Reinforcement Learning - 1
AI & OPTIMIZATION LAB

김 태 민

Contents

I 강화학습의 개요

- **지도학습 (Supervised Learning)**

  → **정답이 주어진 데이터**로 학습해서 새로운 데이터에 대한 값이나 카테고리를 예측하는 것

- **비지도학습 (Unsupervised Learning)**

  → **정답이 없는 데이터**를 적절히 그룹화하거나 각 데이터 간의 관계를 찾아내는 것

- **강화학습 (Reinforcement Learning)**

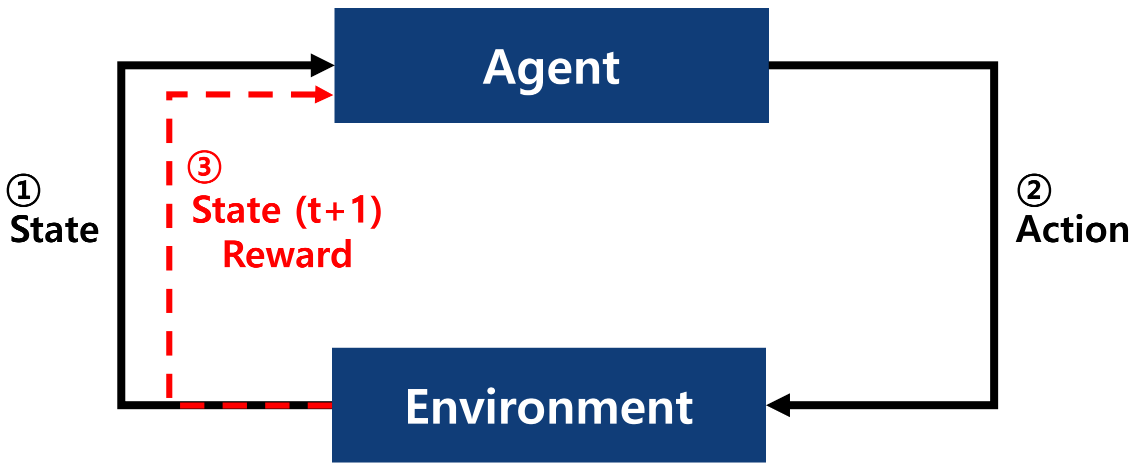
  → 어떤 임의의 존재(Agent)가 주어진 환경 내에서 어떻게 행동해야 하는지에 대해 학습하는 것

  → 학습 과정 속에서 다양한 **상황(State)**에 따라 Agent는 **행동(Action)**을 하면 그것에 대한 **보상(Reward)**을 받는다.

- **강화학습의 최종 목표**

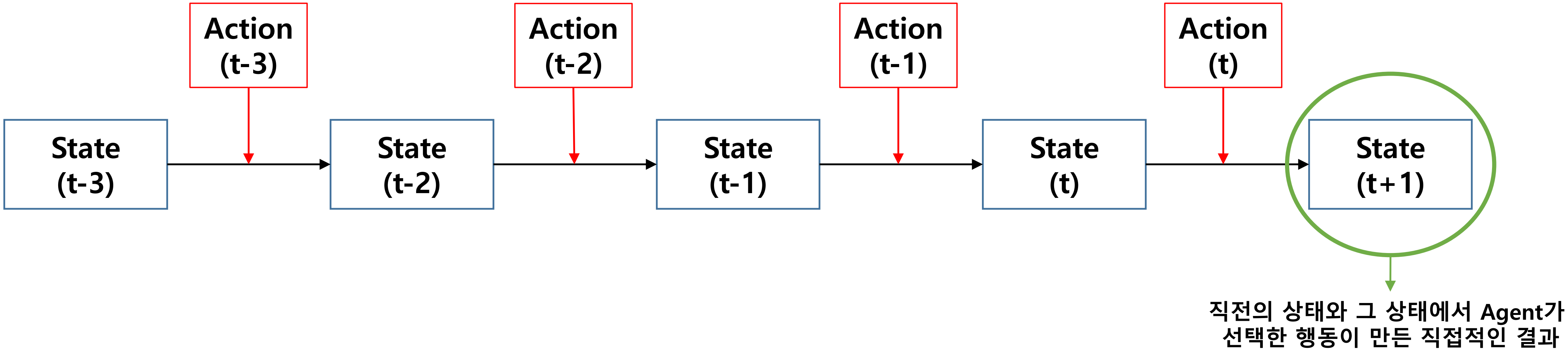
  → 환경과 상호작용을 하는 임의의 Agent를 학습시키는 것

  → Agent의 목표는 처음 시작하는 시점부터 종료시점까지 일어나는 모든 에피소드에서 받을 Reward 값을 최대로 끌어올리는 것

강화학습 구조	강화학습 순서 요약
	① Environment에서 현재 상태**(t시점의 State)**를 Agent에게 전달 ② Agent는 현재 상태**(t시점의 State)**에 따른 **행동(Action)**을 취함 ③ Agent의 행동에 따른 보상**(Reward)**을 계산한다. ④ Agent의 행동에 따라 바뀐 상태**(t+1시점의 State)**를 정의한다.

- 마리오(Agent)에게 주어지는 State는?
  → 매 순간마다 보여지는 이미지
- 마리오(Agent)가 할 수 있는 Action은?
  → 앞으로 이동
  → 뒤로 이동
  → 점프
- 마리오(Agent)에게 주어지는 Reward는?
  → 코인을 먹거나 아이템을 얻을 경우 (+)
  → 적에게 맞아서 죽는 경우 (-)
  → 여기저기 돌아다니기만 하는 경우 (0)

마르코프 결정 프로세스(MDP)와 벨만 방정식



- 연결되어 있는 모든 단계들과 그 순서는 현재 상태를 결정짓는 어떤 정보를 담고 있을 것이고, 그에 따라 지금 순간에 Agent가 어떤 행동을 선택해야 하는지에 대해 직접적인 영향을 미친다.

  → 그렇다면, Agent가 이전 단계의 모든 정보를 사용한다면 더 좋은 선택(Maximize Reward)을 할 수 있지 않을까?

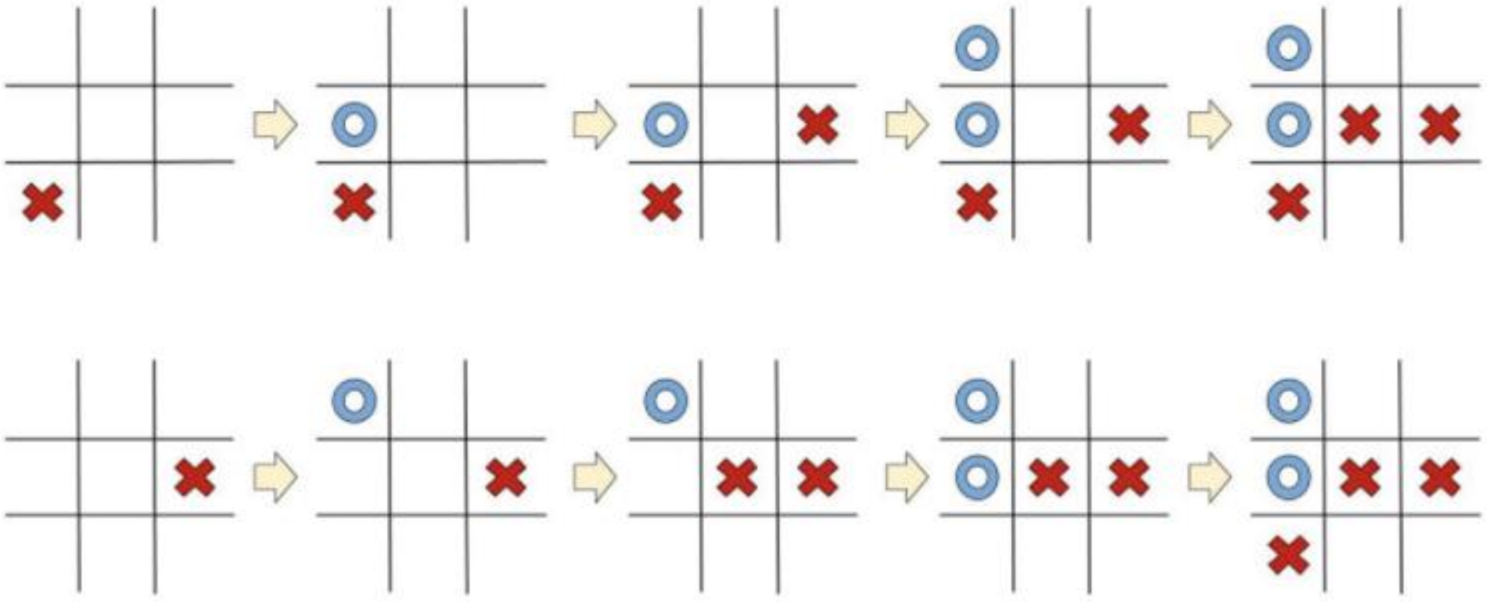
단계가 계속 진행되어 **저장해야 할 정보의 양이 많아지면**

이 데이터를 연산하는데 **많은 저장공간과 시간이 사용**될 것이다.

- **이 문제를 해결하기 위해, 우리는 모든 상태가 Markov State에 해당한다고 가정**

- **Markov State**

  → "모든 상태는 오직 그 직전의 상태와 그 때 한 행동에서만 의존한다."는 가정



<Tic Tac Toe 예제>

- 벨만 방정식

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

- Q(s, a) : 상태 s에서 행동 a를 취할 때 받을 수 있는 모든 보상의 총합
- r(s, a) : 현재 상태 s에서 행동 a를 취했을 때 받을 즉각 보상값
- s' : 현재 상태 s에서 행동 a를 취해 도달한 다음의 상태
- Max Q(s', a) : 다음 상태 s'에서 받을 수 있는 보상의 최대값

→ 이 값을 최대화할 수 있는 행동을 선택해내는 것이 Agent의 목표

- 할인율 : 미래가치에 대한 중요도

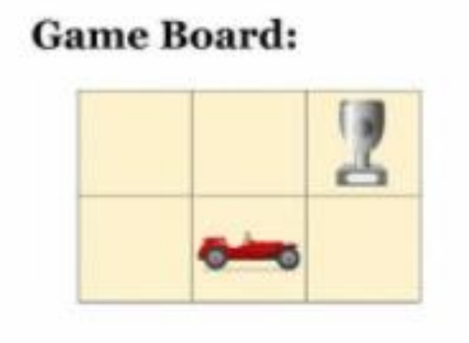
→ 값이 커질수록 미래에 받을 보상이 더 큰 가치를 가짐

→ 작아질수록 즉각 보상을 더 중요하게 고려

- Markov States 가정이 유효하다면, 벨만 방정식으로 미래에 받을 수 있는 보상을 멀리 떨어진 과거로 전파할 수 있다.
- 처음에 실제 Q 값이 얼마인지 알지 못해 추측으로 값을 정하지만, 점점 수렴해서 마지막에는 정답에 도달한다.

Q - Learning

- 벨만 방정식을 통해 우리는 **"어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다."**라는 전략을 수립할 수 있다.



State : 0 0 0
0 1 0

Action :

Reward :

Next State :

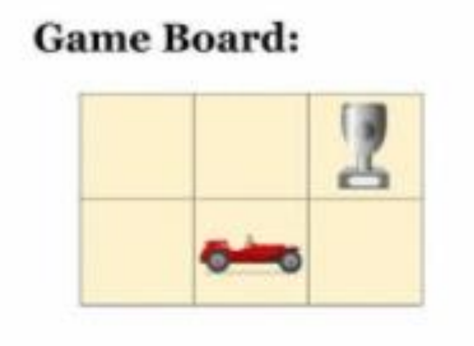
Q Table

할인율 : 0.95

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Q - Learning

- 벨만 방정식을 통해 우리는 "어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다."라는 전략을 수립할 수 있다.



State : 0 0 0
0 1 0

Action : →

Reward :

Next State :

Q Table

할인율 : 0.95

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Q - Learning

- 벨만 방정식을 통해 우리는 **"어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다."**라는 전략을 수립할 수 있다.

Game Board:

State : 0 0 0
0 1 0

Action : →

Reward : 0

Next State : 0 0 0
0 0 1

Q Table

할인율 : 0.95

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Q - Learning

- 벨만 방정식을 통해 우리는 **"어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다."**라는 전략을 수립할 수 있다.

State : 0 0 0
0 1 0

Action : →

Reward : 0

Next State : 0 0 0
0 0 1

Max Q(s′, a) : 1.0

Q Table

할인율 : 0.95

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Ⅲ Q - Learning

- 벨만 방정식을 통해 우리는 **"어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다."**라는 전략을 수립할 수 있다.

Game Board:

State : 0 0 0
0 1 0

Action : →

Reward : 0

Next State : 0 0 0
0 0 1

Max Q(s', a) : 1.0

Q Table

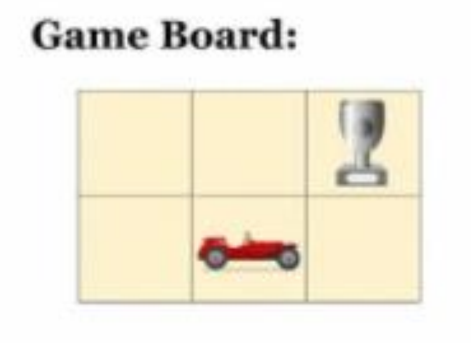
할인율 : 0.95

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

New Q(s, a) = 0 + 0.95 * 1 = 0.95

Ⅲ Q - Learning

- 벨만 방정식을 통해 우리는 **"어떤 상태이든, 가장 높은 누적 보상을 얻을 수 있는 행동을 취한다."**라는 전략을 수립할 수 있다.



State : 0 0 0
0 1 0

Action : →

Reward : 0

Next State : 0 0 0
0 0 1

Max Q(s', a) : 1.0

Q Table

할인율 : 0.95

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.95	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

New Q(s, a) = 0 + 0.95 * 1 = 0.95

Q - Learning

- **앞에 예제처럼 계속 Q Table을 업데이트를 시킨다면 어떤 상태에서든 "최선의 선택"을 할 수 있을 것이다.**

- **탐욕적 알고리즘의 문제**

  → 항상 "최선의 선택"만 고집한다면, 새로운 선택을 해보지 않을 것이고, 더 좋은 최적해가 있음에도 불구하고 찾지 못할 수 있다.

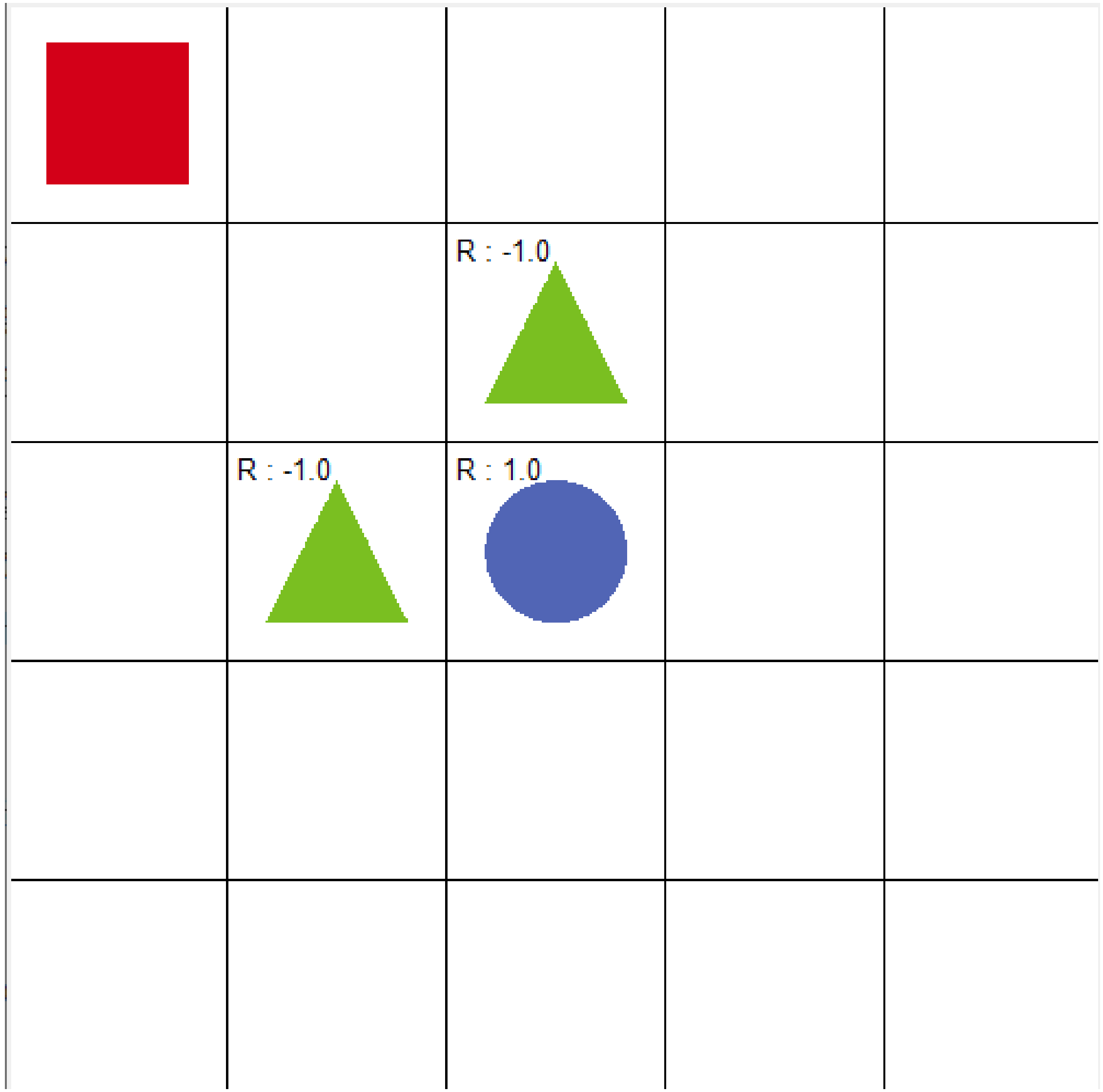
- **ε-greedy 전략의 사용**

  → 0 < ε < 1인 어떤 ε 값을 사용해서, **탐험(Exploration)**과 **활용(Exploitation)**을 적절히 사용

  → 활용(Exploitation), 즉 **탐욕적인 행동을 할 확률 p = 1 – ε**

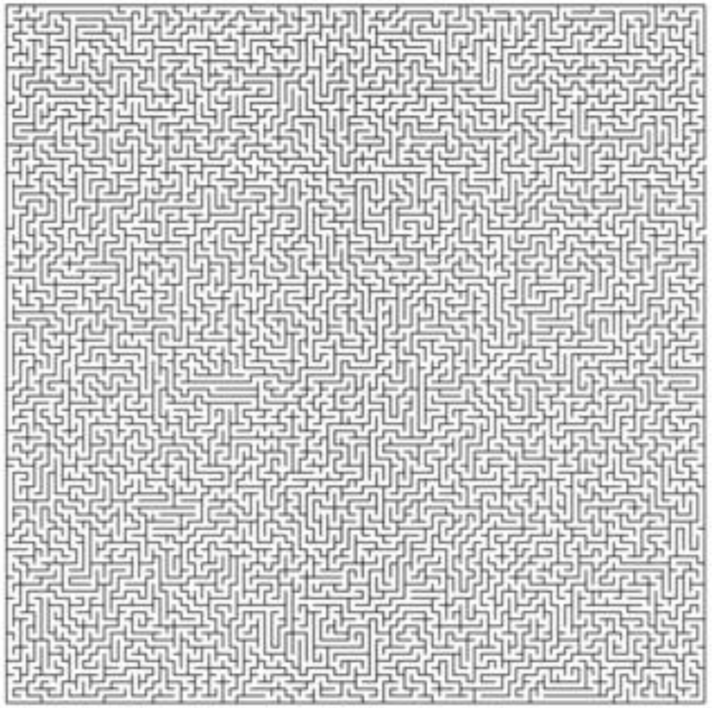
  → 탐험(Exploration), 즉 **랜덤으로 행동을 취할 확률 p = ε**

Q - Learning



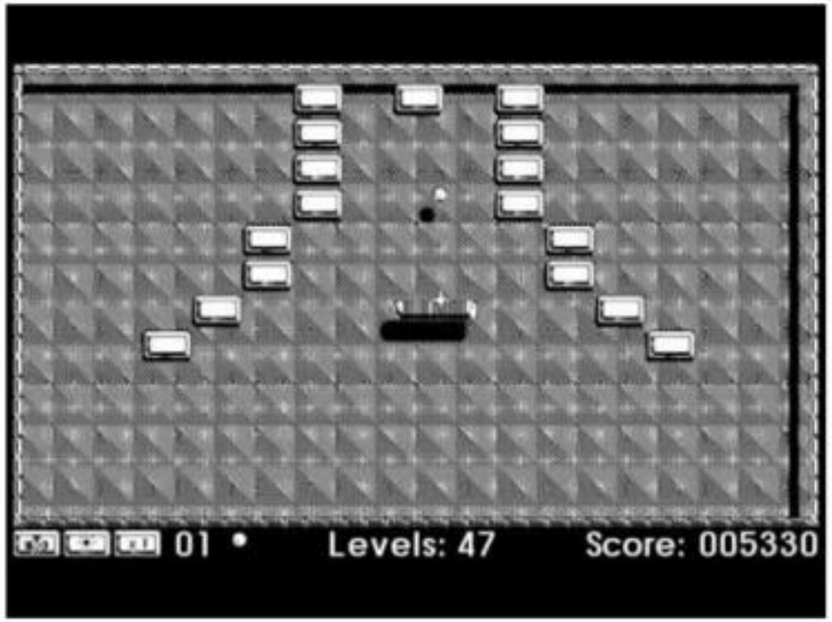
Q - Network

- Q-Learning으로 간단한 문제는 풀 수 있지만, 방대한 양의 상태가 존재하는 문제에서 Q-Table을 작성해 풀기는 어려움



100x100 maze

→ 100 * 100 * 4의 Array가 필요

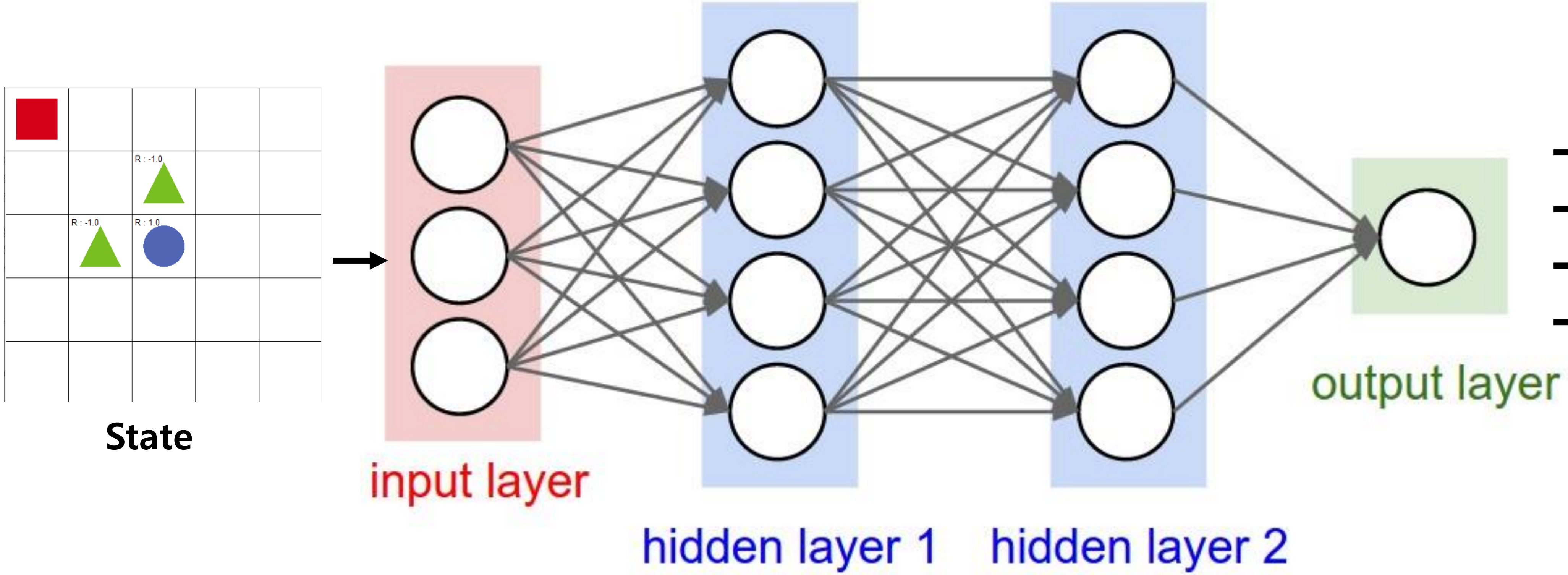


80x80 pixel + 2 color (black/white)

→ 2^(80*80)의 Array가 필요

IV Q - Network

- **Q-Network Idea**



Q-Value for all Action

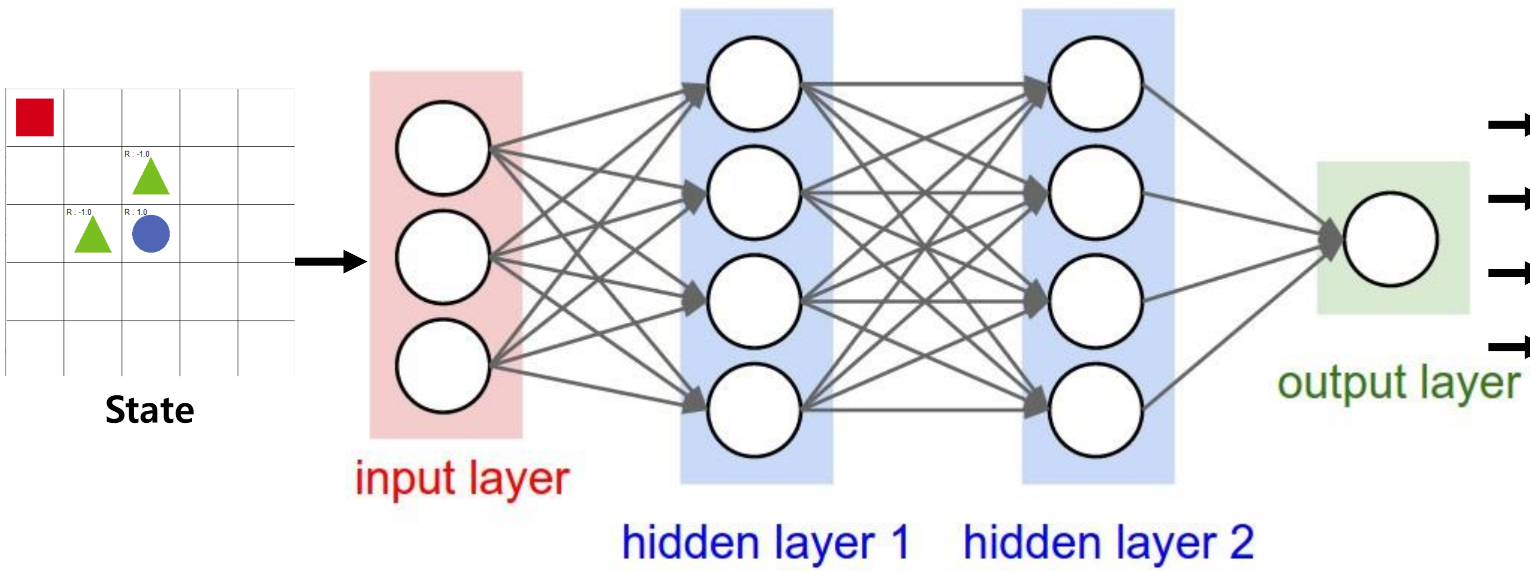
Left : -1.0

Right : 0.5

Up : -1.0

Down : 0.5

- Q-Network Training (Linear Regression)



Q-Value
for all Action

→ Left : -1.0

→ Right : 0.5

→ Up : -1.0

→ Down : 0.5

⇩

$\hat{Q}(s, a \mid \theta) \sim Q^*(s, a)$

↓

$Q^* = r + \gamma \max Q(s')$

Q – Network 학습의 목표

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

Algorithm 1 Deep Q-learning

Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  s1의 전처리
  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 $\phi_{t+1} = \phi(s_{t+1})$

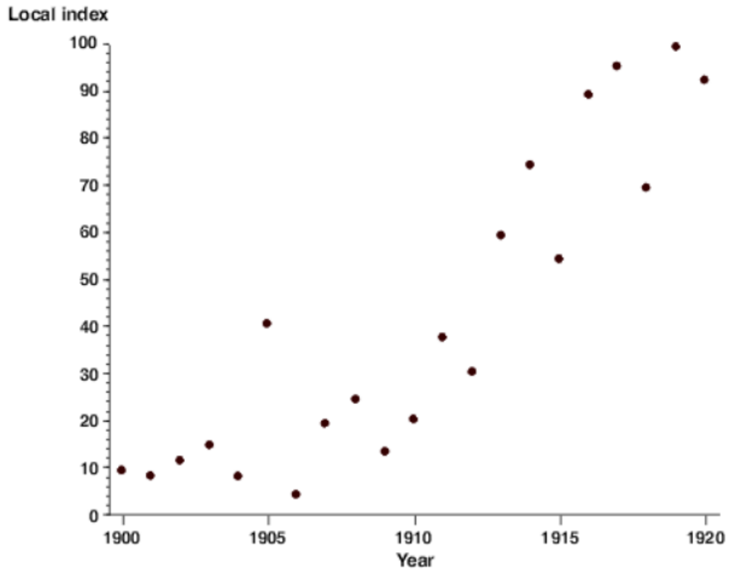
    Set $y_j = \begin{cases} r_j & \text{for } \boxed{\text{terminal } \phi_{j+1}} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ → Game Over일 경우
    Perform a gradient descent step on $\boxed{(y_j - Q(\phi_j, a_j; \theta))^2}$ according to equation 3
    Loss Function
  end for
end for

IV Q - Network

- **Q-Network의 한계**

→ Correlations between samples

→ 어떠한 Action을 취했을 때, 받아오는 State는 굉장히 유사하다.

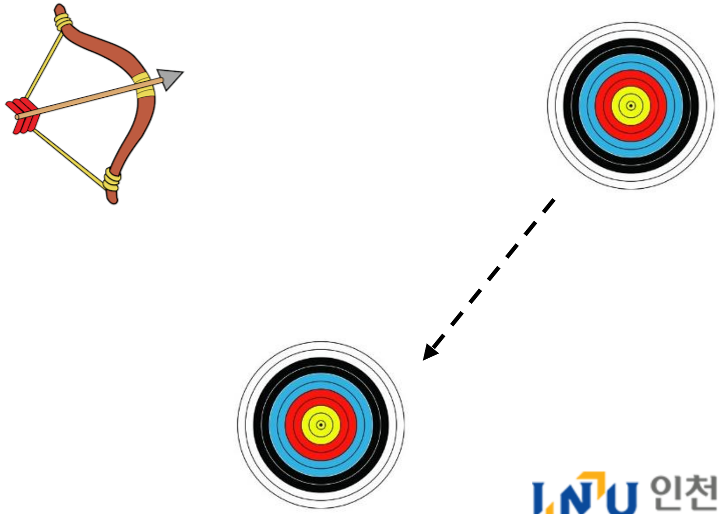


→ Non-stationary targets

→ Target이 움직인다.

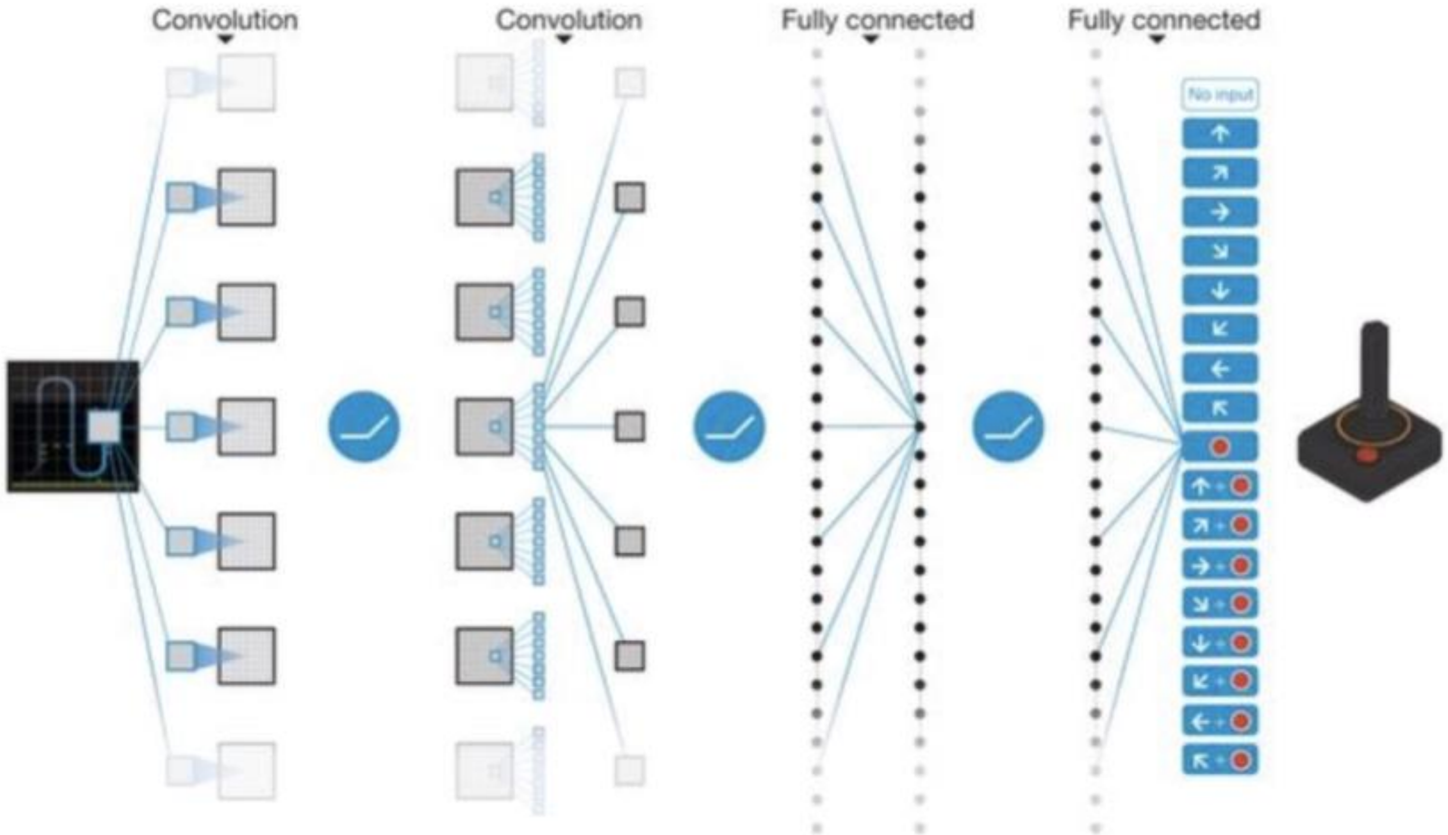
→ 하나의 Network에서 θ를 업데이트하면 Q*의 θ도 업데이트된다.

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

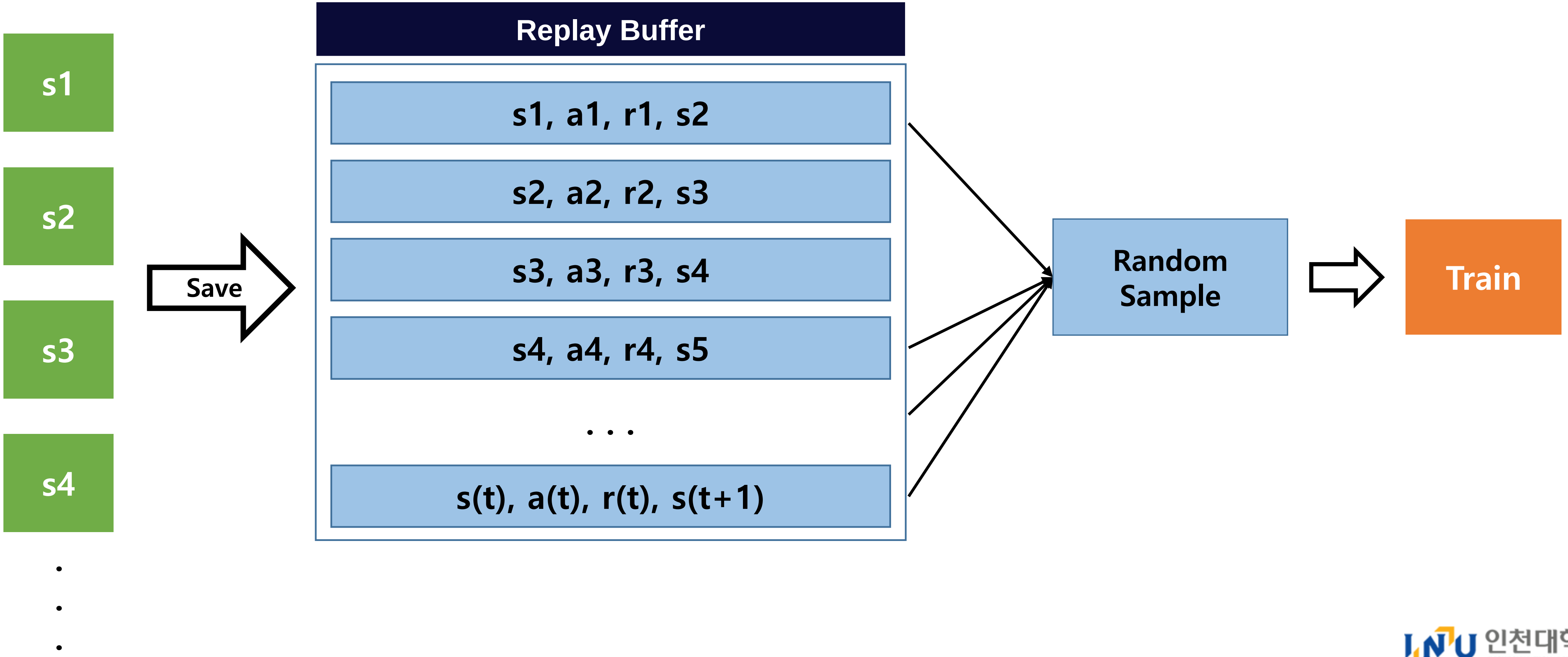


- **Q-Network의 한계를 극복하기 위한 방법**

    1. Go Deep

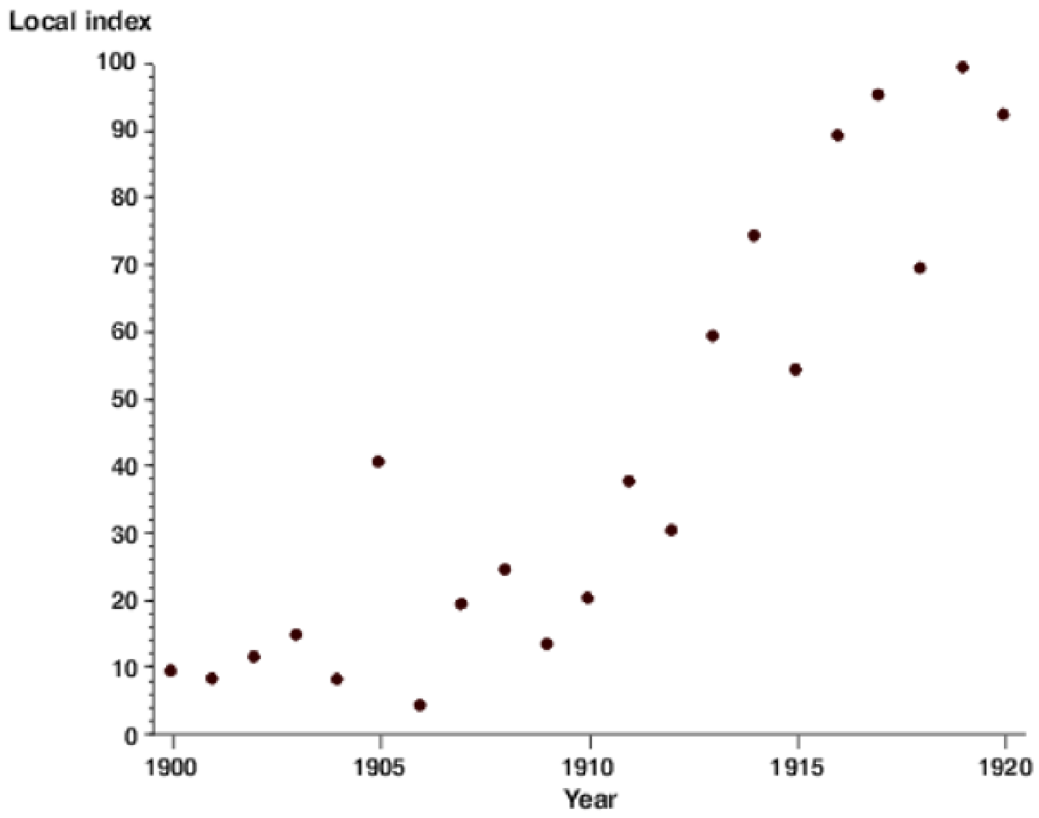


- Q-Network의 한계를 극복하기 위한 방법
    2. Replay Buffer



- **Q-Network의 한계를 극복하기 위한 방법**

  2. **Replay Buffer**



- **Q-Network의 한계를 극복하기 위한 방법**

  3. **Target Network**

기존의 Q-Network

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

DQN

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

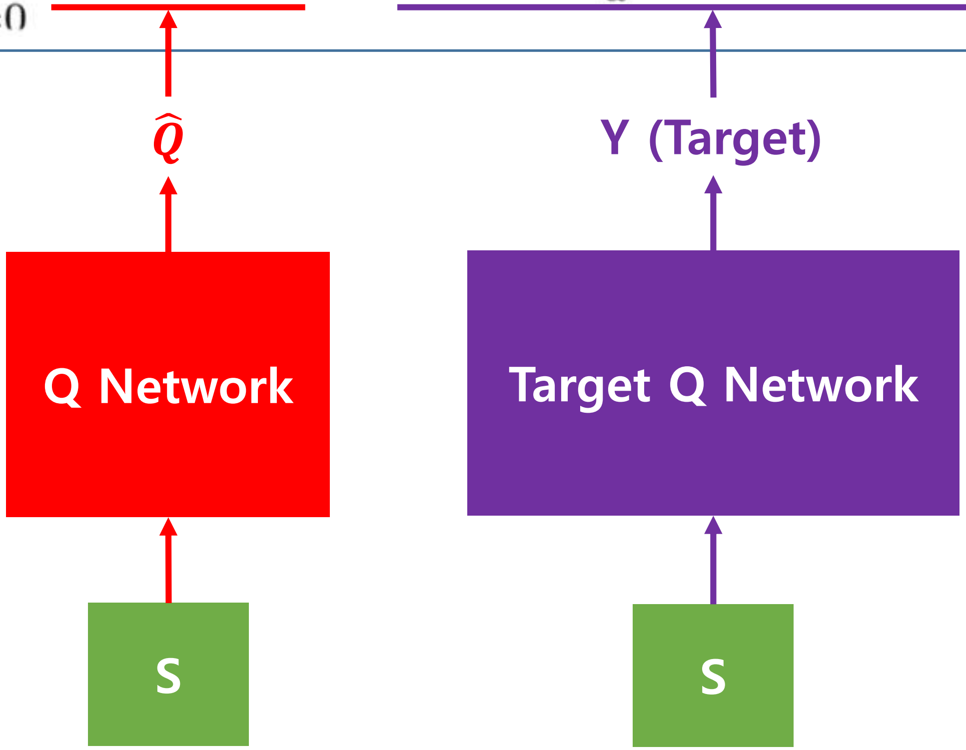
V DQN

- **Q-Network의 한계를 극복하기 위한 방법**

  3. **Target Network**

DQN

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

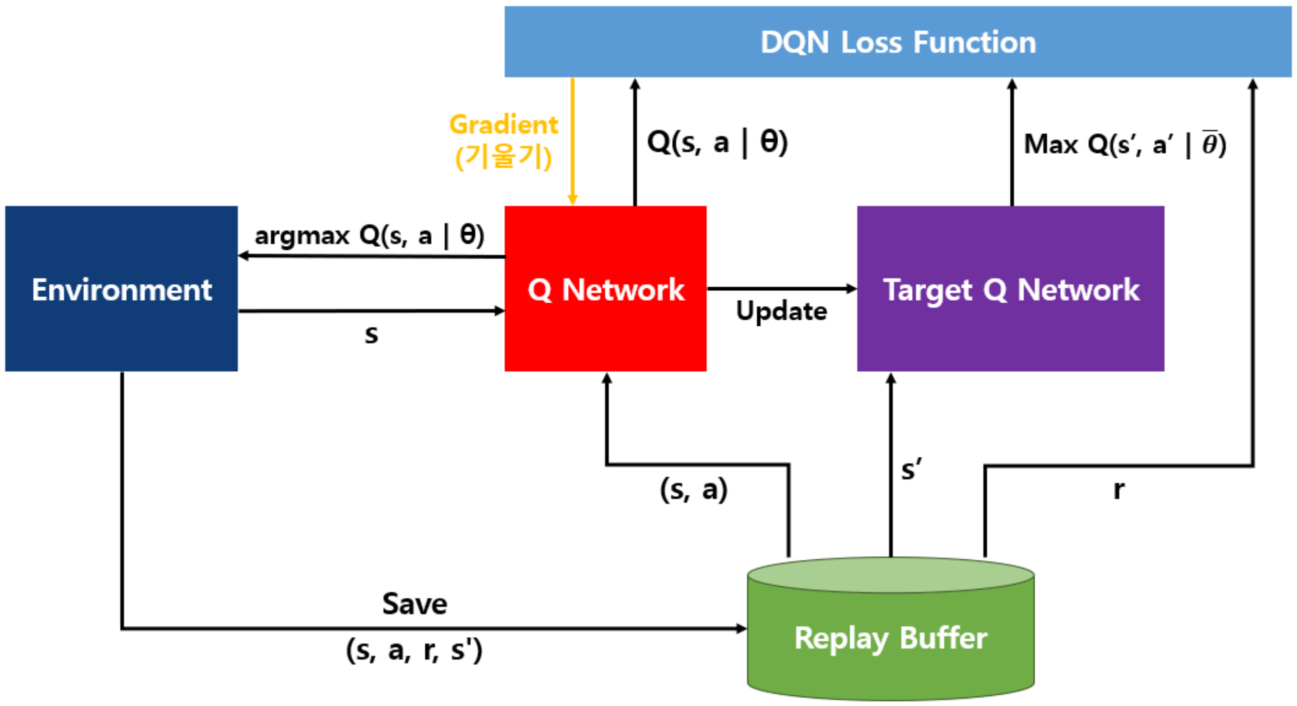


V DQN

DQN Algorithm

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t = 1$,T **do**
        With probability ε select a random action a_t
        otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
        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 D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
        Every C steps reset $\hat{Q} = Q$
    End For
End For

DQN Framework

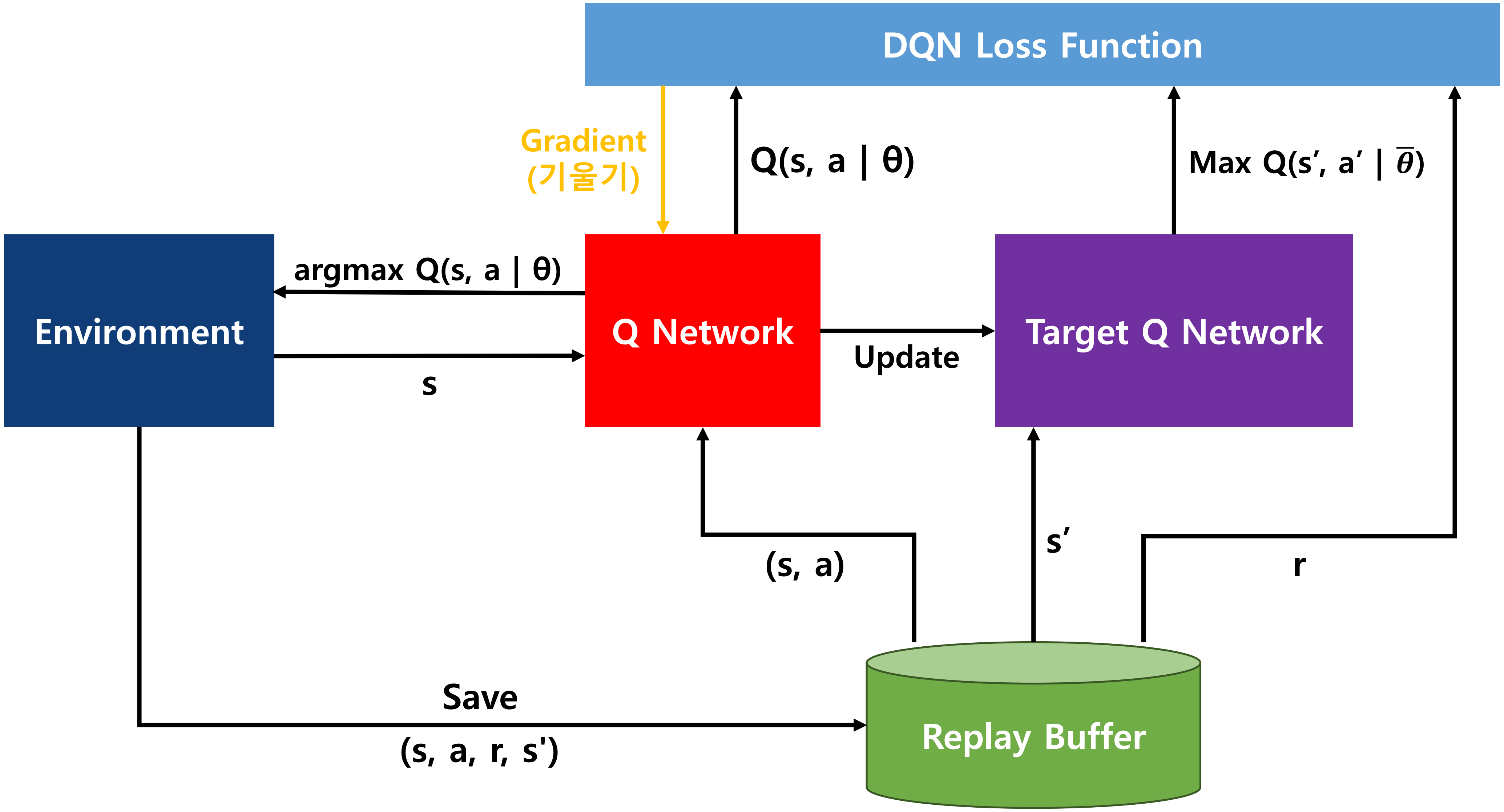


참고문헌

- https://hunkim.github.io/ml/
- https://jeinalog.tistory.com/20
- https://towardsdatascience.com/qrash-course-deep-q-networks-from-the-ground-up-1bbda41d3677
- https://en.wikipedia.org/wiki/Markov_model
- https://greentec.github.io/reinforcement-learning-second/
- Human-level control through deep reinforcement learning, Nature
- 파이썬과 케라스로 배우는 강화학습, 위키북스
- 수학으로 풀어보는 강화학습 원리와 알고리즘, 위키북스
- 강화학습 / 심층강화학습 특강, 위키북스

감사합니다

인천대학교
INCHEON NATIONAL UNIVERSITY



INU 인천대학교 INCHEON NATIONAL UNIVERSITY