데이트 해주면서
주어진 환경에서 가장 나은 action을  구하는 알고리즘




강화학습 개념 그림
http://www.cs.utexas.edu/~eladlieb/RLRG.html

# 01. Q-Learning

## Q-function

# Q-function(state-action value function)


**Q (state, action) = r**

state와 action을 넣어주면 reward 값을 주는 함수

# Policy using Q-function

Policy(정책) : 주어진 상태에서 어떤 행동을 수행할지 나타내는 규칙

**Q (state, action) = r**

**Q (s1, LEFT) = 0**

**Q (s1, RIGHT) = 0.5**

**Q (s1, UP) = 0**

**Q (s1, DOWN) = 0.3**

# Policy using Q-function

Policy(정책) : 주어진 상태에서 어떤 행동을 수행할지 나타내는 규칙 ->$\pi$ 로 표현
Q함수를 학습하고 나면 각 상태에서 최고의 Q를 주는 행동을 수행함으로써 최적의 정책을 유도할 수 있다.

**Q (state, action) = r**

**Q (s1, LEFT) = 0**

**Q (s1, RIGHT) = 0.5**

**Q (s1, UP) = 0**

**Q (s1, DOWN) = 0.3**

**보상이 최대인 행동을 선택!**

01. Q-Learning

Q-function

# Policy using Q-function

**Q (state, action) = r**

Optimal Policy : $\pi^*$

$$\textbf{Max Q} = \max_{a`} \boldsymbol{Q(s, a')}$$

$$\boldsymbol{\pi^*(s) = \arg\max_{a} Q(s, a)}$$

# 01. Q-Learning

## Q-function

# Learning Q-function

Q-Learning의 기본 아이디어

- s′이라는 state가 존재한다.
- s라는 state에서 a라는 action을 하면 s′으로 이동하고 보상 r을 받는다.
- 그리고 s′이라는 state에서 할 수 있는 각 행동들에 대해 어떤 보상을 받을지 알고 있을 때

Optimal policy를 찾기 위한 Q-function은 다음과 같이 표현 할 수 있다.

$$\hat{Q}(s,a) \leftarrow r + \max_{a`} \hat{Q}(s',a')$$

$\hat{Q}$ : 중간중간 학습할 때는 최종적인 Q가 아니기에 ^을 씌워서 표현

01. Q-Learning

Q-Table

# Q-Learning with table

Frozen_Lake Game
Q-Learning의 대표 예제

초록색: Start Point
파란색: Hole
노란색: Goal Point

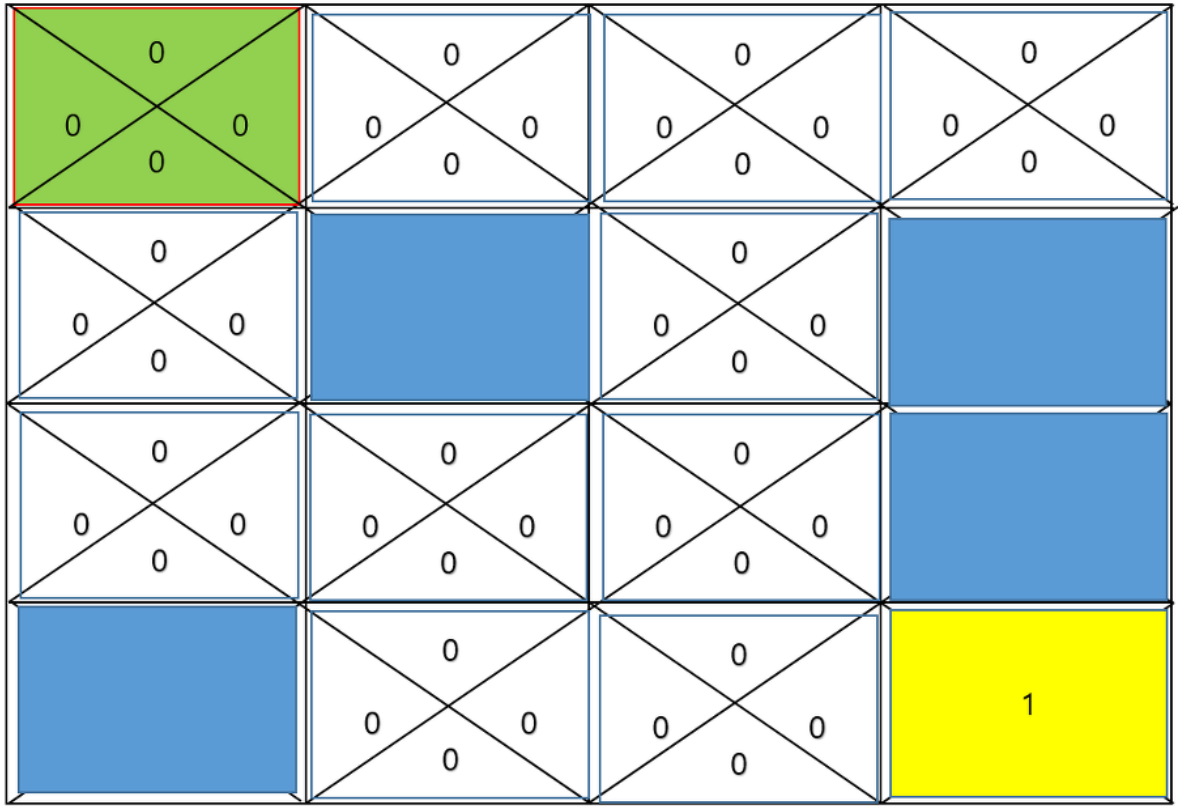
Start Point에서 출발하여 Goal Point에 도달하는 것
이 게임의 목표

Hole에 빠지면 게임이 끝난다.

각 지점에서는 상하좌우로 움직일 수 있다.

Goal Point에 도달하면 Reward를 받음

Q-Learning을 통하여 최적의 경로(정책)을 학습한다.

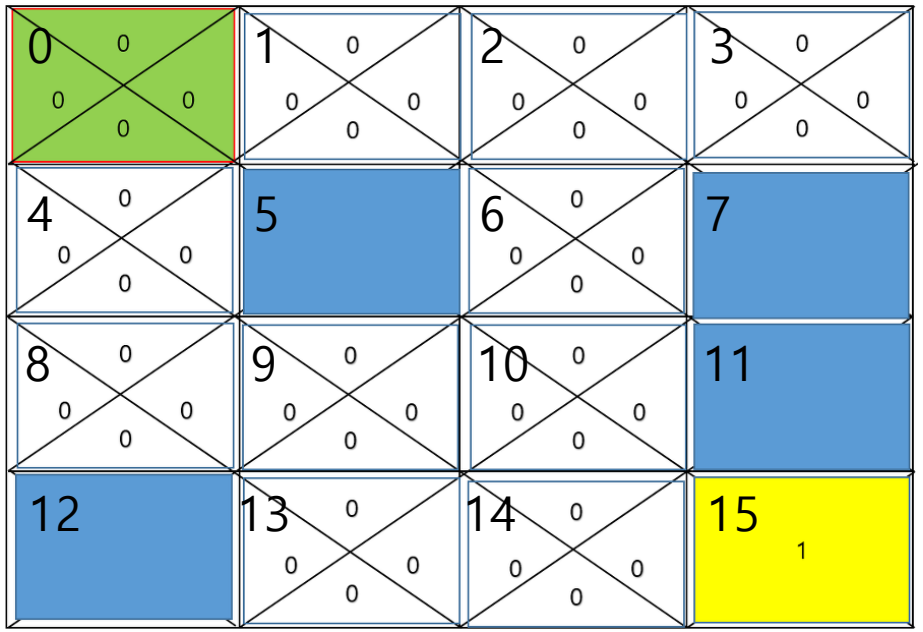


# 01. Q-Learning

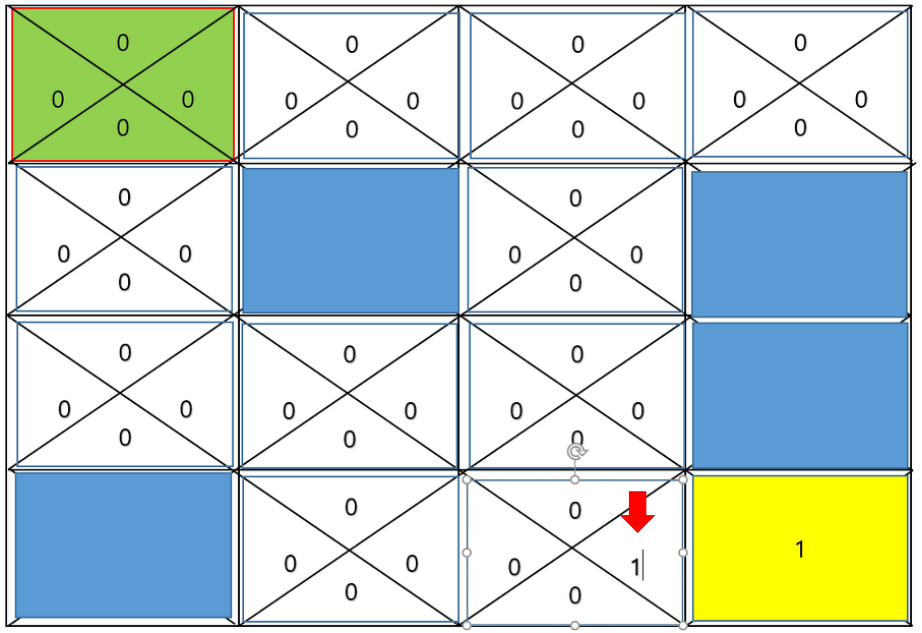
## Q-Table

# Q-Learning with table

$$\widehat{Q}(s,a) \leftarrow r + \max_{a`} \widehat{Q}(s',a')$$




Table을 초기화 한다.




랜덤하게 학습을 진행 하던 중 $S_{14}$에 도착 후 right action을 하면 Q-table이 다음과 같이 업데이트된다.

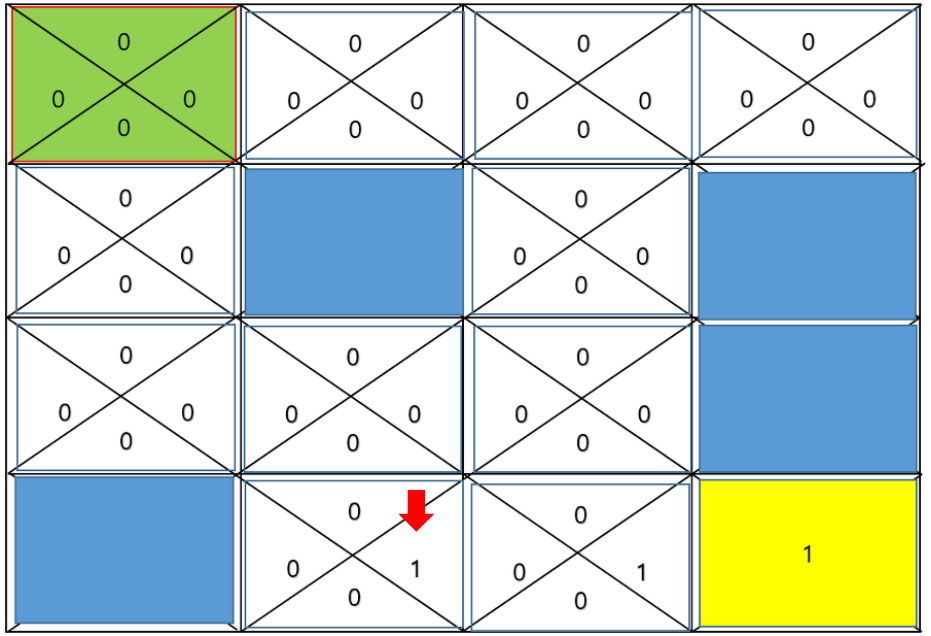
$Q(S_{14}, a_{right})$ = r = 1

# 01. Q-Learning

## Q-Table

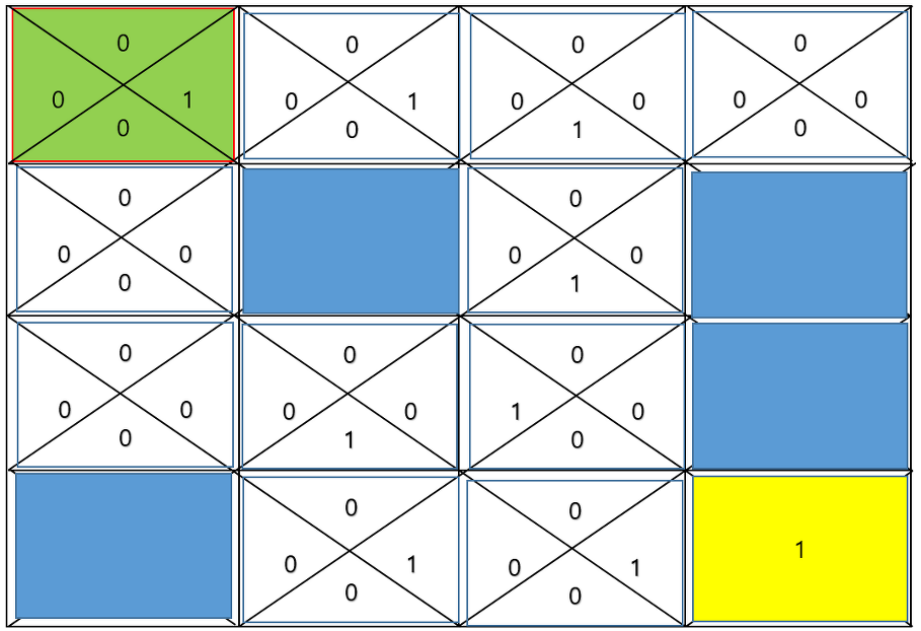
# Q-Learning with table

$$\widehat{Q}(s,a) \leftarrow r + \max_{a`} \widehat{Q}(s',a')$$



$S_{13}$ 에서 right action 을 하면
Q-table이 다음과 같이 업데이트된다.

Q($S_{13}$, $a_{right}$) = r + max( Q($S_{14}$, $a$) ) = 0 + max(0,0,1,0) = 1

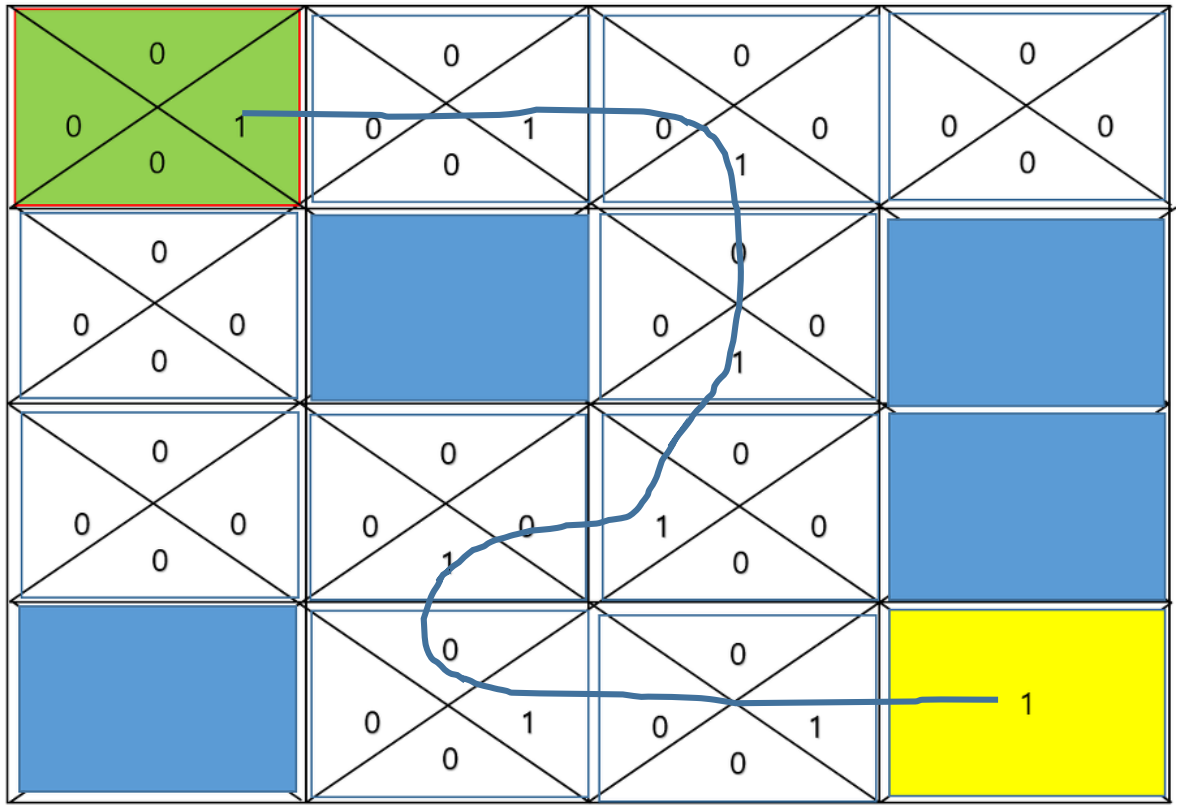


일련의 과정을 거쳐서 Q-table이 위와 같이 학습될 수 있다.

01. Q-Learning

Q-Table

# Exploit VS Exploration



하지만 여기서 문제점이 발생한다.
한번 경로를 찾으면 다른 경로를 찾지 못한다.

여기서 한가지 개념을 더 Q-learning에 추가한다.

Exploit : 이전의 경로를 이용할 것이냐?
Exploration : 안 가본 경로로 가볼 것이냐?

# Exploit VS Exploration

2가지 방법으로 위 개념을 적용함

**E-greedy**

```
e = 0.1
If rand < e:
    a = random
else:
    a = argmax(Q(s,a))
```

**Add random noise**

```
a = argmax(Q(s,a) + random_values)

a = argmax([0.5 0.6 0.3 0.1] + [0.1 0.2 0.7 0.3]
```

**decaying** 개념
학습이 진행될 수록 모험을 덜함

**decaying E-greedy**

```
for i in range(1000)
    e = 0.1 / (i+1)
    If random(1) < e:
        a = random
    else:
    a = argmax(Q(s,a))
```

**decaying random noise**

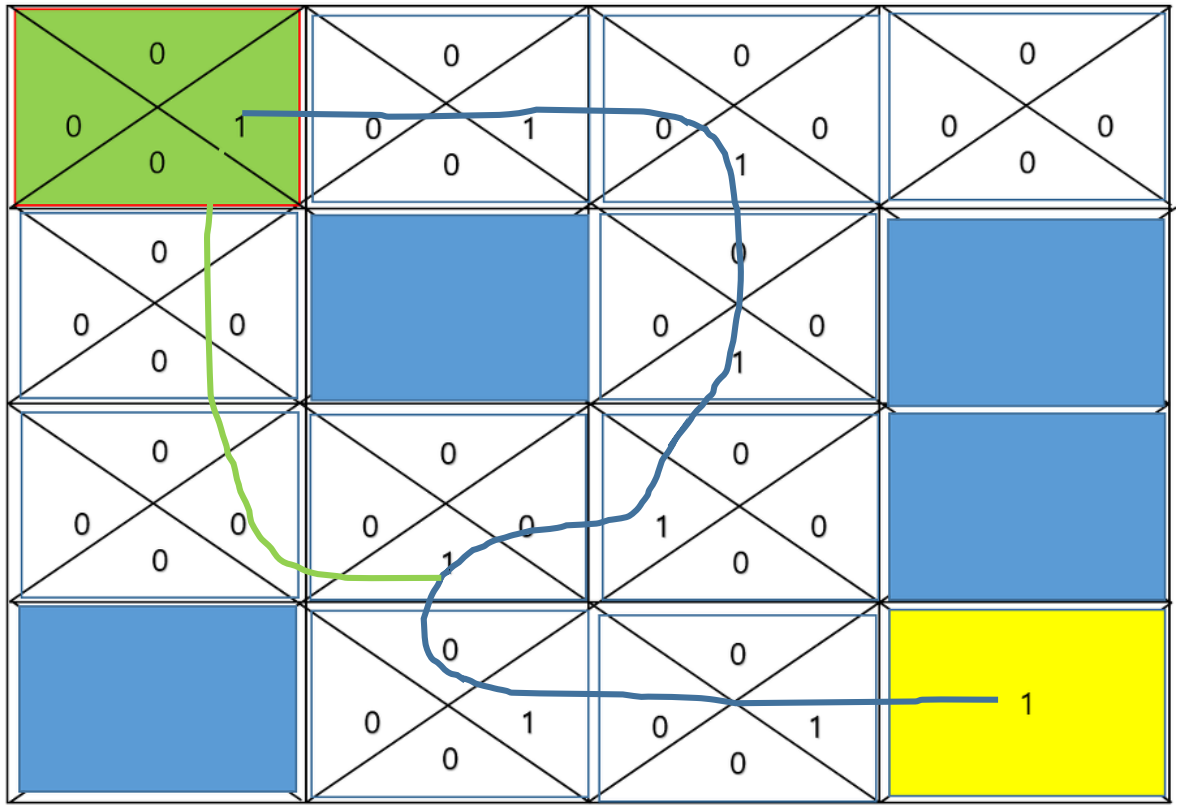
```
for i in range(1000)

    a = argmax(Q(s,a) + random_values / (i+1)
```

01. Q-Learning

Q-Table

# Discounted future reward



Exploit VS Exploration 개념을 추가하여 다양한 경로를 학습함
그런데 최적의 정책은 어떻게 찾을 것인가?

여기서 한가지 개념을 Q-function에 추가한다.

Discounted future reward : 100년 뒤에 50억을 받을실래요?
아니면 1년 뒤에 10억을 받으실래요?

->나중에 받는 보상일 수록 그 값어치를 적게 처리한다

이를 통해 최적 경로를 찾음.

# Discounted future reward

$$\widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{r} + \max_{\boldsymbol{a}`} \widehat{\boldsymbol{Q}}(\boldsymbol{s}',\boldsymbol{a}')$$

위 식에서의 Future reward는 다음과 같다.

$R_t = r_t + r_{t+1} + r_{t+2} + \ldots + r_n$

다음과 같이 Q-function에 0~1사이의 임의의 값 감마를 추가하여 미래의 보상에 대한 가중치를 줄일 수 있다.

$$\widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a}) \leftarrow \boldsymbol{r} + \boldsymbol{\gamma} \max_{\boldsymbol{a}`} \widehat{\boldsymbol{Q}}(\boldsymbol{s}',\boldsymbol{a}')$$

위 식의 future reward는 다음과 같다.

$R_t = r_t + \gamma^1 r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-t} r_n$

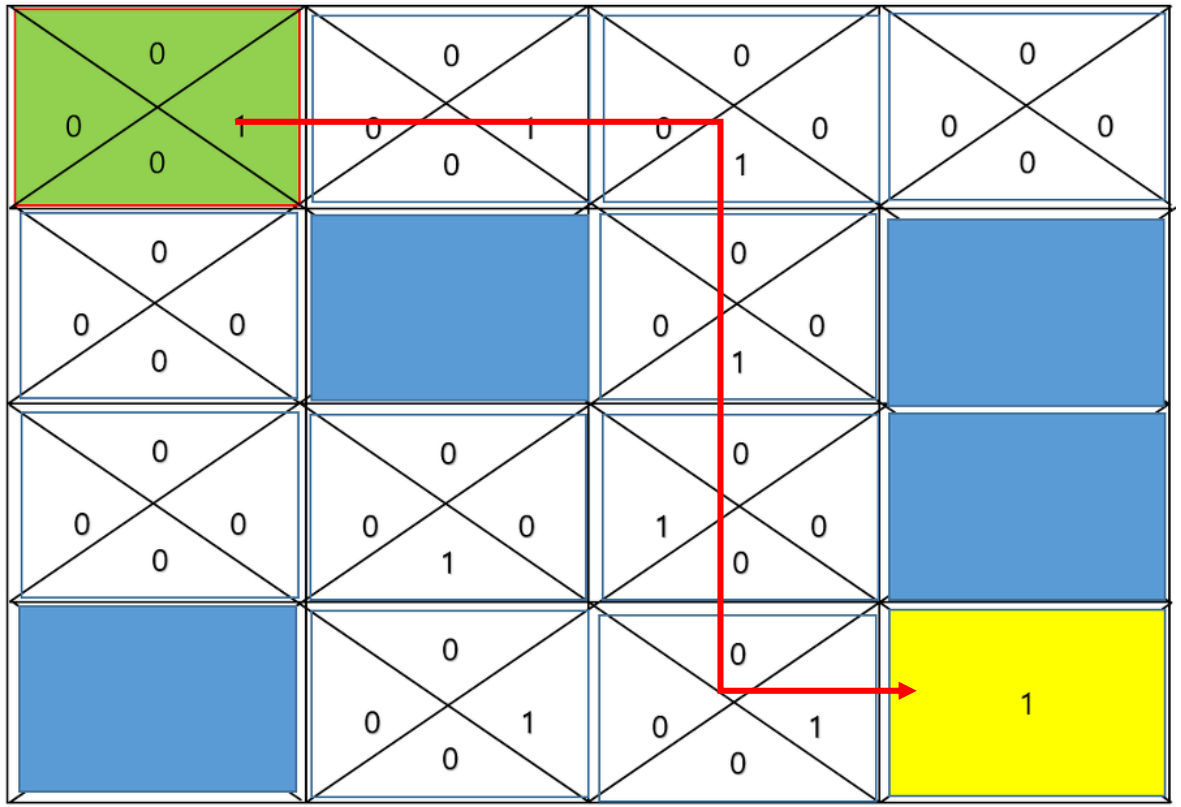
$= r_t + \gamma R_{t+1}$

최적의 정책은 discounted future reward를 최대화 시키는 것이다.

01. Q-Learning

Q-Table

# Discounted future reward



Discounted future reward를 적용하여
돌아가는 길이 아니라 최소 경로를 찾아낸다.

Q function :
지금 상태에서 이 행동을 선택했을 때 미래에 받을 것이라
기대하는 보상의 합

# 02

# DQN

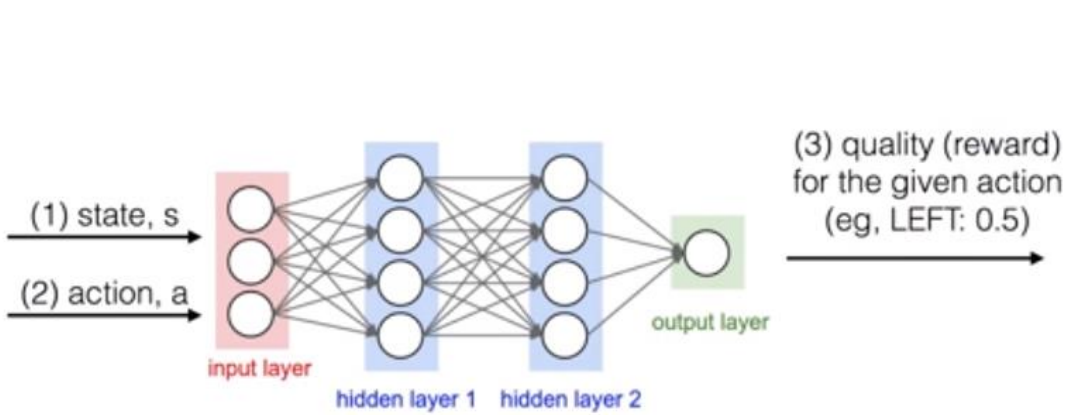
# Q-Network training

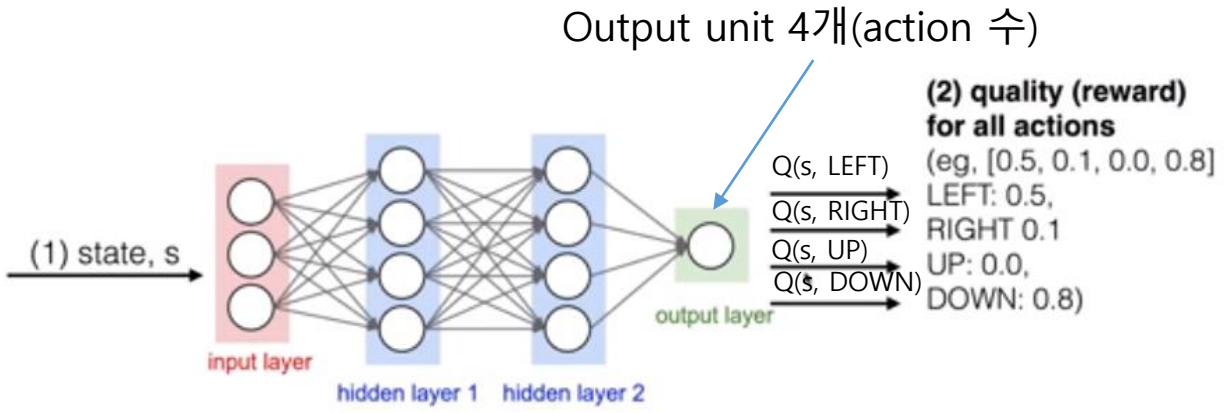
기본아이디어

좀 더 복잡한 환경을 표현하기 위해 Neural Network에 Q-Learning을 적용함

- Input: state, (action)
- Output: Q-function = reward

각 입력들에 대하여 제일 좋은 Q-function을 찾자




Q-Network 2가지 모델링

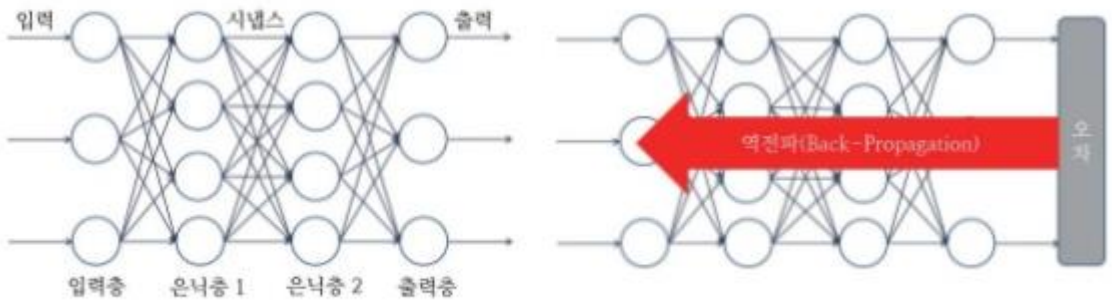
# Q-Network training

MSE loss function $(\widehat{\boldsymbol{Q}}(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{w})-y)^2$ 를 gradient descent 로 최소화 시킴

y = optimal Q = $\boldsymbol{r}+\boldsymbol{\gamma}\max_{\boldsymbol{a}`}\boldsymbol{Q}(\boldsymbol{s}',\boldsymbol{a}') \rightarrow \boldsymbol{r}+\boldsymbol{\gamma}\max_{\boldsymbol{a}`}\widehat{\boldsymbol{Q}}(\boldsymbol{s}',\boldsymbol{a}'|\boldsymbol{w})$

• (정답 – 예측)의 오차를 이용해서 인공신경망에 역전파

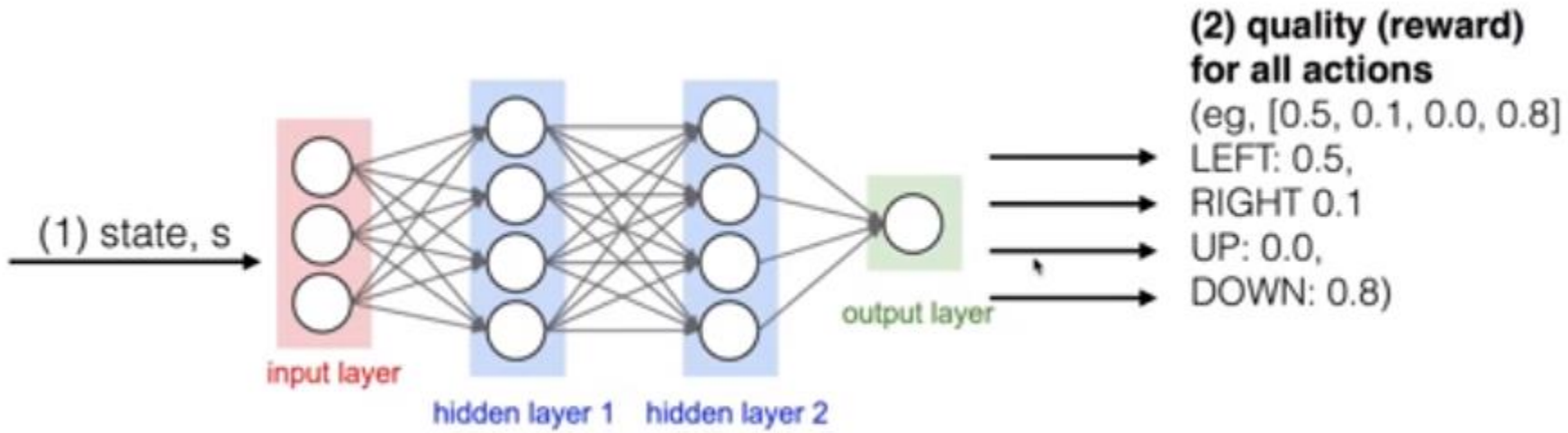
→ 인공신경망의 업데이트



# Q-Network training

Q-Table과 다르게 Q-Network는 수렴하지 않음.

1) Correlations between samples
2) Non-stationary target

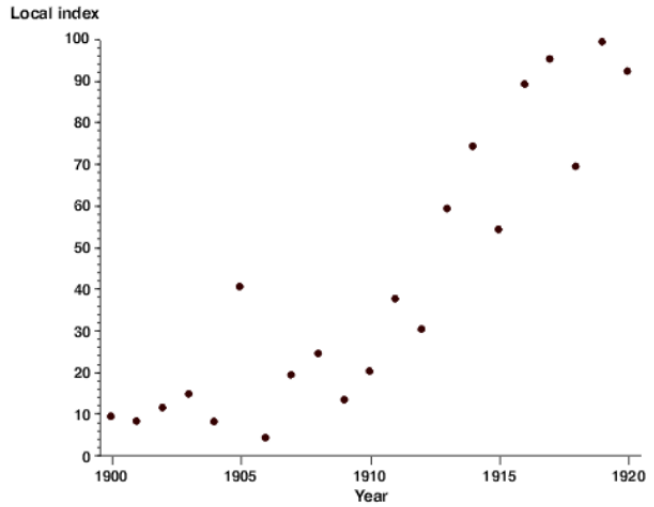


02. DQN

Q-Network

# Correlations between samples

게임을 순서대로 진행하는 것을 학습하는데
이 때 input 값들이 너무 비슷해서 학습이 제대로 이루어 지지 않는다.



# Non-stationary target

y = optimal Q = $\boldsymbol{r} + \boldsymbol{\gamma} \max_{\boldsymbol{a}`} \boldsymbol{Q}(\boldsymbol{s}', \boldsymbol{a}') \rightarrow \boldsymbol{r} + \boldsymbol{\gamma} \max_{\boldsymbol{a}`} \widehat{\boldsymbol{Q}}(\boldsymbol{s}', \boldsymbol{a}' | \boldsymbol{w})$

MSE = $(\widehat{\boldsymbol{Q}}(\boldsymbol{s}, \boldsymbol{a} | \boldsymbol{w}) - \left(\boldsymbol{r} + \boldsymbol{\gamma} \max_{\boldsymbol{a}`} \widehat{\boldsymbol{Q}}(\boldsymbol{s}', \boldsymbol{a}' | \boldsymbol{w})\right))^2$

정답이 학습 과정에서 계속 바뀌기에 학습이 제대로 이루어지지 않는다.

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$

DQN (NIPS 2013)

# DQN

게임 화면을 상태 입력으로 받아서 학습(CNN)

사람보다 게임 플레이를 더 잘하는 에이전트를 만듬

샘플들 간의 강한 상관 관계를 해결

-> 강화 학습을 다시 부흥시킴

02. DQN

DQN (NIPS 2013)

# Breakout

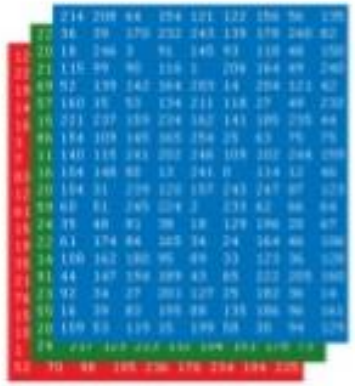

**행동** : 제자리, 좌, 우

**보상** : 벽돌을 깰 때 마다 점수를 받으며
위층의 벽돌을 깰수록 더 큰 점수를 받음

**게임 세팅** : 1에피소드에서 에이전트는 5개의 목숨을 가짐

**목표** : 1 에피소드 동안 최대의 점수 얻기

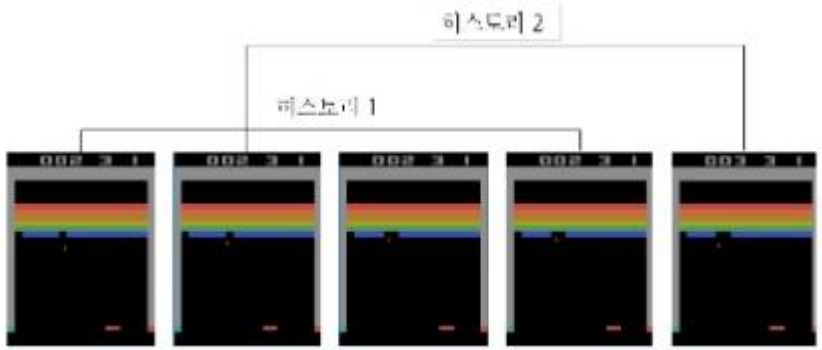
모든 상태의 큐함수를 다 저장하고
업데이트 할 수 없음



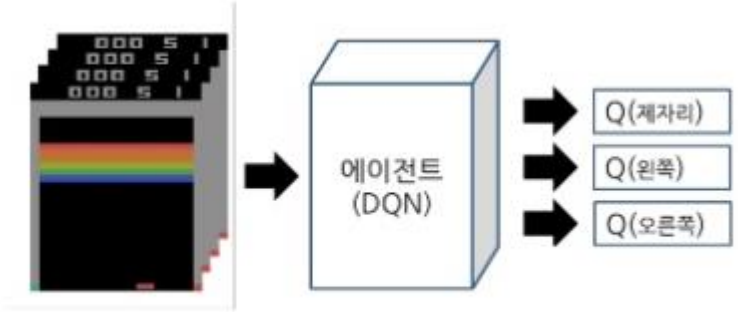
02. DQN

DQN (NIPS 2013)

# CNN + Q-Learning




현재의 이미지 하나만으로는 게임 화면이 어떤 상황인지
알 수 없기에 4개의 연속된 이미지를 입력으로 받음



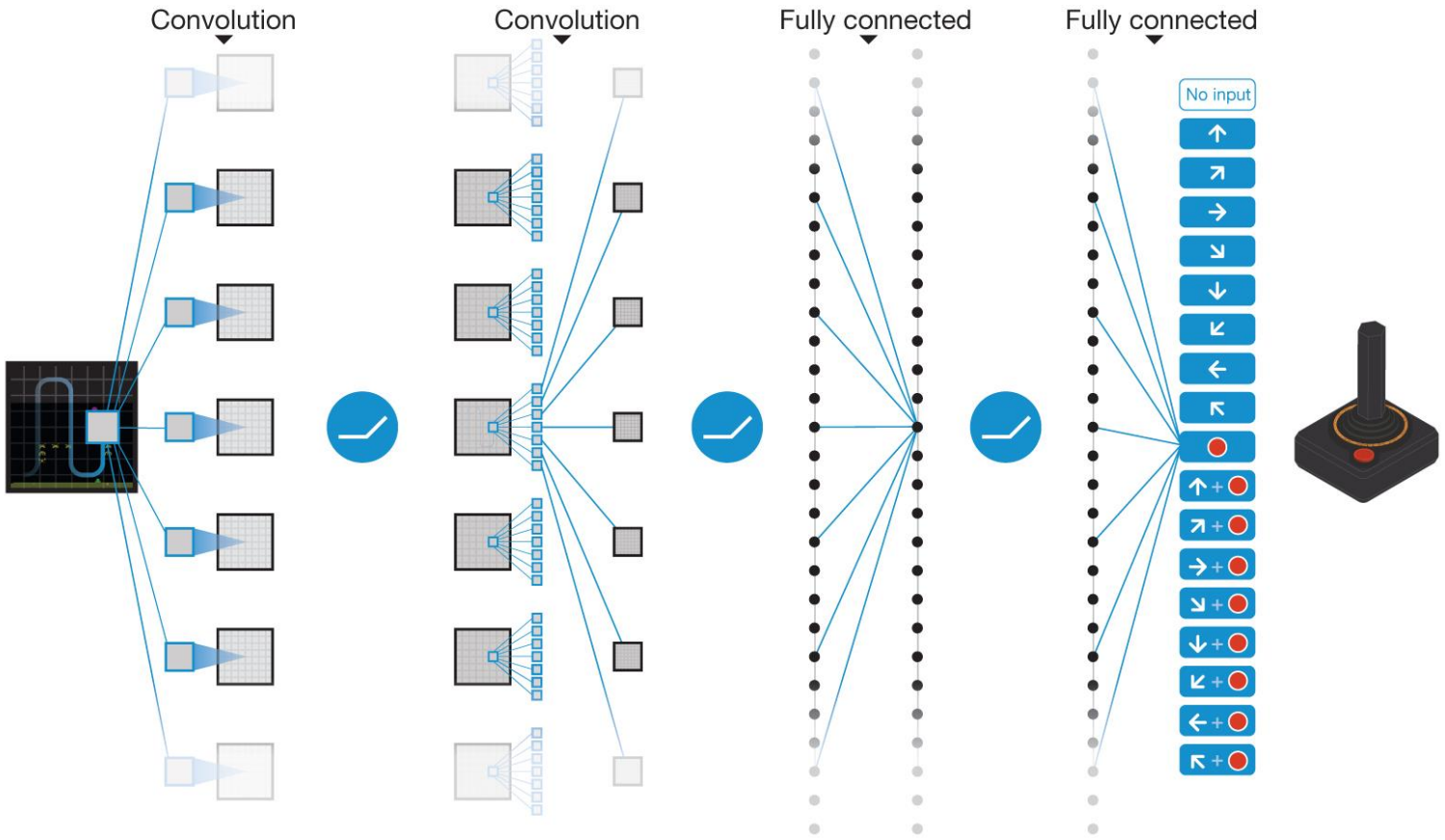

DQN을 통하여 각 행동들에 대한 Q함수 값이 출력으로 나옴

상태(히스토리) → 큐-신경망 → 각 행동에 대한 큐함수 값

02. DQN

DQN (NIPS 2013)

# CNN + Q-Learning



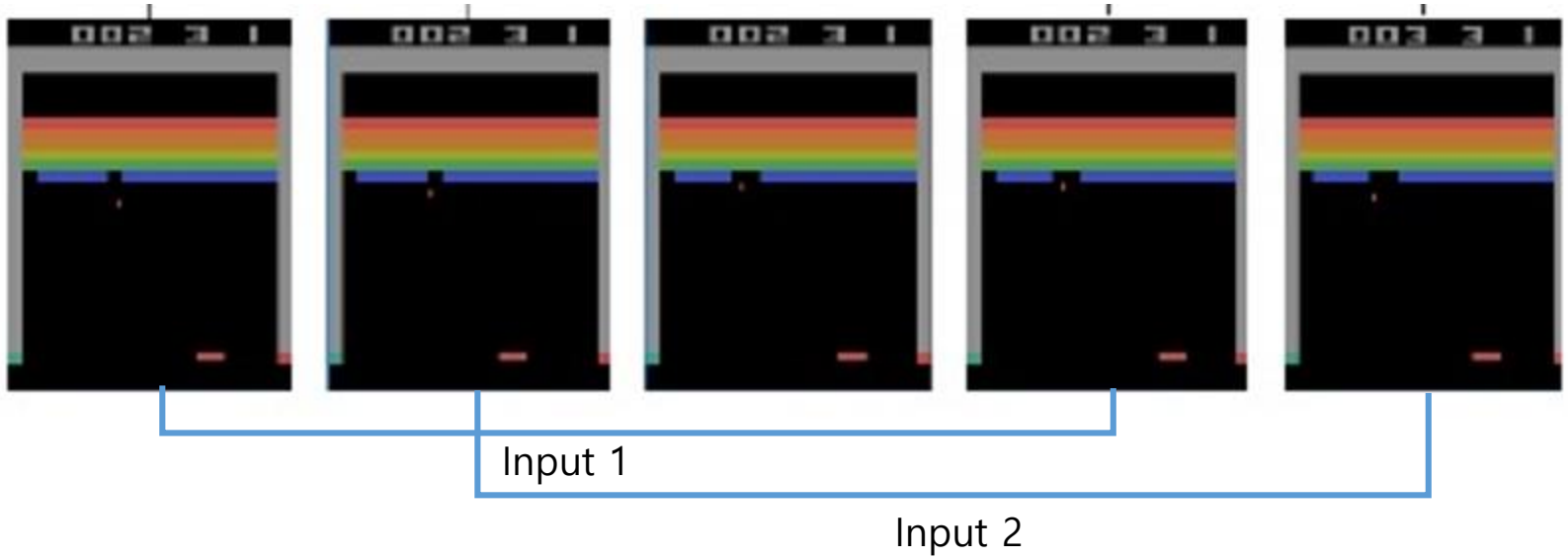

https://www.nature.com/articles/nature14236

Q함수를 추정하는 DQN의 개념도

02. DQN

DQN (NIPS 2013)

# Experience Replay

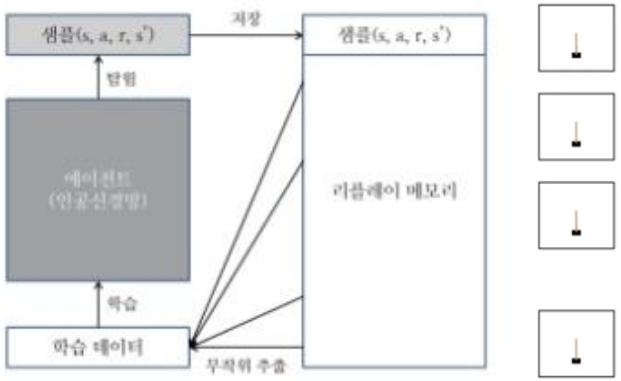



입력 간의 상관 관계가 너무 크기에 학습이 제대로 되지 않음
(**Correlations between samples**)

02. DQN

DQN (NIPS 2013)

# Experience Replay






Capture

$s_1, a_1, r_2, s_2$
$s_2, a_2, r_3, s_3$
$s_3, a_3, r_4, s_4$
…
$s_t, a_t, r_{t+1}, s_{t+1}$

random sample
& Replay

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$


그림 7.14 4개의 연속된 화면의 입력

Input들을 미리 100만 사이즈의 버퍼에 저장한 뒤 랜덤 추출하여 학습!
-> **Correlations between samples** 해결

Frame-skipping 기법을 함께 사용하여 학습을 향상

02. DQN

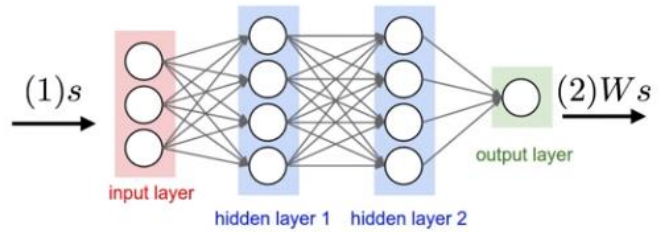
DQN (Nature 2015)

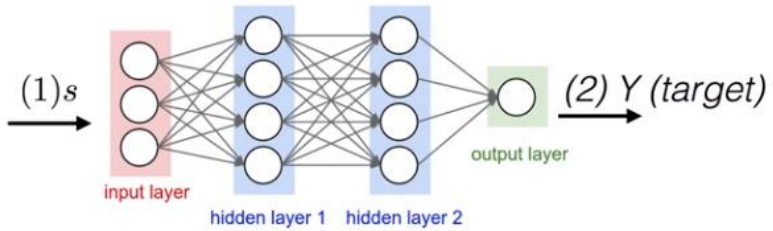
# target Q-network

## Solution 3: separate target network

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\bar{\theta}))]^2$$

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$




Target network를 분리한다.

기존에 같이 학습되었던 target을 분리하여 타겟을 고정시키고, 일정 주기로 q-network를 target-network로 복사한다.

-> **non-stationary targets** 문제 해결
->Nature!

# 03

## A3C

03. A3C

A3C란?

# A3C = Asynchronous Actor-Critic Agents

**DQN**

1. 많은 메모리의 사용
2. 느린 학습 속도
3. 불안정한 학습 과정 (가치 함수에 대한 greedy policy)

**A3C**

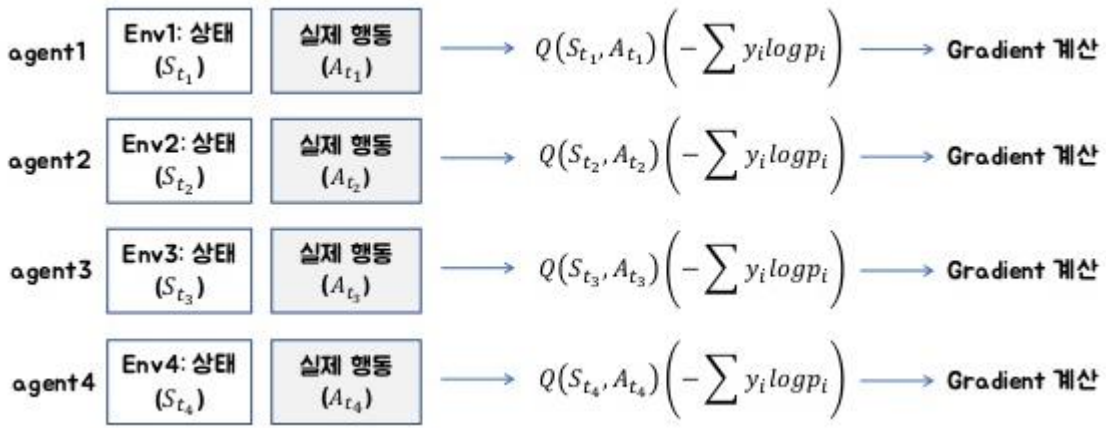
1. 샘플 사이의 상관 관계를 비동기 업데이트로 해결
2. 리플레이 메모리를 사용하지 않음
3. Policy gradient 알고리즘 사용가능(Actor-Critic) (DQN Value gradient)
4. 상대적으로 빠른 학습 속도 (여러 에이전트가 환경과 상호작용)

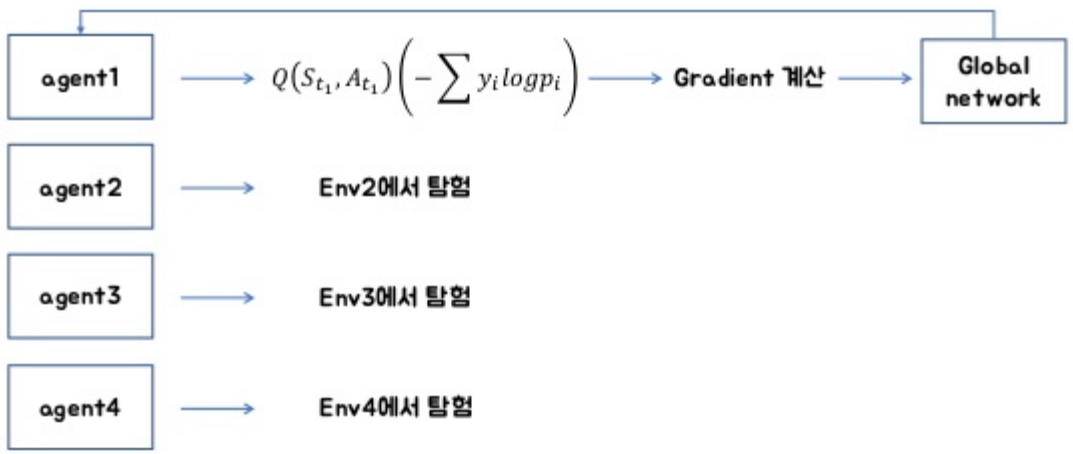
# 03. A3C

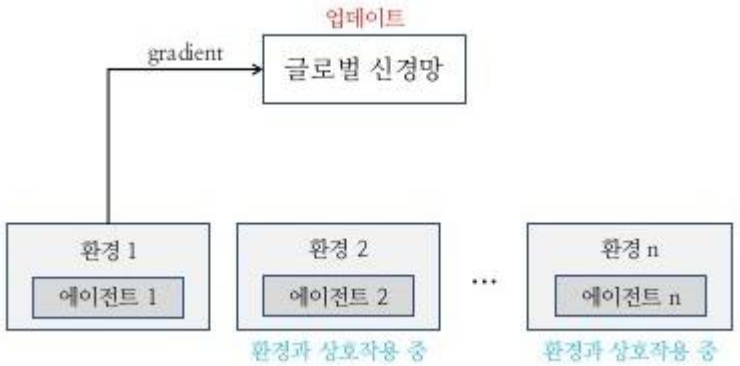
## A3C란?

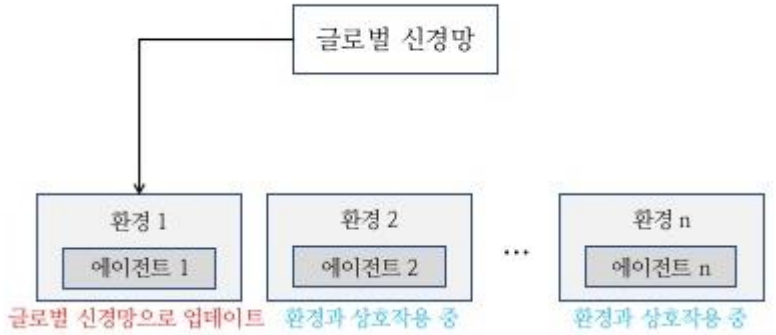
### 기본 아이디어

여러 개의 에이전트를 만들어서 각각 gradient를 계산한다.
비동기적으로 global network를 업데이트 한다.



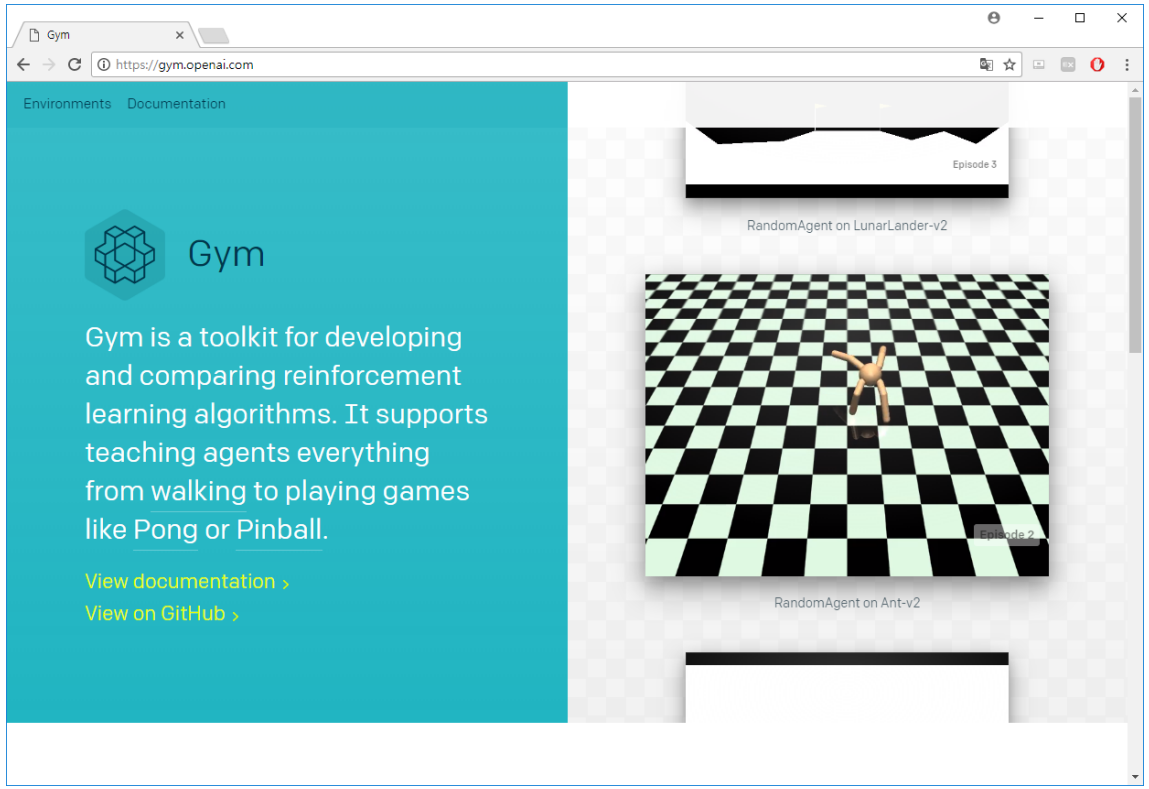



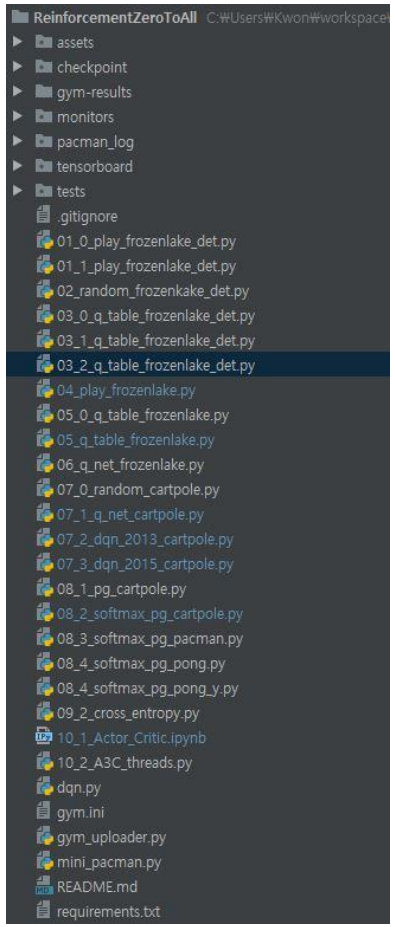
# 04

## CODE

# 04. CODE




OpenAi gym
강화 학습 환경이 구축되어 있는 오픈소스 라이브러리




OpenAi gym을 이용한 다양한 예제
Reference – code 참고

# 05. REFERENCE


https://www.wikipedia.org/ 위키피디아

https://www.inflearn.com/course/reinforcement-learning/ 모두를 위한 딥러닝 –RL편
http://passi0n.tistory.com/86?category=748105 Q-Table

https://www.youtube.com/watch?v=V7_cNTfm2i8&t=0s&index=6&list=PLlMkM4tgfjnJhhd4wn5aj8fVTYJwIpWkS
DQN 2013 논문 리뷰
https://www.nature.com/articles/nature14236 DQN Nature 2015
https://www.slideshare.net/WoongwonLee/ss-78783597 DQN 참조
http://www.modulabs.co.kr/RL_library/3305
https://jay.tech.blog/2017/01/10/deep-q-network-dqn/

https://arxiv.org/abs/1602.01783 A3C ICML 2016
https://www.slideshare.net/WoongwonLee/rlcode-a3c A3C 이해

https://github.com/hunkim/ReinforcementZeroToAll CODE
https://gym.openai.com/

# Thank you