

# Reinforcement Learning

Introduction to Reinforcement Learning and  
Natural Language Processing

Jeiyoon Park

Department of Computer Science and Engineering



고려대학교  
KOREA UNIVERSITY



Natural Language  
Processing  
& Artificial Intelligence

**IPA** Korea IT Business Promotion Association  
한국IT비즈니스진흥협회

# Outline

---

- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Outline

---

- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Overview

---

## 1. 강화학습이란?



# Overview

## 1. 강화학습이란?



강화학습이란 에이전트(Agent)가 환경(Environment)으로 부터 얻어지는 보상정보(Reward)를 통해 좋은 행동을 점점 더 많이 하도록 하는 학습 방법이다.

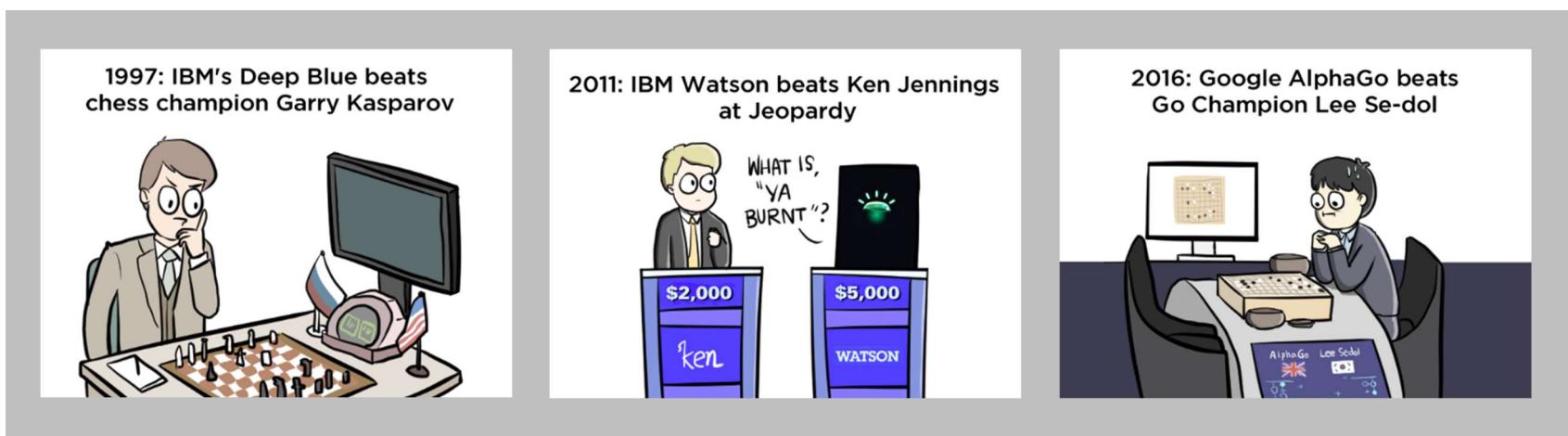
# Overview

---

## 2. 강화학습의 장점

- (1) 기존의 방식으로 풀기 어려웠던 복잡한 문제들을 해결할 수 있다.
- (2) 학습 데이터가 없어도 경험으로부터 학습할 수 있다.
- (3) 인간의 학습방법과 굉장히 유사하다. 따라서 학습이 직관적이고 완벽함을 추구하며 현실세계의 문제들을 해결할 수 있다.

실제로 복잡한 문제들을 해결할 때 사람보다 더 잘할 수 있다.



# Overview

---

## # 진단 평가

- 1) 강화학습이란?
- 2) 에이전트란?
- 3) 환경이란?
- 4) 보상이란?
- 5) 관찰결과란? 관찰결과의 역할은?

# Outline

---

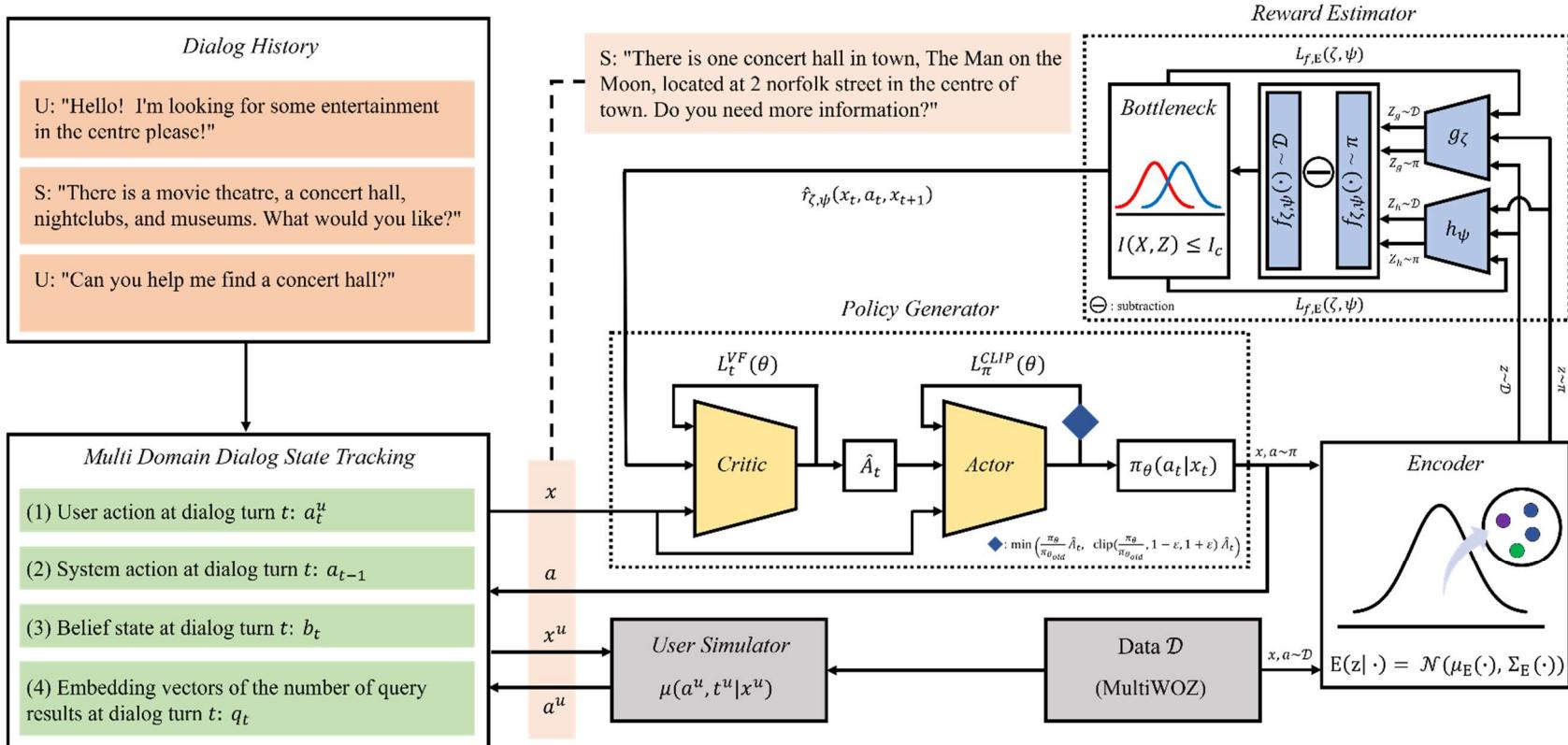
- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Applications

## 1. Task-Oriented Dialog System

- Variational Reward Estimator Bottleneck: Learning Robust Reward Estimator for Multi-Domain Task-Oriented Dialog

(J Park et al., Arxiv 2020)



# Applications

## 2. Vision Language

- THE NEURO-SYMBOLIC CONCEPT LEARNER: INTERPRETING SCENES, WORDS, AND SENTENCES FROM NATURAL SUPERVISION (Mao et al., ICLR 2019.)

### I. Learning basic, object-based concepts.



Q: What's the color of the object?

A: Red.

Q: Is there any cube?

A: Yes.



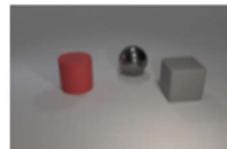
Q: What's the color of the object?

A: Green.

Q: Is there any cube?

A: Yes.

### II. Learning relational concepts based on referential expressions.



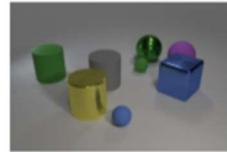
Q: How many objects are right of the red object?

A: 2.

Q: How many objects have the same material as the cube?

A: 2

### III. Interpret complex questions from visual cues.



Q: How many objects are both right of the green cylinder and have the same material as the small blue ball?

A: 3

**Optimization objective.** The optimization objective of NS-CL is composed of two parts: concept learning and language understanding. Our goal is to find the optimal parameters  $\Theta_v$  of the visual perception module Perception (including the ResNet-34 for extracting object features, attribute operators, and concept embeddings) and  $\Theta_s$  of the semantic parsing module SemanticParse, to maximize the likelihood of answering the question  $Q$  correctly:

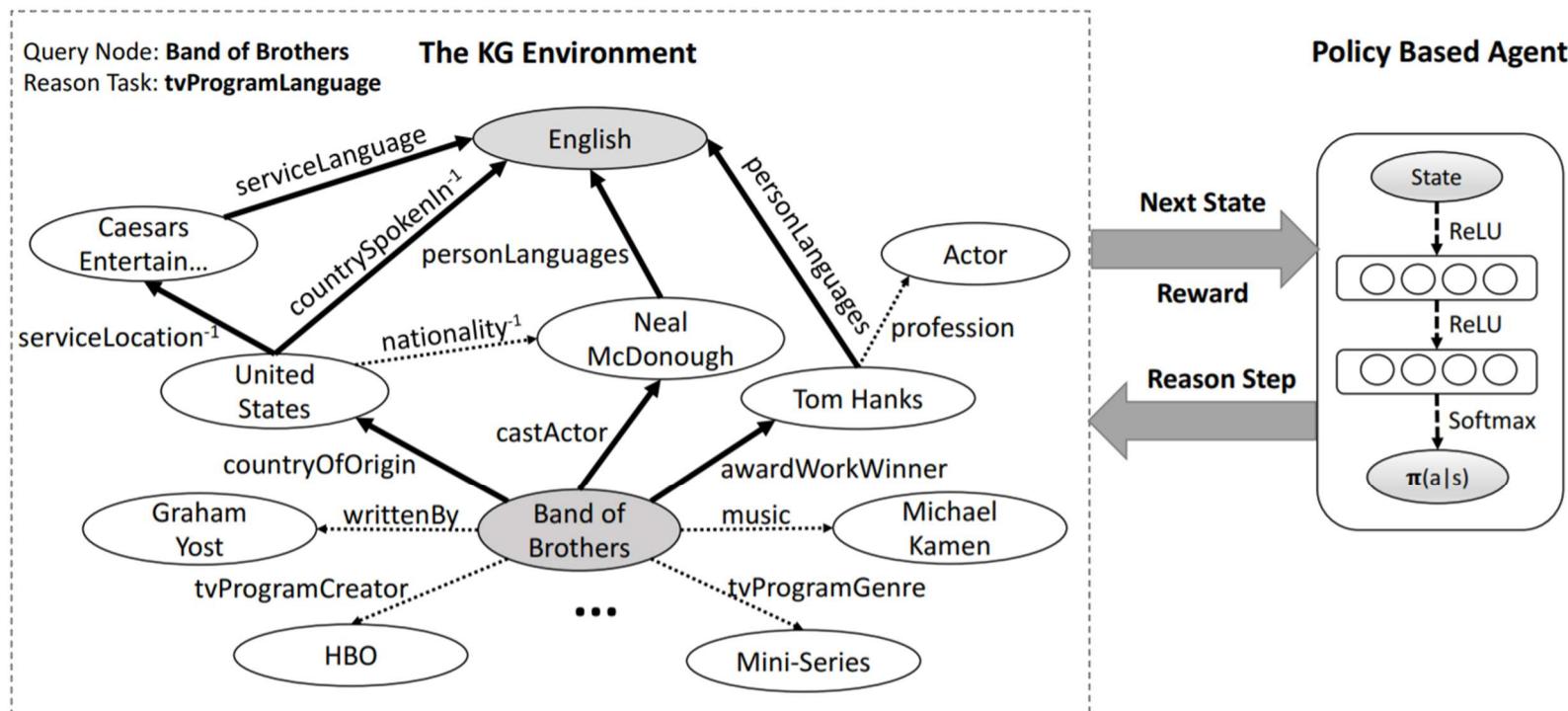
$$\Theta_v, \Theta_s \leftarrow \arg \max_{\Theta_v, \Theta_s} \mathbb{E}_P [\Pr[A = \text{Executor}(\text{Perception}(S; \Theta_v), P)]], \quad (1)$$

where  $P$  denotes the program,  $A$  the answer,  $S$  the scene, and Executor the quasi-symbolic executor. The expectation is taken over  $P \sim \text{SemanticParse}(Q; \Theta_s)$ .

# Applications

## 3. Knowledge Graph Reasoning

- DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning (Wenhan et al., EMNLP 2017.)



$$r_{\text{GLOBAL}} = \begin{cases} +1, & \text{if the path reaches } e_{\text{target}} \\ -1, & \text{otherwise} \end{cases}$$

# Outline

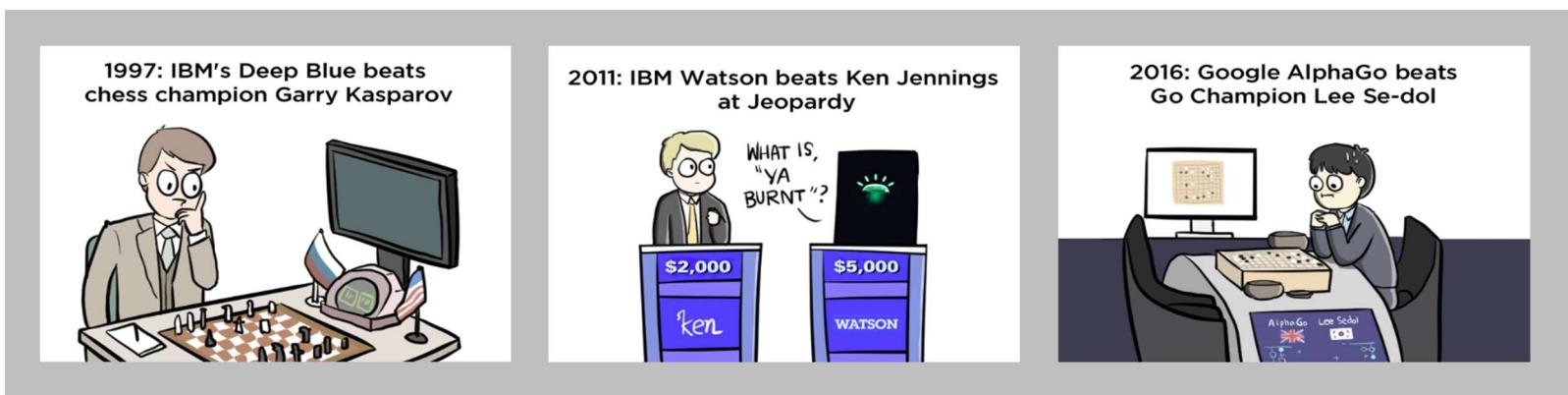
---

- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Basics (1/6)

## 1. Markov Decision Process

- **Markov Decision Process(MDP)**란 강화학습 같은 순차적으로 행동을 결정하는 문제를 정의할 때 사용하는 방법
- MDP의 구성요소는 크게 다섯가지임. 상태(state), 행동(action), 보상함수(reward), 상태변환확률(state transition probability), 감가율(discount factor)
- **모든 강화학습은 MDP를 “사용자”가 정의하는 것 부터 시작**



# Basics (1/6)

---

## 1. Markov Decision Process

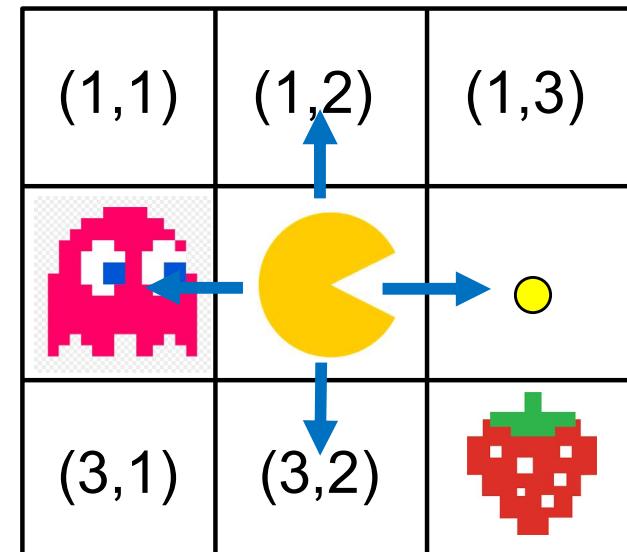
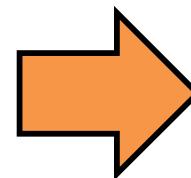
- ex) Pac Man



# Basics (1/6)

## 1. Markov Decision Process

- ex) Pac Man



- 상태(state) :  $S = \{(1,1), \dots, (3,3)\}$

- 행동(action) :  $A = \{\leftarrow, \uparrow, \rightarrow, \downarrow\}$

- 보상함수(reward) :  $R = \{+1 \text{ or } +10 \text{ or } -1\}$

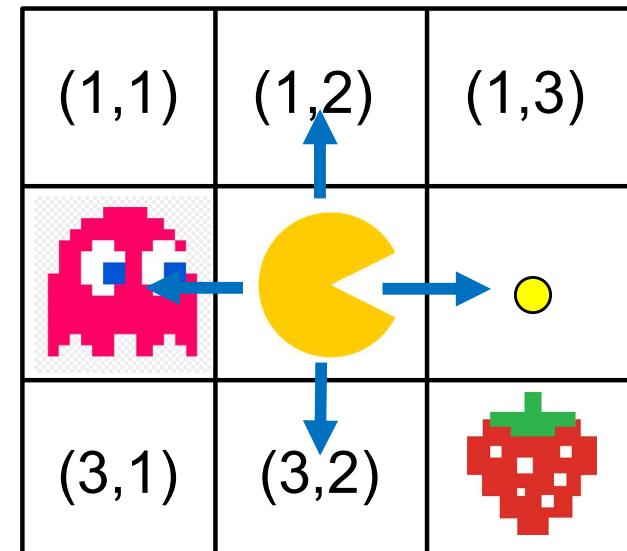
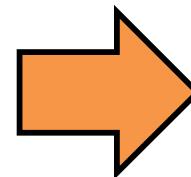
$$R_s^a = E[R_{t+1} | S_t = s, A_t = a]$$



# Basics (1/6)

## 1. Markov Decision Process

- ex) Pac Man



- 상태변환확률(state transition probability) : 팩맨이 (2, 2)에 있을때, (2, 3)에 있는 공을 먹으러 갈 확률

- 감가율(discount factor) : 보상이 모두 똑같다면 지금 먹은 공과 시간이 흐른 뒤 먹은 공을 구분할 수 없음. 또한 딸기(+10)를 먹는 것과 공을 열개(+10)먹는 것을 구분할 수 없음. 따라서 보상에 가감율을 곱해줌.

# Basics (1/6)

---

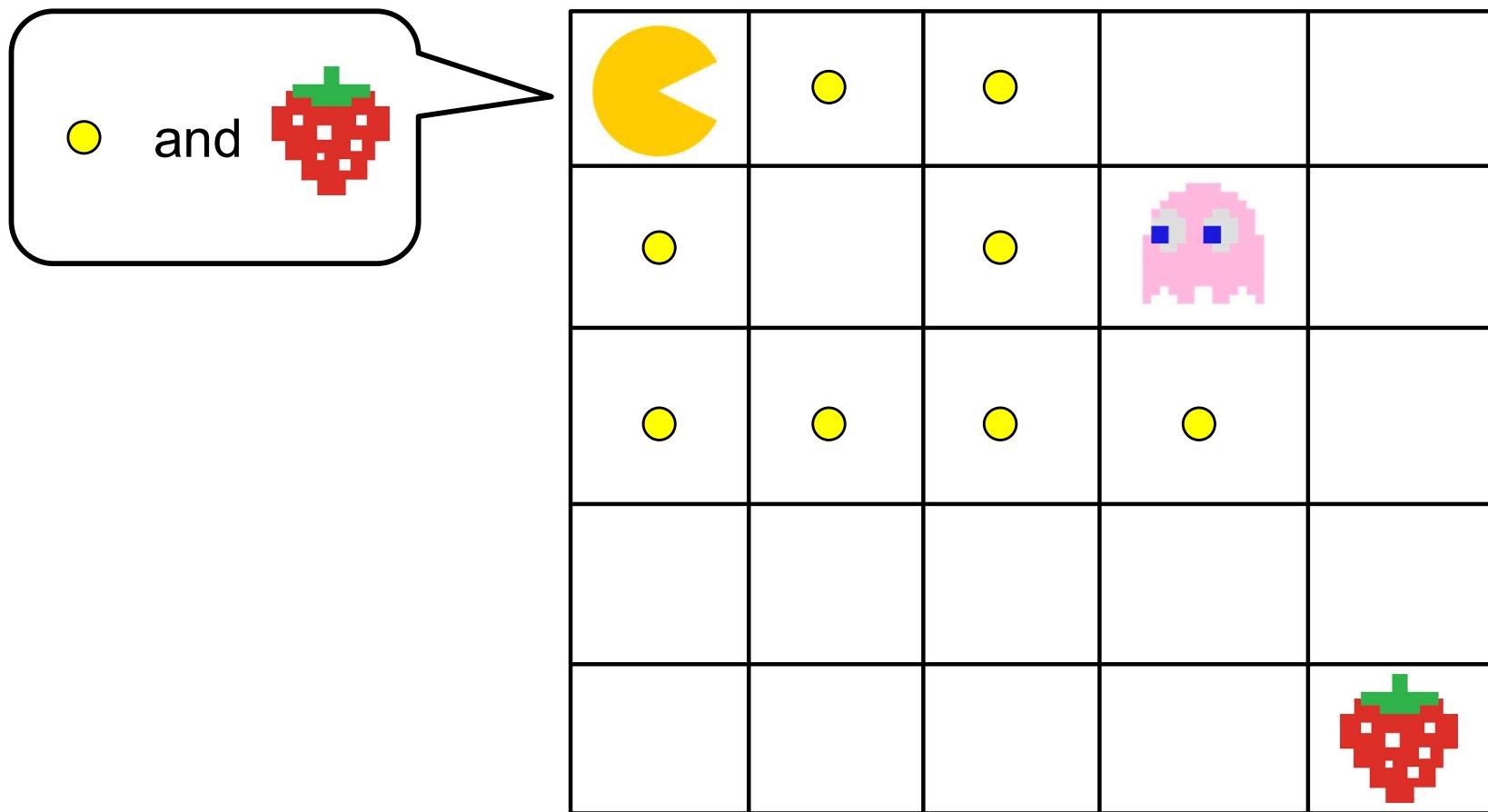
## 1. Markov Decision Process

- 상태(state) : 에이전트가 관찰 가능한 상태의 집합
- 행동(action) : 에이전트가 특정 상태에서 할 수 있는 행동의 집합
- 보상함수(reward) : 환경이 에이전트에게 주는 정보. 에이전트가 학습할 수 있는 유일한 정보
- 상태변환확률(state transition probability) : 에이전트가 어떠한 상태  $s$ 에서 행동  $a$ 를 해서 다른 상태  $s'$ 에 도달할 확률
- 감가율(discount factor) : 같은 보상이면 나중에 받을 수록 가치가 떨어짐. 이를 수학적으로 표현하기 위한 개념

# Basics (2/6)

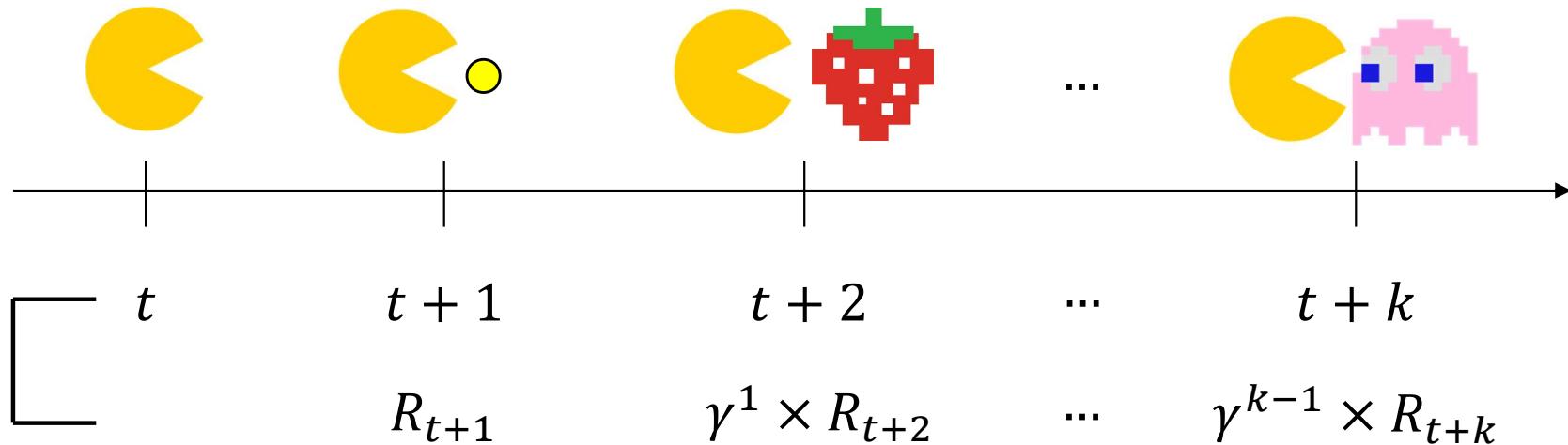
## 2. Value Function

에이전트(팩맨) 입장에서는 어떤 행동을 하는 것이 좋은지 어떻게 알까? 아직 받지 않은 보상들을 어떻게 고려하고 행동할 수 있을까?



# Basics (2/6)

## 2. Value Function



- **반환값(return):** 에이전트가 실제로 환경을 탐험하며 받은 보상

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

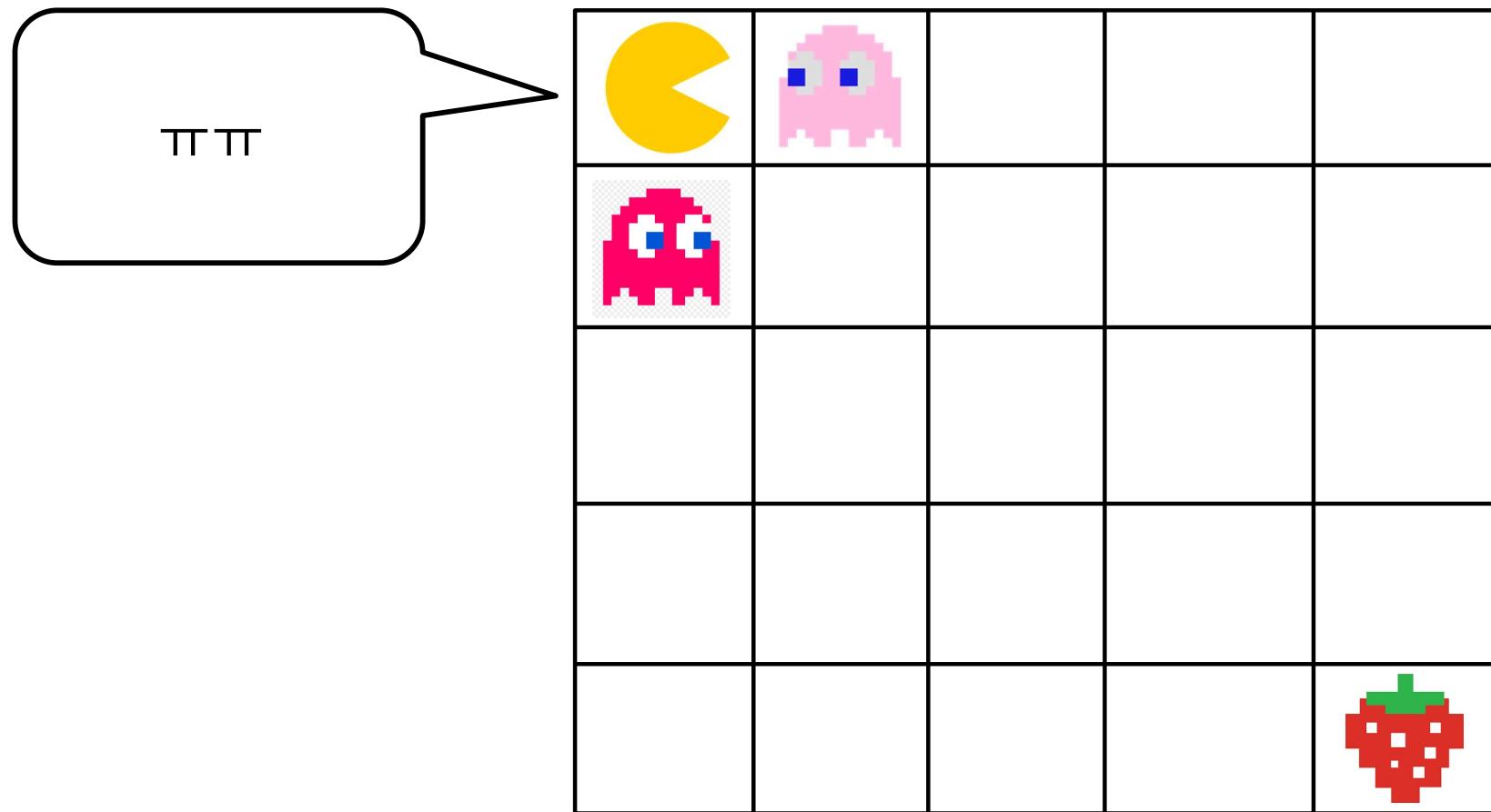
- **가치함수(value function):** 에이전트가 얼마의 보상을 받을 것인지에 대한 기댓값. 매 시행마다 값이 다르기 때문에 기댓값 사용.

$$v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

# Basics (2/6)

## 2. Value Function

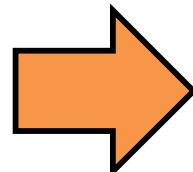
즉, 가치함수란 **어떠한 상태**에 있을때 이 상태에 있으면 앞으로  
얼마의 **보상**을 받을 지에 대한 **기댓값**이다.

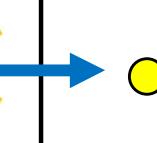
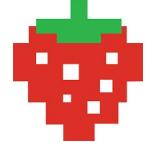


# Basics (3/6)

## 3. Policy

- 정책(policy)이란 모든 상태에서 에이전트가 할 행동



(1,1)	(1,2)	(1,3)
		
(3,1)	(3,2)	

$$\pi(a|s) = P[A_t = a | S_t = s]$$

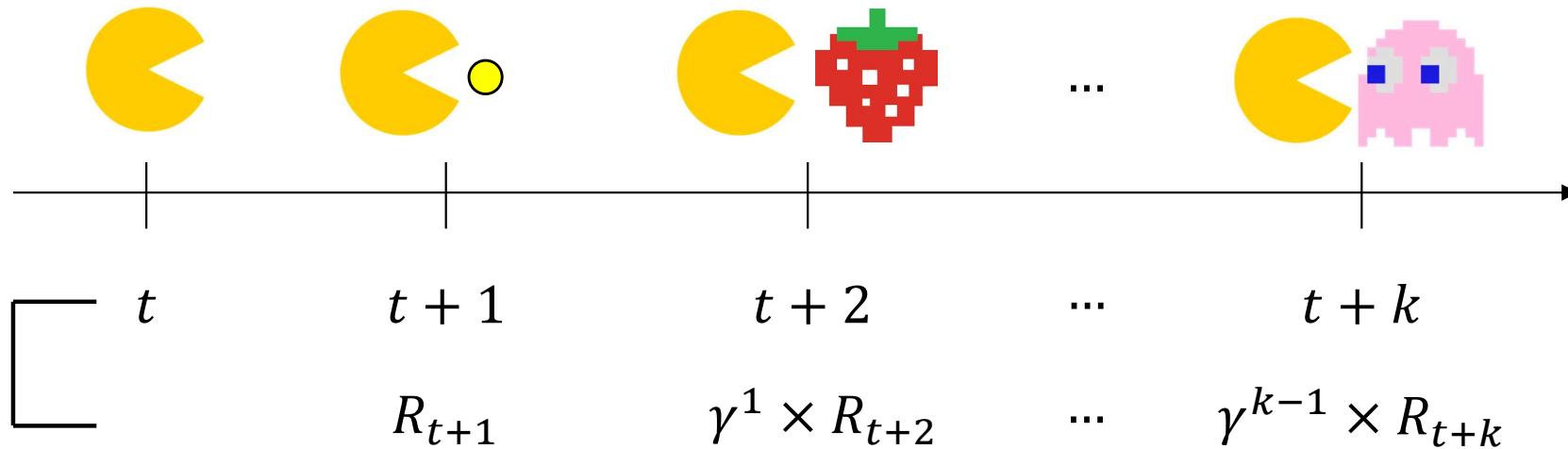
- ex)  $\pi(a = \text{왼쪽} | s = (2,2)) = 0.1$

$$\pi(a = \text{오른쪽} | s = (2,2)) = 0.9$$

# Basics (4/6)

## 4. Bellman Expectation Equation

벨만 기대 방정식이란 현재 상태의 가치함수와 다음상태의 가치함수 사이의 관계를 말해주는 방정식이다.

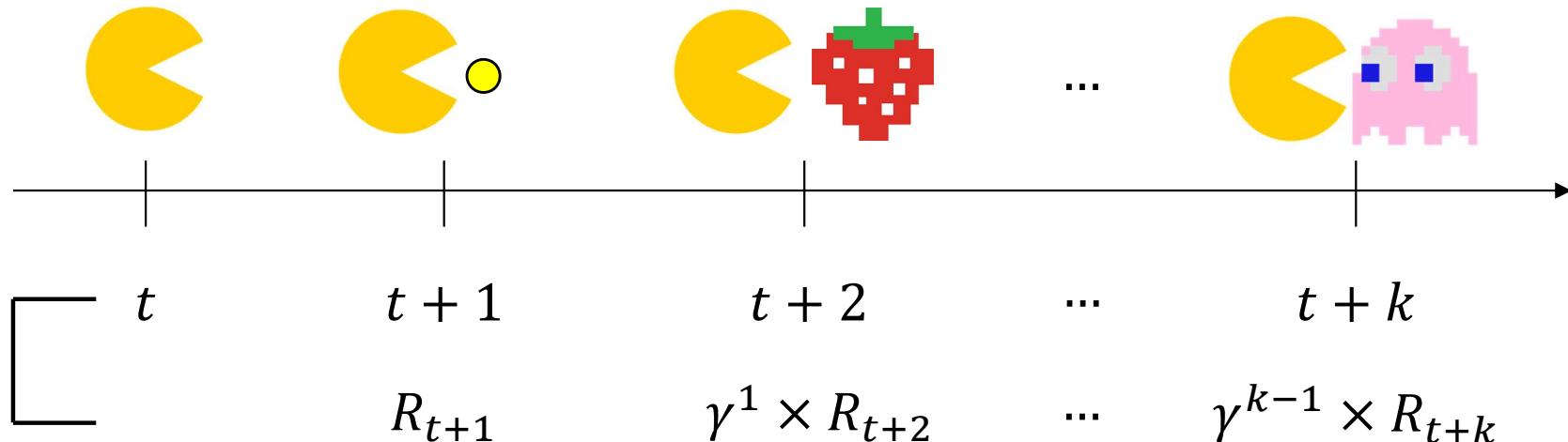


$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

# Basics (4/6)

## 4. Bellman Expectation Equation



$$v(s) = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

$$= E[R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \dots) | S_t = s]$$

$$= E[R_{t+1} + \gamma(G_{t+1}) | S_t = s]$$

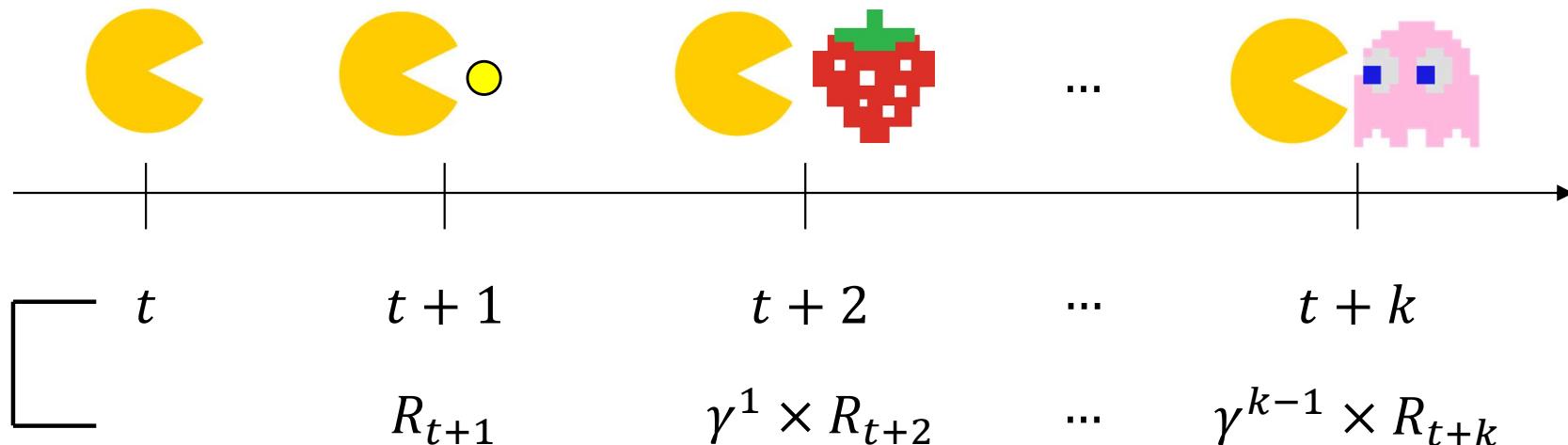
$$= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

반환값이긴 하지만 사실 에이전트가  
실제로 받은 보상이 아직은 아님.  
따라서 가치함수 형태로 나타낼 수 있음

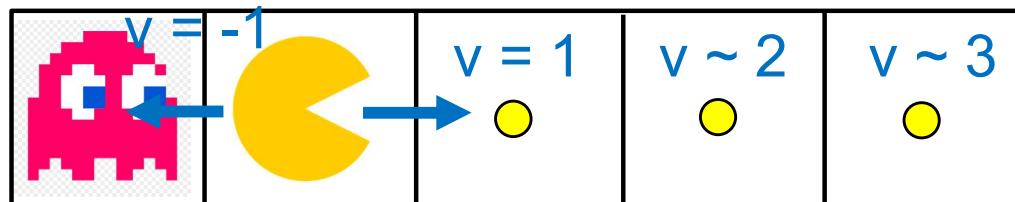
$\therefore v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$  : 현재상태의 가치함수와 다음상태의 가치함수 사이의 관계를 말해줌

# Basics (4/6)

## 4. Bellman Expectation Equation



$$\therefore v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$



# Basics (4/6)

## 4. Bellman Expectation Equation

벨만 기대방정식을 계산할때는 상태에서의 가치함수 뿐만 아니라 정책(에이전트가 어떤 행동을 할지에 대한 확률) 또한 고려해야 함.

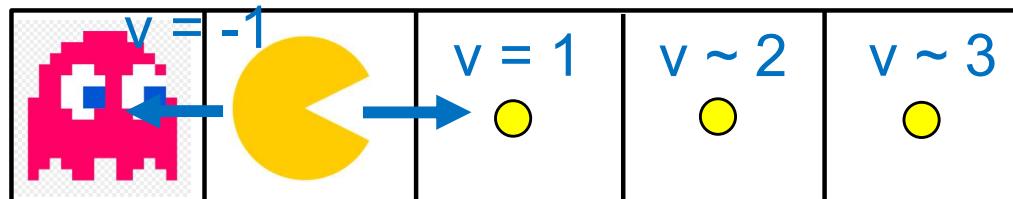
- ex)  $\pi(a = \text{왼쪽} | s = (2,2)) = 0.1$

$$\pi(a = \text{오른쪽} | s = (2,2)) = 0.9$$

- $\square \pi(a|s) = P[A_t = a | S_t = s]$

- $\square v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$

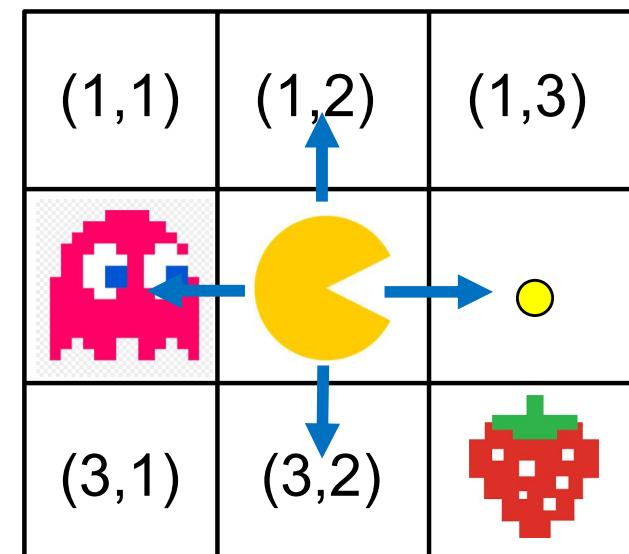
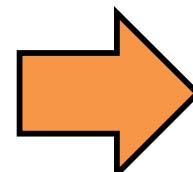
$$\therefore v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$



# Basics (5/6)

## 5. Q-Function

- 가치함수(value function)는 어떤 ‘상태’에 대한 보상의 기댓값
- 큐함수(q-function)는 어떤 ‘상태’에서 어떤 ‘행동’이 얼마나 좋은지 알려주는 함수. 행동 가치함수라고도 함.
- 따라서 큐함수는 상태와 행동이라는 두가지 변수를 가짐



# Basics (5/6)

---

## 5. Q-Function

- 어떠한 상태에 있을때 어떠한 보상을 얻을 지 아는 것도 중요하지만 어떠한 행동을 했을때 어떠한 보상을 얻을 지, 즉 행동 가치함수의 기댓값을 아는 것도 중요하다.

$$\begin{aligned} q_{\pi}(s, a) &= E[R_{t+1} + \gamma q_{\pi}(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \\ \pi(a|s) &= P[A_t = a | S_t = s] \end{aligned}$$

- 특히 각 행동에 대해 가치함수를 계산하여 정보를 가져올 수 있으면 굳이 상태에 대한 가치함수를 계산할 필요가 없다.

$$v(s) = \sum_{a \in A} \pi(a|s) q_{\pi}(s, a)$$

# Basics (6/6)

---

## 6. Bellman Optimality Equation

- 그렇다면 가치함수의 역할은 뭘까? 정책을 정하고 그 정책을 따라갔을 때 받는 보상들의 합인 가치함수로 더 좋은 정책을 찾아내는 것
- 그렇다면 최적의 가치함수는 어떻게 구할 수 있을까?

$$v_*(s) = \max_{\pi} [v_{\pi}(s)]$$

$$= E[R_{t+1} + \gamma \max_{a'} q_*(s_{t+1}, a') | S_t = s, A_t = a]$$

# Basics (6/6)

---

## # 진단 평가

- 1) MDP의 다섯가지 요소는?
- 2) 가치함수란? 필요한 이유는?
- 3) 정책이란?
- 4) 벨만 기대 방정식은? 의미는?
- 5) 큐함수란?
- 6) 최적의 가치함수는 어떻게 구할까? 필요한 이유는?

# Outline

---

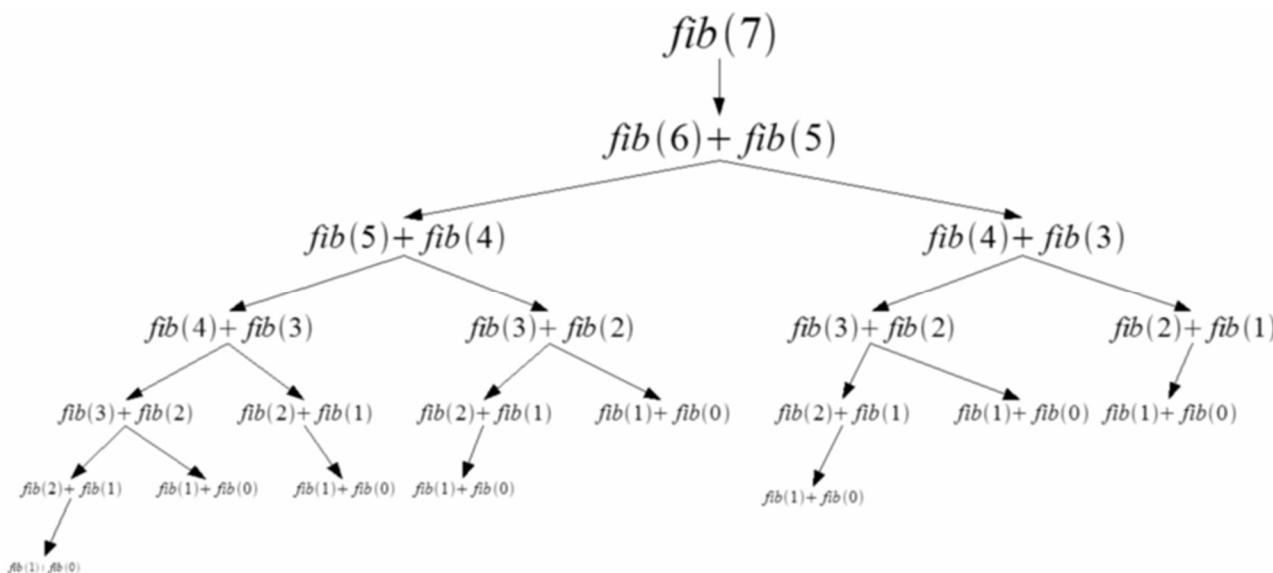
- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Dynamic Programming

## 1. 동적 프로그래밍 이란?

- 동적 → 기억하기
- 프로그래밍 → (순차적 프로세스에 대한) 테이블 만들기
- 큰 문제를 한 번에 해결하기 힘들 때 작은 여러 개의 문제로 나누어서 푸는 기법

e.g.) 피보나치 수열



# Dynamic Programming

## 2. 그렇다면 무엇을 어떻게 작은문제로 나누어서 풀까?

- (1) 우리가 구하고 싶은것: 각 상태의 가치함수(i.e.  $v_\pi(s_n)$ )
- (2) 이때, 모든 상태에 대해 가치함수를 구하고 iteration을 돌며 각 상태에 대한 가치함수를 업데이트 한다.

	$s_2$	$s_3$	$s_4$	$s_5$
$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$
$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{15}$
$s_{16}$	$s_{17}$	$s_{18}$	$s_{19}$	$s_{20}$
$s_{21}$	$s_{22}$	$s_{23}$	$s_{24}$	

각각의 모든 상태에 대해  
진짜 가치함수  $v_\pi(s)$  를  
구하는 것이 목표!

# Dynamic Programming

## 3. 가치함수가 구해지는 과정 (벨만방정식 푸는 과정)

	$s_2$	$s_3$	$s_4$	$s_5$		iteration	가치함수 ( $v_\pi$ )
$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$		1	$v_0(s_1), \dots, v_0(s_{25})$
$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{15}$			
$s_{16}$	$s_{17}$	$s_{18}$	$s_{19}$	$s_{20}$			
$s_{21}$	$s_{22}$	$s_{23}$	$s_{24}$				

$$v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

# Dynamic Programming

## 3. 가치함수가 구해지는 과정 (벨만방정식 푸는 과정)

	$s_2$	$s_3$	$s_4$	$s_5$		iteration	가치함수 ( $v_\pi$ )
$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$		1	$v_0(s_1), \dots, v_0(s_{25})$
$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{15}$		2	$v_1(s_1), \dots, v_1(s_{25})$
$s_{16}$	$s_{17}$	$s_{18}$	$s_{19}$	$s_{20}$			
$s_{21}$	$s_{22}$	$s_{23}$	$s_{24}$				

$$v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

# Dynamic Programming

## 3. 가치함수가 구해지는 과정 (벨만방정식 푸는 과정)

	$s_2$	$s_3$	$s_4$	$s_5$		iteration	가치함수 ( $v_\pi$ )
$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$		1	$v_0(s_1), \dots, v_0(s_{25})$
$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{15}$		2	$v_1(s_1), \dots, v_1(s_{25})$
$s_{16}$	$s_{17}$	$s_{18}$	$s_{19}$	$s_{20}$		...	...
$s_{21}$	$s_{22}$	$s_{23}$	$s_{24}$			k	$v_\pi(s_1), \dots, v_\pi(s_{25})$

$$v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

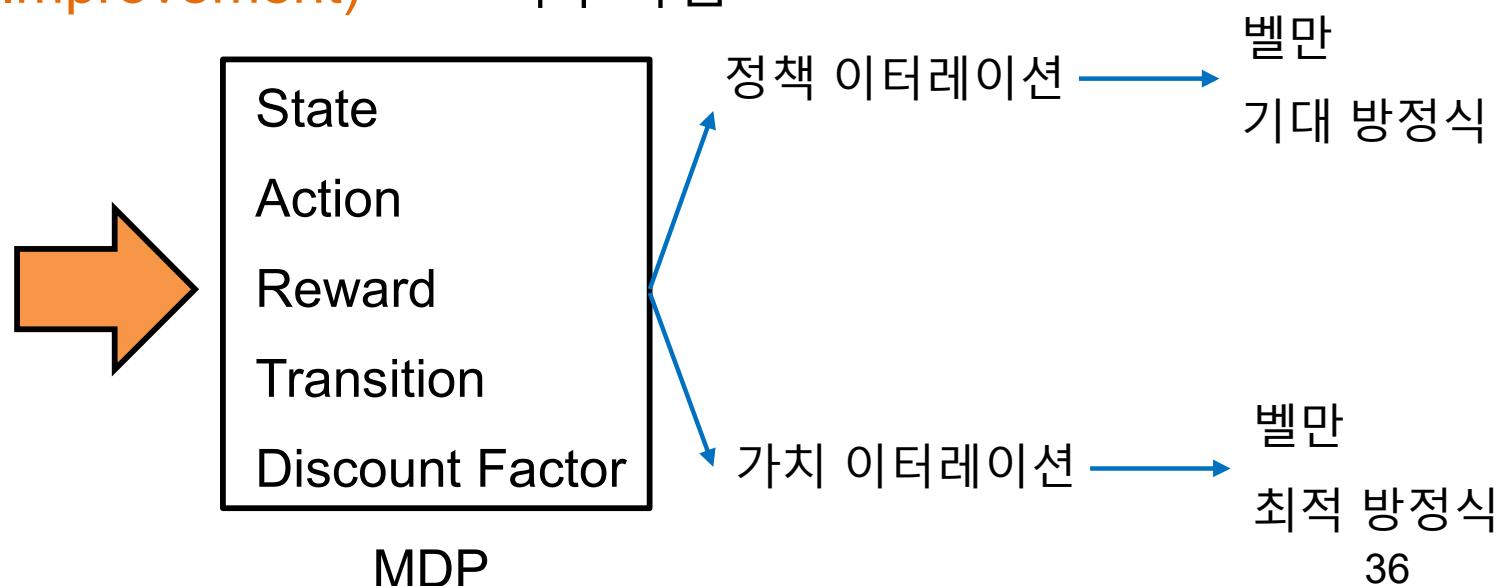
# Dynamic Programming

## 4. 정책 이터레이션: 정책 평가와 정책 발전

- 핵심은 다이나믹 프로그래밍으로 벨만방정식을 풀어서 가치함수를 구하는 과정을 이해하는 것!
- 이 과정을 **정책 이터레이션**이라고 함
- 처음에는 무작위 행동을 한 후 이터레이션을 돌며 가치함수를 최적화 시켜나감
- 위와 같은 최적화 과정은 **정책 평가(Policy Evaluation)**와 **정책 발전(Policy Improvement)**으로 나누어짐



순차적인 행동을  
결정해야되는 문제



# Dynamic Programming

## 5. 정책 평가 (from 정책 이터레이션)

- 어떤 정책(Policy)이 있을때 그 정책을 정책 평가를 통해 얼마나 좋은지 판단하고 그 결과를 기준으로 더 좋은 정책으로 발전시킴
- 그렇다면 정책을 어떻게 평가할 수 있을까?
- 그 근거는 바로 다이나믹 프로그래밍으로 구한 각각의 상태에 대한 가치함수가 된다!

$$v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

- 위 수식은 각각의 상태에 대한 아주 먼 미래까지 고려해야하기 때문에 계산량이 급격하게 늘어남
- 하지만 다이나믹 프로그래밍을 통해 이터레이션을 돌며 가치함수를 최적화 할 수 있음!

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

# Dynamic Programming

## 5. 정책 평가 (from 정책 이터레이션)

- 벨만 “기대”방정식은 가치함수에 대한 기댓값 형태(확률곱)으로 나타낼 수 있다.

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

$$= \sum_{a \in A} \pi(a|s)(R_{t+1} + \gamma v_{\pi}(s'))$$

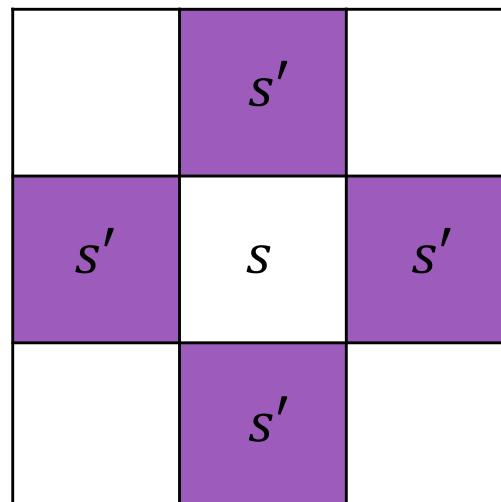
$$\therefore v_{\pi}^{n+1}(s) = \sum_{a \in A} \pi(a|s)(R_{t+1} + \gamma v_{\pi}^n(s'))$$

# Dynamic Programming

## 5. 정책 평가 (from 정책 이터레이션)

(1) 현재 상태  $s$ 에서 갈 수 있는 다음 상태  $s'$ 에 대한 가치함수를 불러옴 ( $v_{\pi}^{n+1}(s')$ , 보라색 부분 중 하나)

(2) 가치함수에 감가율을 곱하고 그 상태로 가는 것에 대한 보상을 더함 ( $R_{t+1} + \gamma v_{\pi}^n(s')$ )



Iteration n

# Dynamic Programming

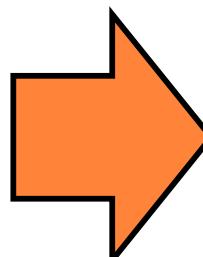
## 5. 정책 평가 (from 정책 이터레이션)

(3) (2)에서 구한 값에 그 행동을 취할 확률(정책)을 곱하여 기댓값 형태로 나타냄 ( $\pi(a|s)(R_{t+1} + \gamma v_\pi^n(s'))$ )

(4) (3)을 모든 가능한 행동에 대해 반복하고 그 값을 더함

( $\sum_{a \in A} \pi(a|s)(R_{t+1} + \gamma v_\pi^n(s'))$ ). 결과를  $n+1$  가치함수로 사용

	$s'$	
$s'$	$s$	$s'$
	$s'$	



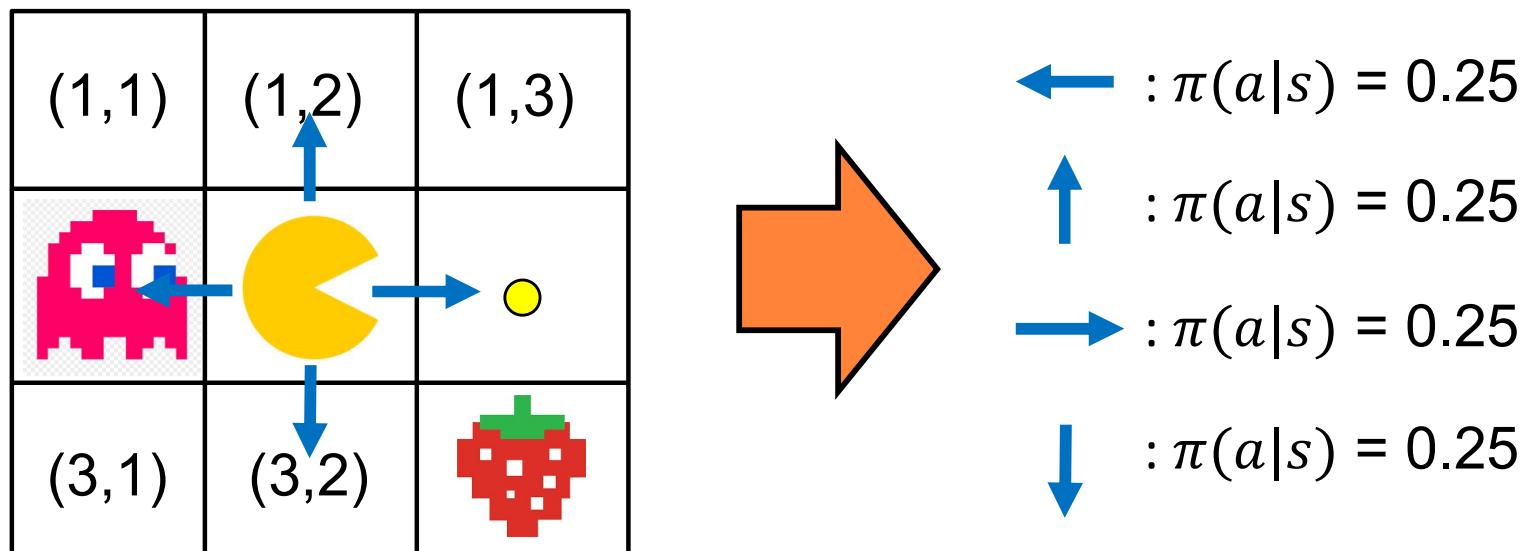

Iteration n

Iteration n+1

# Dynamic Programming

## 6. 정책 발전 (from 정책 이터레이션)

- (1) 정책 평가를 바탕으로 어떻게 정책을 발전시킬 수 있을까?
- (2) 가치함수 최적화 전에는 무작위 행동을 하고 점점 가치함수가 높은 행동을 더 많이 하도록 학습함
- (3) 가장 유명한 방법중 하나인 **탐욕 정책 발전(Greedy Policy Improvement)**를 사용하려고 함

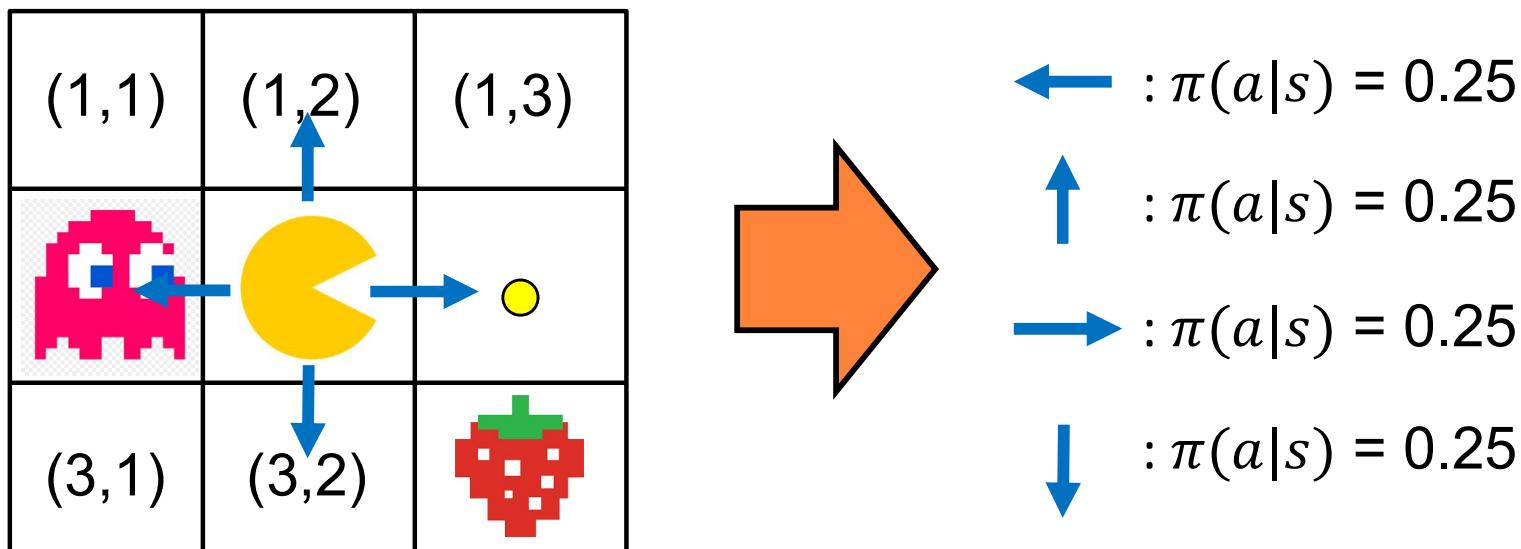


# Dynamic Programming

## 6. 정책 발전 (from 정책 이터레이션)

(4) 정책에 대한 평가를 거치면 큐함수(Q-function)을 이용하여 행동에 대한 가치함수를 알 수 있음

$$\begin{aligned} q_{\pi}(s, a) &= E[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = a] \\ &= R_{t+1} + \gamma v_{\pi}(S_{t+1}) \end{aligned}$$

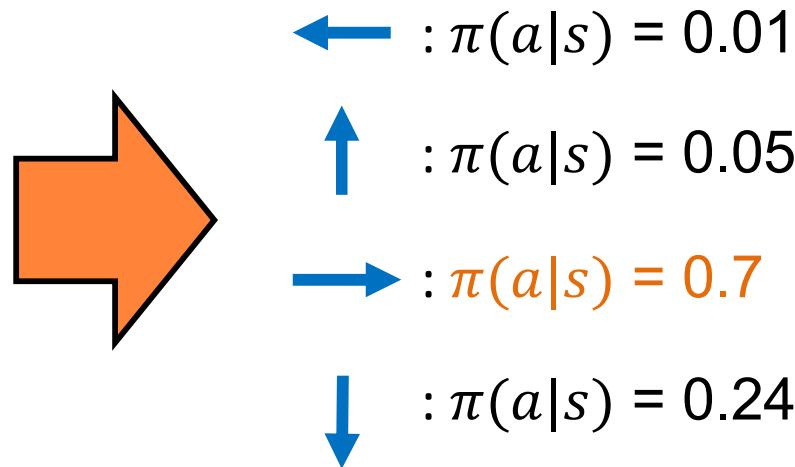
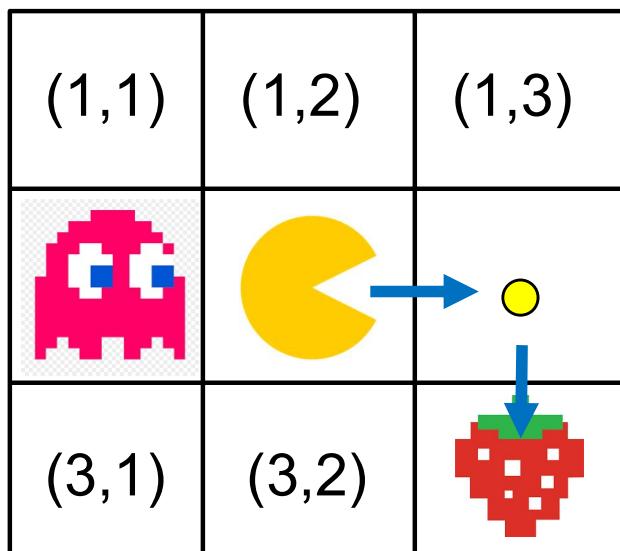


# Dynamic Programming

## 6. 정책 발전 (from 정책 이터레이션)

(5) 상태  $s$ 에서 큐함수들을 비교하여 가장 큰 큐함수를 가지는 행동, 즉 행동 가치함수값이 가장 높은 행동을 선택함. 따라서 더 높은 보상을 주는 행동을 반복하도록 학습됨.

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}} q_{\pi}(s, a)$$



# Dynamic Programming

## 7. 가치 이터레이션

(1) 가치 이터레이션(Value iteration)이란 현재의 가치함수가 최적은 아니지만 최적이라는 전제하에 각 상태에 대한 가치함수를 업데이트하는 방법 (벨만최적방정식 사용)

(2) 벨만 기대방정식은 기댓값 형태이기 때문에 정책을 고려했었음. 하지만 벨만 최적방정식에서는 현재 상태에서 가능한 최고의 가치함수 값을 고려하면 됨.

$$v_{n+1}(s) = \max_{a \in A} (R_{t+1} + \gamma v_n(s'))$$

# Dynamic Programming

## 8. 동적 프로그래밍의 한계

- (1) 계산 복잡도 (i.e. 5x5 그리드 월드가 아니라  $n \times n$ 이라면?)
- (2) 차원의 저주 (i.e. 그리드 월드처럼 2차원이 아니라  $n$ 차원이라면?)
- (3) 환경에 대한 완벽한 정보를 알아야 한다 (우리는 실제로 세상을 탑뷰로 바라보며 모든 환경에 대한 정보를 인지하고 있는가?)

# Dynamic Programming

## # 진단 평가

- 1) 동적 프로그래밍이란? 장점은?
- 2) 동적 프로그래밍으로 풀고자 하는 문제는?
- 3) 가치함수가 업데이트 되는 과정을 설명할 수 있는가?
- 4) 정책 이터레이션이란? 정책 평가와 정책 발전은 각각 무엇을 의미하는가?
- 5) 정책 이터레이션과 가치 이터레이션의 차이는?
- 6) 동적 프로그래밍의 한계는?

# Outline

---

- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Q-Learning

---

## 1. 학습 방법

- 사람은 바둑을 어떻게 둘까? 다이나믹 프로그래밍때처럼 한칸한칸 모든 경우의 수를 매 턴마다 생각하면서 둘까?
- 많은 경우가 일단 해보고 복기를 하고 복기 내용을 바탕으로 학습하여 더욱 잘해지는 과정을 반복한다.
- 강화학습은 사람의 학습방법처럼 겪은 경험으로부터 가치함수를 업데이트 함

2016: Google AlphaGo beats  
Go Champion Lee Se-dol



# Q-Learning

---

## 2. 예측과 제어

- 에이전트는 환경과 상호작용을 통해 주어진 정책에 대한 가치함수를 학습할 수 있음. 이를 예측(prediction)이라고 함  
(앞에서 얘기했던 정책 평가에 해당)

e.g.) 몬테카를로 예측, 시간차 예측

- 또한 가치함수를 토대로 정책을 끊임없이 발전시켜 나가 최적의 정책을 학습할 수 있음. 이를 제어(control)라고 함  
(앞에서 얘기했던 정책 발전에 해당)

e.g.) SARSA

# Q-Learning

---

## 2. 예측과 제어

- 즉, 예측은 현재 정책을 따랐을 때 참 가치함수를 구하는 과정.  
= ‘정책 평가’라고도 함. 왜냐하면 이 정책을 따랐을 때 보상의 합인 가치함수가 얼마인지 나오고 그거에 따라 정책이 좋은지 나쁜지 평가하기 때문.
- 예측의 결과로 정책을 발전 시키는 것을 제어라고 함.  
= ‘정책 발전’이라고도 함.
- 정책 평가와 정책 발전을 번갈아 가며 진행하는 것을 통해 학습함.
- 그렇다면 예측(가치함수를 구하는 과정)은 어떻게 이루어질까?

# Q-Learning

---

## 3. 몬테카를로 예측

### (1) 몬테카를로 한줄 요약: 일단 해보자

ex) 원의 넓이 =  $\pi r^2$

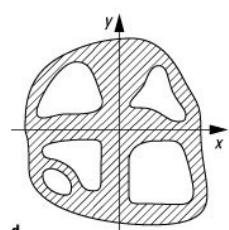
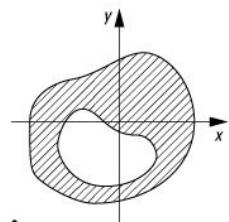
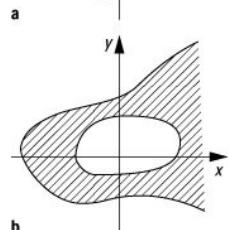
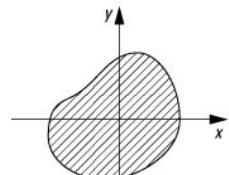
하지만, 원의 넓이 공식을 모른다면?

# Q-Learning

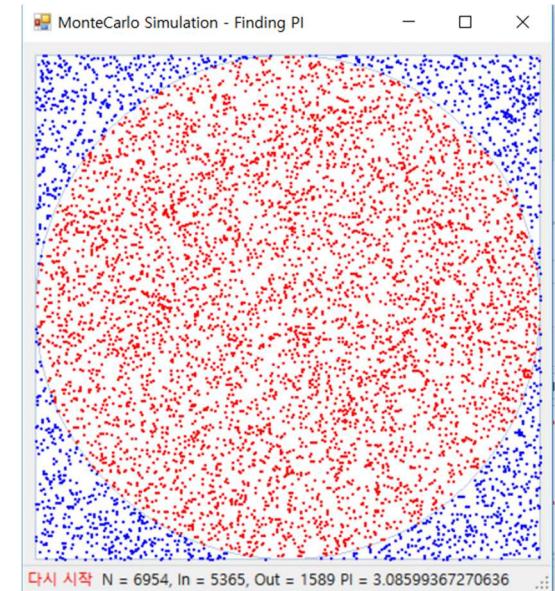
## 3. 몬테카를로 예측

### (1) 몬테카를로 한줄 요약: 일단 해보자

ex) 원의 넓이 =  $\pi r^2$



하지만, 원의 넓이 공식을 모른다면?  
(혹은 방정식이 원이아니라면?)



- 점들을 샘플링, 붉은 점의 갯수를 전체 점의 갯수로 나눔
- 예를들어 전체 점이 6954개이고 빨간색 점이 5365개라면  
원의 넓이는  $5365/6954 = 0.7714$ 로 계산
- 샘플링 한 점의 갯수가 무한대로 가면 원의 넓이에 수렴

# Q-Learning

---

## 3. 몬테카를로 예측

### (2) 샘플링과 몬테카를로 예측

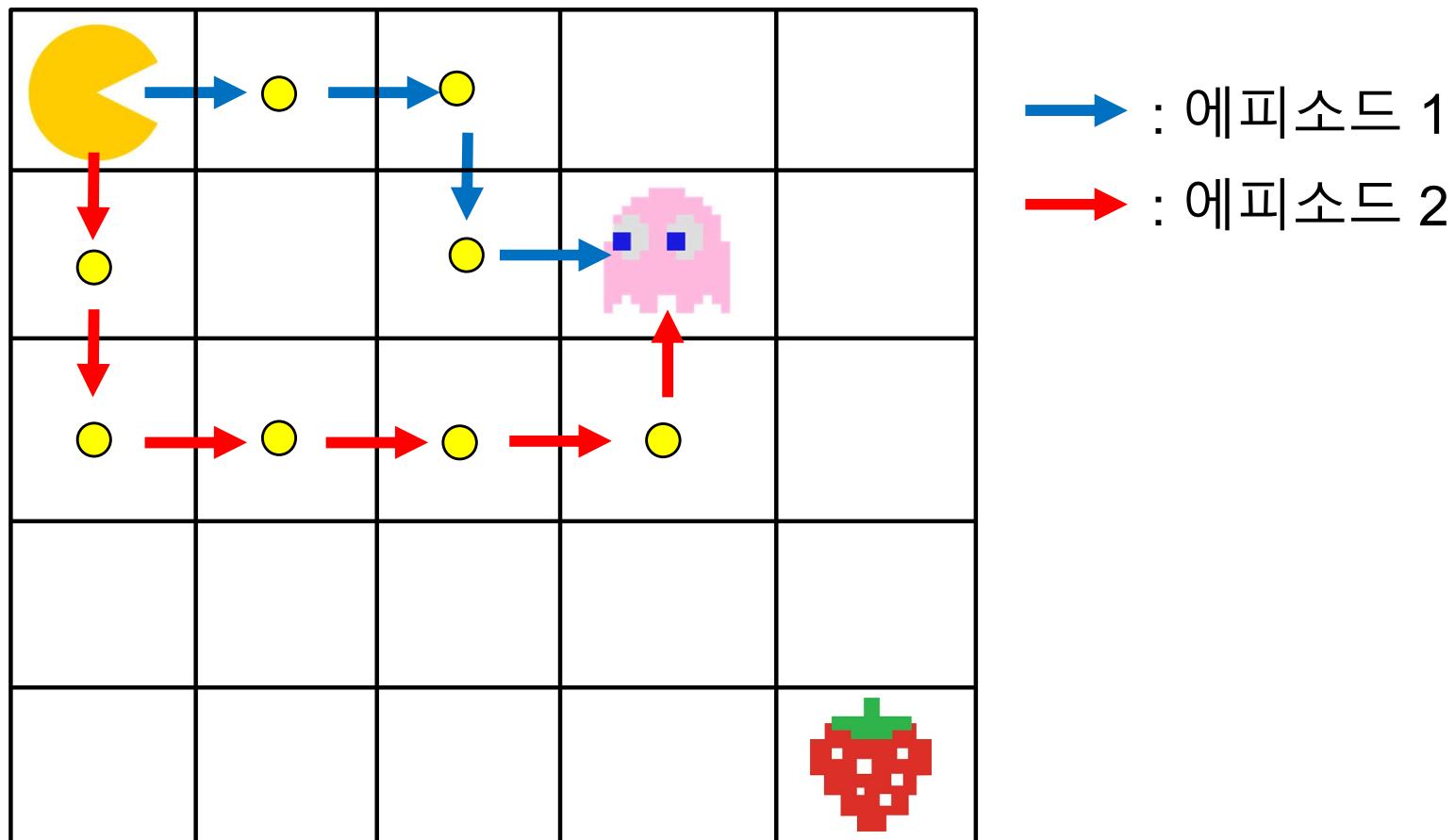
- 앞에서 본 것처럼 가치함수에 대한 모델을 모르는 경우에도 몬테카를로 예측을 통해 가치함수를 추정하는 것이 가능함
- 가치함수를 추정할 때 에이전트가 환경과 상호작용한 한 에피소드를 샘플링함 (점 하나 뿐만 아니라)
- 여러 번의 샘플링을 통해 가치함수의 기댓값을 계산하지 않고 샘플들의 평균으로 가치함수를 최적화 하려면 (i.e. 예측하려면) 어떻게 해야 할까?

$$v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

# Q-Learning

## 3. 몬테카를로 예측

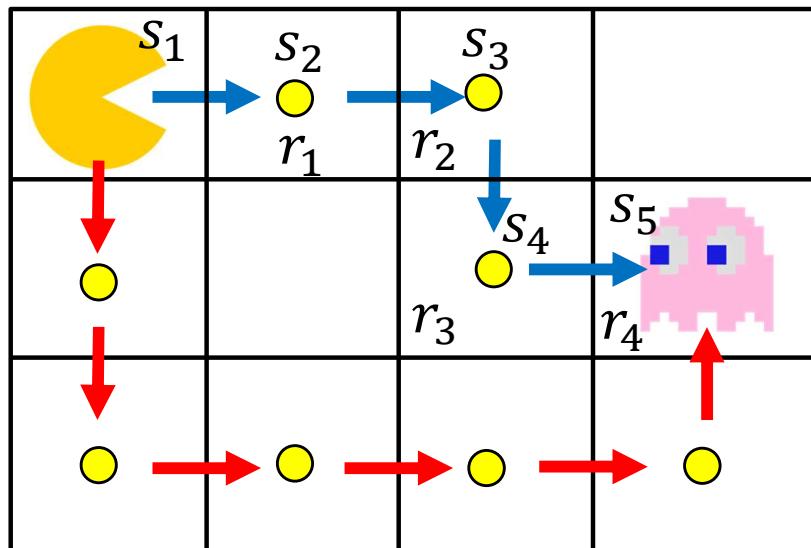
(3) 가치함수를 추정할 때는 에이전트가 환경에서 한 에피소드를 진행 한 것을 샘플링함. (원의 넓이 구할 때 점 하나 뿌리기)



# Q-Learning

---

## 3. 몬테카를로 예측



$$G(s_1) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4$$

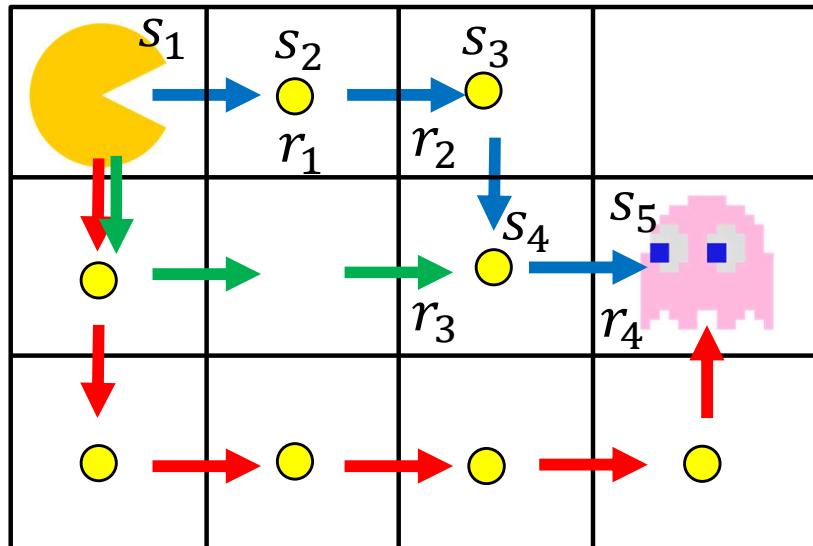
$$G(s_2) = r_2 + \gamma r_3 + \gamma^2 r_4$$

$$G(s_3) = r_3 + \gamma r_4$$

$$G(s_4) = r_4$$

# Q-Learning

## 3. 몬테카를로 예측



$$G(s_1) = r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4$$

$$G(s_2) = r_2 + \gamma r_3 + \gamma^2 r_4$$

$$G(s_3) = r_3 + \gamma r_4$$

$$G(s_4) = r_4$$

$$v_{\pi}(s) \sim \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s)$$

$N(s)$  는 상태  $s$ 를 여러번의 에피소드 동안 방문한 횟수

$G_i(s)$  는 그 상태를 방문한  $i$ 번째 에피소드에서  $s$ 의 반환값

# Q-Learning

---

## 3. 몬테카를로 예측

$$v_{\pi}(s) \sim \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i(s)$$

$$v_{n+1}(s) = \frac{1}{n} \sum_{i=1}^n G_i = \left( G_n + \sum_{i=1}^{n-1} G_i \right)$$

$$= \frac{1}{n} \left( G_n + (n-1) \frac{1}{(n-1)} \sum_{i=1}^{n-1} G_i \right)$$

$$= \frac{1}{n} ( G_n + (n-1) v_n )$$

$$= v_n + \frac{1}{n} (G_n - v_n)$$

$$\therefore V(s) \leftarrow V(s) + \alpha (G(s) - V(s)) \quad \alpha: learning rate$$

# Q-Learning

---

## 3. 몬테카를로 예측

- (4) 몬테카를로 예측에서 에이전트는 이 업데이트 식을 통해  
에피소드 동안 경험한 모든 상태에 대해 가치함수를 업데이트 함
- (5) 샘플 수가 많아질수록 더 정확한 가치함수 최적화가 이루어짐
- (6) 단점은 없을까???

# Q-Learning

---

## 4. 시간차 예측 (Temporal Difference Prediction)

- (1) 몬테카를로 예측의 가장 큰 단점은 실시간이 아니라는 점이다.  
다시말해, 한 에이전트의 한 에피소드가 끝나기 전까지는  
가치함수의 업데이트를 할 수 없다.
- (2) 만약 한 에피소드가 정말 길어진다거나 끝이 없다면 몬테카를로  
예측은 사용할 수 없게된다.



# Q-Learning

## 4. 시간차 예측 (Temporal Difference Prediction)

(3) 시간차 예측이란 매 타임스텝마다 가치함수를 업데이트 하는 방법

$$V(s) \leftarrow V(s) + \alpha(G(s) - V(s))$$

(4) 위의 몬테카를로 예측 기반 가치함수 예측식에서 반환값  $G(s)$ 은 한 에피소드가 끝나야 알 수 있음

(5) 시간차 예측에서는 다음 스텝의 보상과 가치함수를 샘플링하여 현재 상태의 가치함수를 업데이트 한다.

$$R + \gamma V(s_{t+1}) - V(s_t)$$

# Q-Learning

## 4. 시간차 예측 (Temporal Difference Prediction)

$$V(s) \leftarrow V(s) + \alpha(G(s) - V(s))$$



$$\begin{aligned} v(s) &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= E[R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \dots) | S_t = s] \\ &= E[R_{t+1} + \gamma(G_{t+1}) | S_t = s] \\ &= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \end{aligned}$$

반환값이긴 하지만 사실 에이전트가  
실제로 받은 보상이 아직은 아님.  
따라서 가치함수 형태로 나타낼 수 있음



$$V(S_t) \leftarrow V(S_t) + \alpha(R + \gamma V(S_{t+1}) - V(s_t))$$

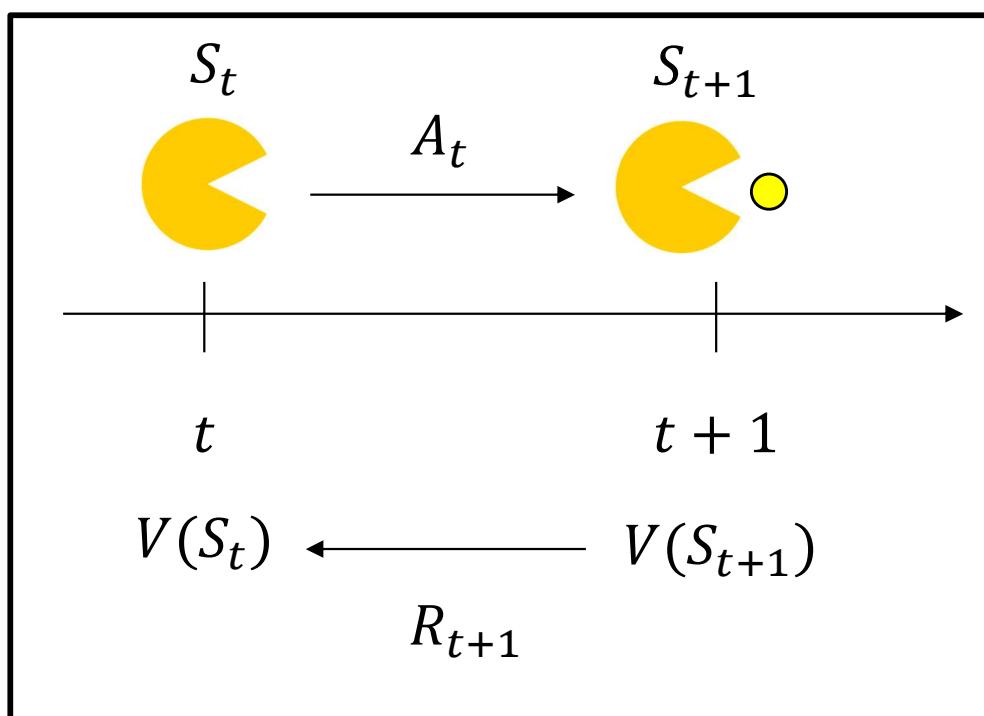
여기서  $R + \gamma V(S_{t+1})$  를 시간차 에러(Temporal-difference error)  
라고 함

# Q-Learning

## 4. 시간차 예측 (Temporal Difference Prediction)

- 따라서 시간차 예측은 어떤 상태에서 행동을 하면 보상을 받고 다음 상태를 알게되고 다음 상태의 가치함수와 알게된 보상을 더해 그 값을 업데이트의 목표로 삼는다는 것. 이 과정을 반복

$$V(S_t) \leftarrow V(S_t) + \alpha(R + \gamma V(S_{t+1}) - V(s))$$



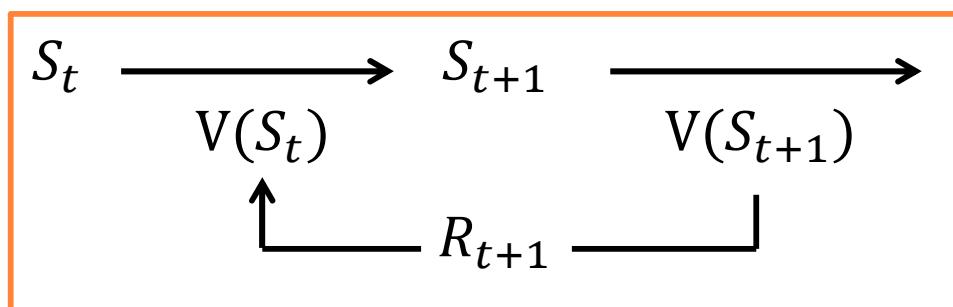
# Q-Learning

## 4. 시간차 예측 (Temporal Difference Prediction)

(6) 시간차 예측은 매 타임스텝마다 현재 상태에서 하나의 행동을 하고 환경으로부터 보상을 받고 다음 상태를 알게 됨

(7) 다음 상태의 예측값을 통해 현재의 가치함수를 업데이트 하는 방식을 강화학습에서는 **부트스트랩(Bootstrap)**이라고 함. 즉, 목표가 정확하지 않은 상태에서 현재의 가치함수를 업데이트 함

(8) 단점은 없을까?



# Q-Learning

---

## 5. 살사 (SARSA)

- (1) 한줄요약: 살사 = 정책 이터레이션 + 가치 이터레이션
- (2) 정책 이터레이션 = 정책 평가(예측) + 정책 발전(제어)
- (3) 예측: 가치함수 학습
- (4) 제어: 예측을 기반으로 정책을 발전 시킴
- (5) 시간차 예측의 문제점은 가치함수를 현재상태에서만 업데이트함. 즉, 모든 상태에서의 정책을 발전시키기 어렵다.
- (6) 이 문제를 살사에서는 가치 이터레이션을 통해 해결함

# Q-Learning

---

## 5. 살사 (SARSA)

(7) 즉 살사 = 시간차 예측 + 탐욕정책( $\varepsilon$ -greedy)

(8) 살사에서 업데이트 하는 대상은 가치함수가 아닌 큐함수임.  
왜냐하면 현재상태의 정책을 발전시키려면  $R_{t+1} + \gamma v_n(s')$ 의  
최댓값을 알아야하는데 그러려면 정책에 대한 정보 (환경에 대한  
정보)를 알아야 하기 때문

$$v_{n+1}(s) = \operatorname{argmax}_{a \in A}(R_{t+1} + \gamma v_n(s'))$$

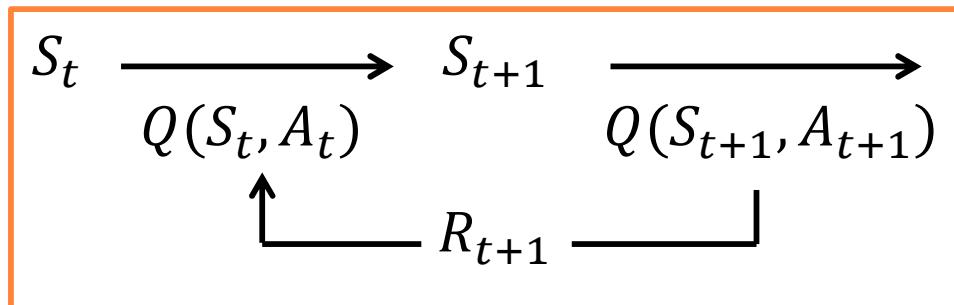
$$\pi'(s) = \operatorname{argmax}_{a \in A} q_\pi(s, a)$$

# Q-Learning

## 5. 살사 (SARSA)

(9) 큐함수를 업데이트 하려면 샘플이 필요함

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$



(10) 따라서 살사에서는  $[S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}]$ 를 샘플로 사용함

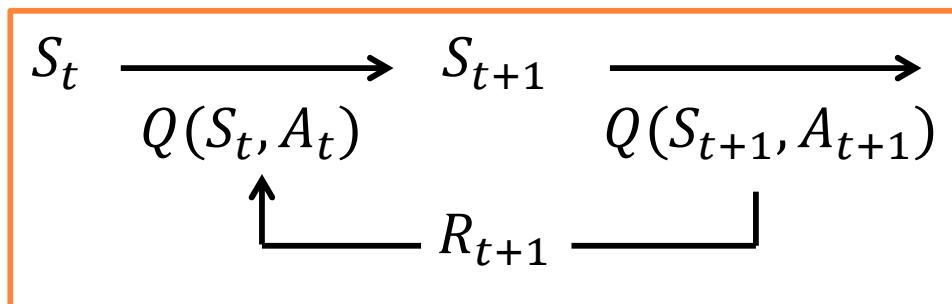
(11) 즉, 샘플의 상태  $S_t$ 에서 탐욕정책에 따라  $A_t$ 로 행동하고 그 다음스텝의 보상  $R_{t+1}$ 을 받음. 여기서 에이전트가 한번 더  $S_{t+1}$ 에서 행동  $A_{t+1}$ 을 하면 샘플이 생성되고 이 샘플로 큐함수를 학습시킴

# Q-Learning

## 5. 살사 (SARSA)

(12) 살사는 큐함수를 토대로 샘플을 탐욕 정책으로 모으고 그 샘플로 방문한 큐함수를 업데이트하는 과정을 반복함

(13) 기존의 탐욕정책으로 살사 알고리즘을 실행하면 문제가 없을까?



# Q-Learning

---

## 5. 살사 (SARSA)

- (14) 초기에 무작위로 행동하는게 맞지만 계속 그렇게 할 경우 잘못된 학습을 할 가능성이 매우 큼
- (15) 따라서 앞에서 사용한 탐욕정책을 개선한  $\varepsilon$ -탐욕정책을 사용함
- $$\pi(s) = \begin{cases} a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a), [1 - \varepsilon] \\ \quad a \neq a^*, [\varepsilon] \end{cases}$$
- (16) 하지만 이 방법은 최적의 큐함수를 찾아도 일정 확률( $\varepsilon$ )로 무작위 행동을 하며 탐험한다는 단점이 있음
- (17) 이는 초기에 설정한  $\varepsilon$ 값을 학습 시간이 흐름에 따라 점점 감소시키는 방법을 통해 해결할 수 있음

# Q-Learning

---

## 5. 살사 (SARSA)

### (18) 살사 요약

- $\varepsilon$ -greedy policy (exploration)

$$\pi(s) = \begin{cases} a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a) & , [1 - \varepsilon] \\ a \neq a^* & , [\varepsilon] \end{cases}$$

- Sampling

$$[S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}]$$

- Update Q-function

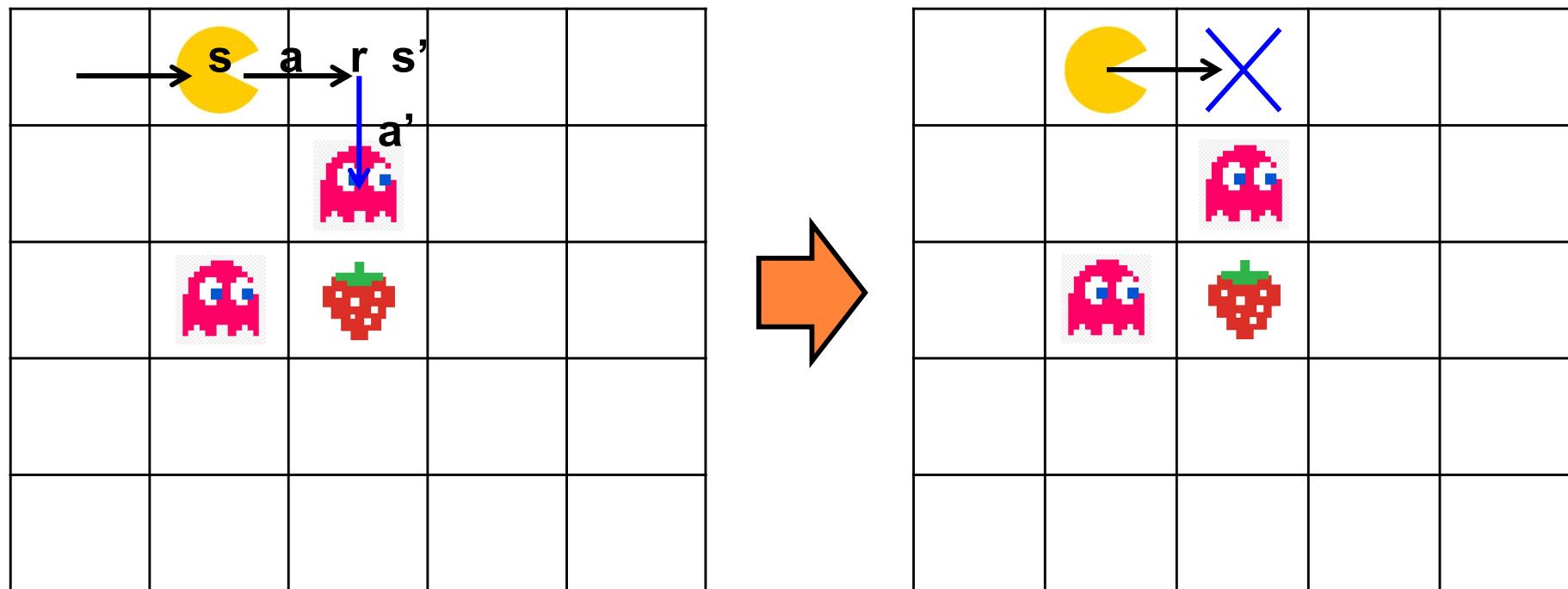
$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

# Q-Learning

## 5. 살사 (SARSA)

(19) 살사의 한계(a.k.a On-Policy)

(20) 온폴리시(On-Policy)란 행동 정책과 학습 정책이 같은걸 의미



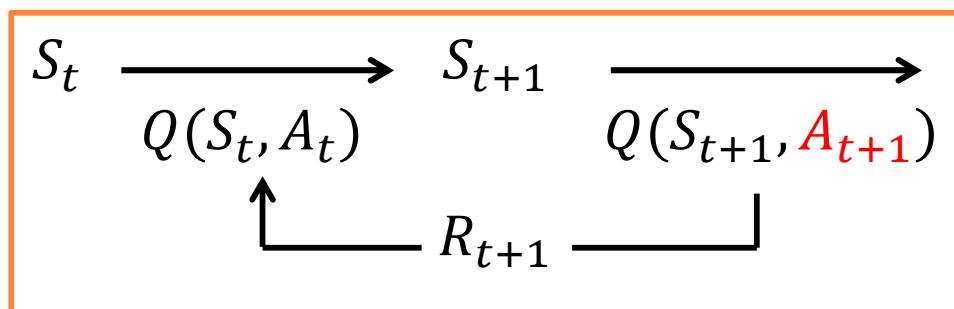
# Q-Learning

## 6. 큐러닝 (Q-Learning)

(1) 온폴리시 학습의 경우 탐험에서 문제점이 발생함

(2) 강화학습에서 탐험은 필수적인데 그렇다고 안할 수도 없고...

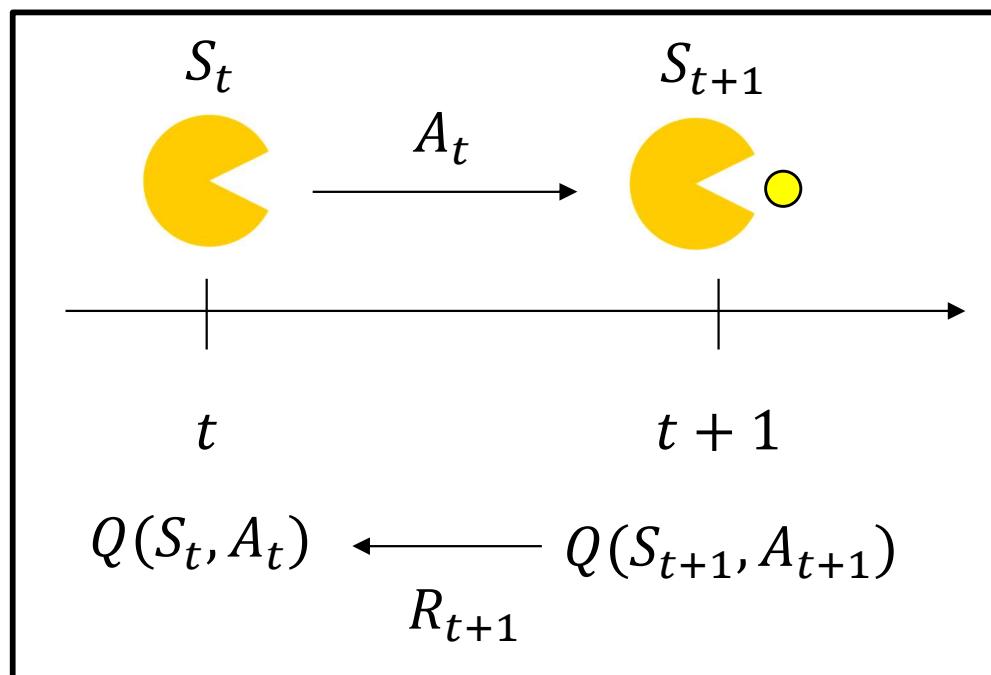
(3) 온폴리시에서 문제가 발생하는 이유가 행동 정책과 학습 정책이 같아서니까 그 둘이 다르면 되지 않을까???



# Q-Learning

## 6. 큐러닝 (Q-Learning)

- 큐러닝은 에이전트가 다음상태를 알게되면 그 상태에서 가장 큰 큐함수를 현재 큐함수의 업데이트에 사용함.



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( R + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

# Q-Learning

---

## 6. 큐러닝 (Q-Learning)

- (4) 이와 같은 방식을 **오프폴리시(Off-Policy)**라고 함
- (5) 큐러닝은 살사의 딜레마를 해결하기 위해 행동 선택은  $\epsilon$ -탐욕정책으로, 업데이트는 벨만 최적 방정식으로 진행한다.

# Q-Learning

---

## 6. 큐러닝 (Q-Learning)

### (6) 큐러닝 요약

- $\varepsilon$ -greedy policy (exploration)

$$\pi(s) = \begin{cases} a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a) & , [1 - \varepsilon] \\ a \neq a^* & , [\varepsilon] \end{cases}$$

- Sampling

$[S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}]$

- Update Q-function

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

# Q-Learning

---

## 6. 큐러닝 (Q-Learning)

### (6) 큐러닝 요약

- $\varepsilon$ -greedy policy (exploration)

$$\pi(s) = \begin{cases} a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a) & , [1 - \varepsilon] \\ a \neq a^* & , [\varepsilon] \end{cases}$$

- Sampling

$[S_t, A_t, R_{t+1}, S_{t+1}]$

- Update Q-function

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$$

# Q-Learning

---

## 6. 큐러닝 (Q-Learning)

### (6) 큐러닝 요약

- $\varepsilon$ -greedy policy (exploration)

$$\pi(s) = \begin{cases} a^* = \underset{a \in A}{\operatorname{argmax}} Q(s, a) & , [1 - \varepsilon] \\ a \neq a^* & , [\varepsilon] \end{cases}$$

- Sampling

$[S_t, A_t, R_{t+1}, S_{t+1}]$

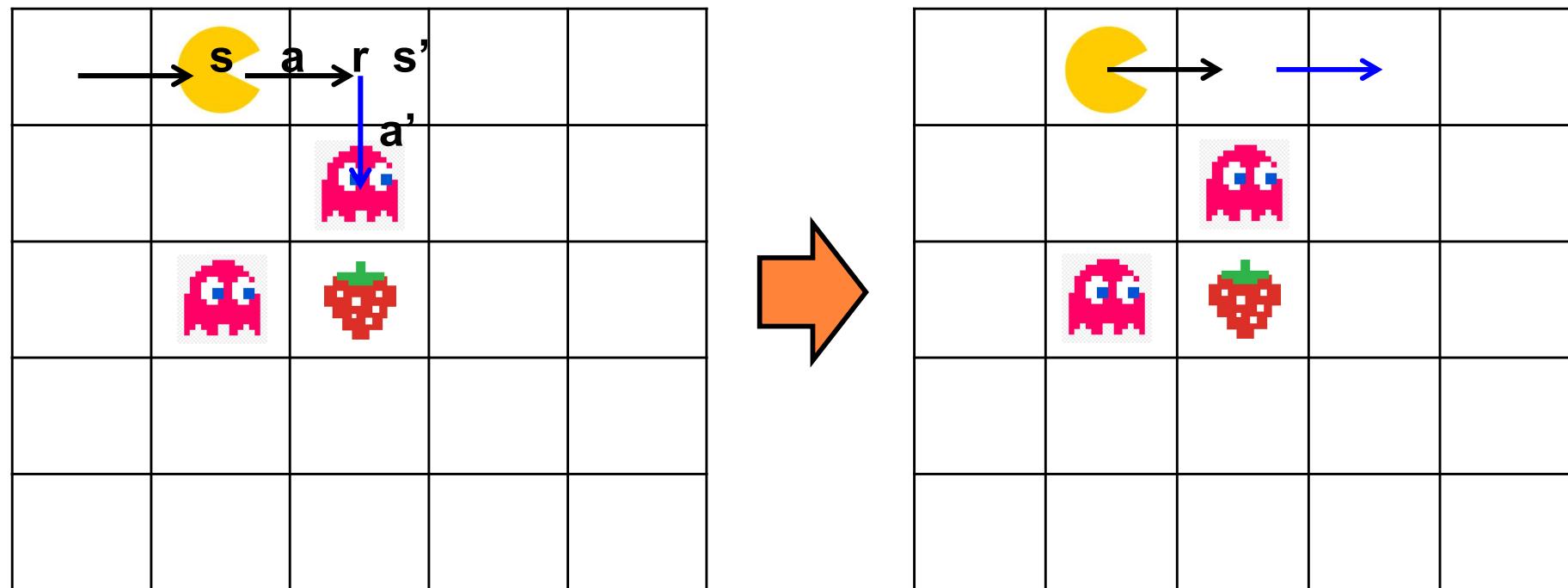
- Update Q-function

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( R + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

# Q-Learning

## 6. 큐러닝 (Q-Learning)

아까와 같이 구석에 같히는 경우가 발생하지 않음



# Q-Learning

---

## # 진단 평가

- 1) 예측과 제어에 대해서 설명하시오
- 2) 몬테카를로 예측이란? 문제점은?
- 3) 시간차 예측이란? 문제점은?
- 4) 살사란? 문제점은?
- 5) 큐러닝이란? 문제점은?

# Outline

---

- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Deep Reinforcement Learning

---

## 1. 몬테카를로, 살사, 큐러닝의 한계?

- 동적 프로그래밍의 한계

(1) 계산 복잡도 (i.e. 5x5 그리드 월드가 아니라  $n \times n$ 이라면?)

(2) 차원의 저주 (i.e. 그리드 월드처럼 2차원이 아니라  $n$ 차원이라면?)

(3) 환경에 대한 완벽한 정보를 알아야 한다 (우리는 실제로 세상을 탑뷰로 바라보며 모든 환경에 대한 정보를 인지하고 있는가?)

# Deep Reinforcement Learning

---

## 1. 몬테카를로, 살사, 큐러닝의 한계?

- 동적 프로그래밍의 한계

(1) 계산 복잡도 (i.e. 5x5 그리드 월드가 아니라  $n \times n$ 이라면?)

(2) 차원의 저주 (i.e. 그리드 월드처럼 2차원이 아니라  $n$ 차원이라면?)

~~(3)~~ 환경에 대한 완벽한 정보를 알아야 한다 (우리는 실제로 세상을 탑뷰로 바라보며 모든 환경에 대한 정보를 인지하고 있는가?)

- 위의 세 방법은 동적 프로그래밍에서 발생하는 문제 중 (3)은 해결 하였지만 (1), (2)는 해결하지 못했음
- 왜냐하면 테이블 형식을 사용하기 때문

# Deep Reinforcement Learning

## 1. 몬테카를로, 살사, 큐러닝의 한계?

- 예를 들어, 아래의 그리드 월드는 상태가 25가지, 할 수 있는 행동은 5가지이다(제자리에 있기 추가). 그러면 모든 상태의 갯수는  $25 \times 5 = 125$ 개임. 그런데 만약 장애물(귀신)이 움직인다면?

	$s_2$	$s_3$	$s_4$	$s_5$
$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$
$s_{11}$	$s_{12}$	$s_{13}$	$s_{14}$	$s_{15}$
$s_{16}$	$s_{17}$	$s_{18}$	$s_{19}$	$s_{20}$
$s_{21}$	$s_{22}$	$s_{23}$	$s_{24}$	

# Deep Reinforcement Learning

---

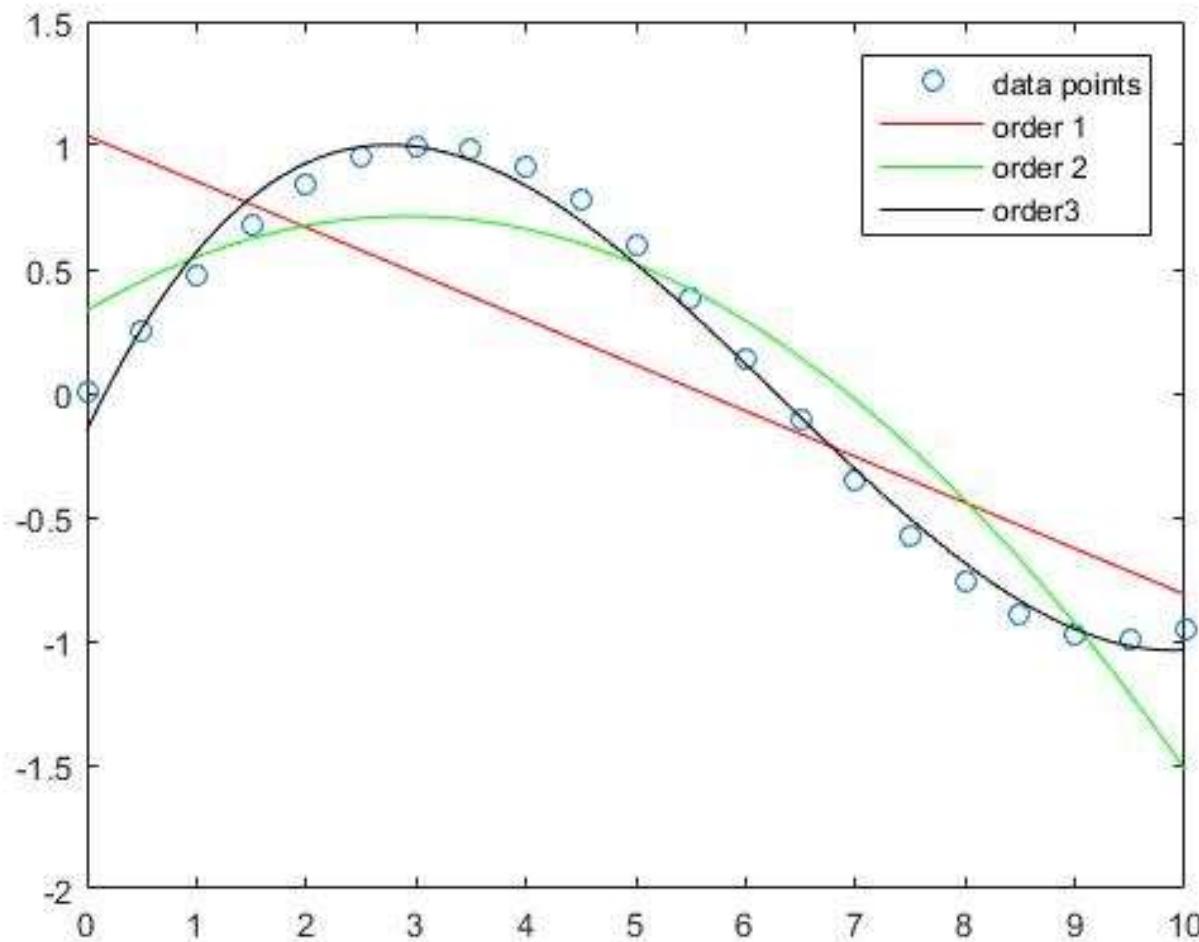
## 1. 몬테카를로, 살사, 큐러닝의 한계?

- 앞에서 배운 고전 강화학습 알고리즘은 상태가 적은 경우에만 적용이 가능함.
- 그렇다면 이 문제를 어떻게 해결할 수 있을까?

# Deep Reinforcement Learning

## 2. 큐함수의 매개변수화 & 근사화

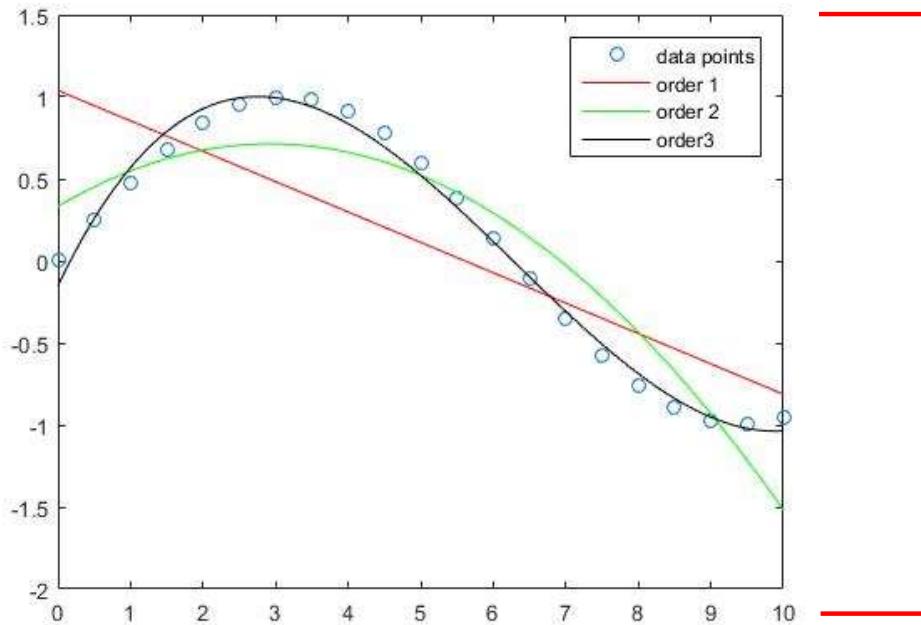
### (1) 근사화?



# Deep Reinforcement Learning

## 2. 큐함수의 매개변수화 & 근사화

### (2) 매개변수화?



$$ax^3 + bx^2 + cx + d$$

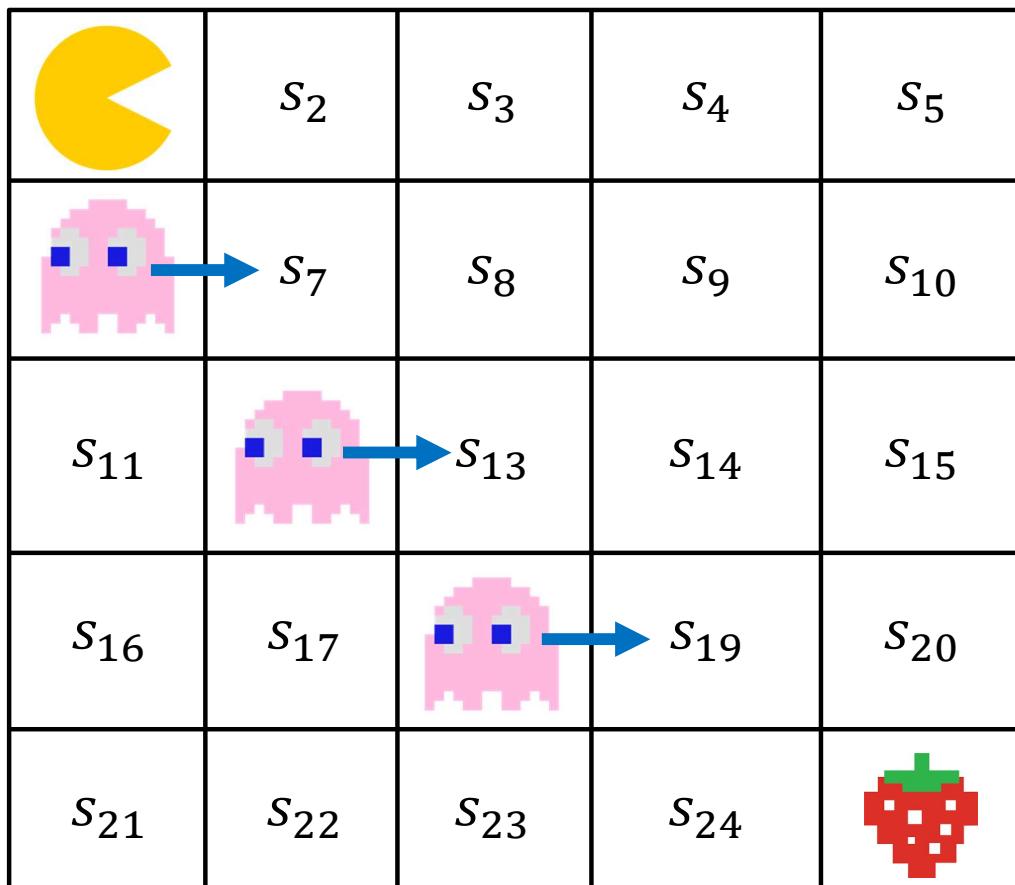
→  $(a, b, c, d)$ : 매개변수

매개 변수들만으로도 기존의 데이터를 대체할 수 있음  
따라서 기존의 테이블 형식을 사용하지 않고 **큐함수를 근사화** 한다.

# Deep Reinforcement Learning

## 3. Deep-SARSA

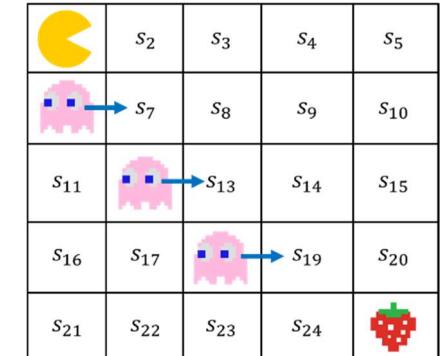
(1) 문제: 팩맨은 좌우로 움직이는 귀신 3마리를 피해서 딸기를 먹어야함



# Deep Reinforcement Learning

## 3. Deep-SARSA

(2) 복잡한 순차적 의사결정 문제 → MDP



사람도 어떠한 장애물을 피할때 정확하지는 않아도 물체가 내쪽으로 오고있는지 혹은 멀어지고 있는지, 그리고 속도가 어느정도 되는지 알아야한다.

MDP를 정의할때도 에이전트가 충분히 잘 학습할 수 있도록 상태 정보를 잘 정의해주어야한다.

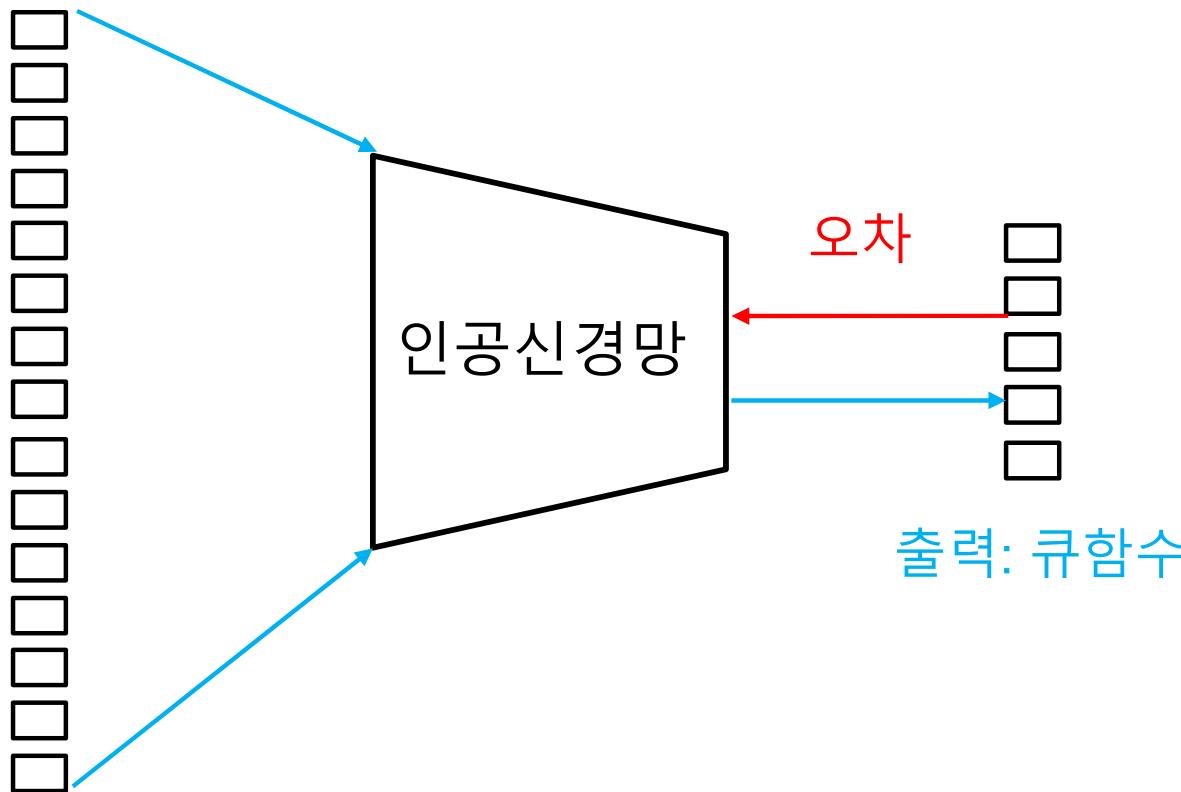
상태에 들어가야하는 정보: 도착지점의 상대위치, 도착지점의 레이블, 장애물의 상대위치, 장애물의 레이블, 장애물의 속도

장애물이 3개이므로  $12 + 3 = 15$ 개의 원소를 상태정보에 반영!

# Deep Reinforcement Learning

## 3. Deep-SARSA

(3) 딥 살사에서는 경사하강법(Gradient-Descent)을 이용하여 뉴럴네트워크를 업데이트함.



입력: 상태정보(15개의 원소)

# Deep Reinforcement Learning

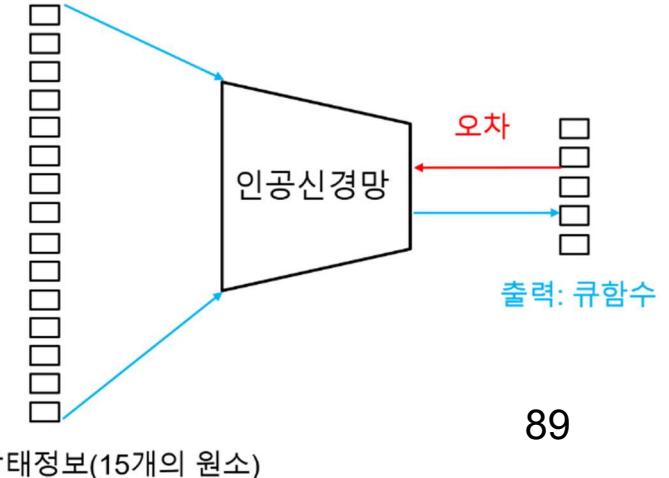
## 3. Deep-SARSA

(3) 딥 살사에서는 경사하강법(Gradient-Descent)을 이용하여 뉴럴네트워크를 업데이트함.

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

오학습을 위한 오차를 정의해주어야함

$$\text{MSE} = (R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))^2$$



# Deep Reinforcement Learning

---

## 4. Policy Gradient

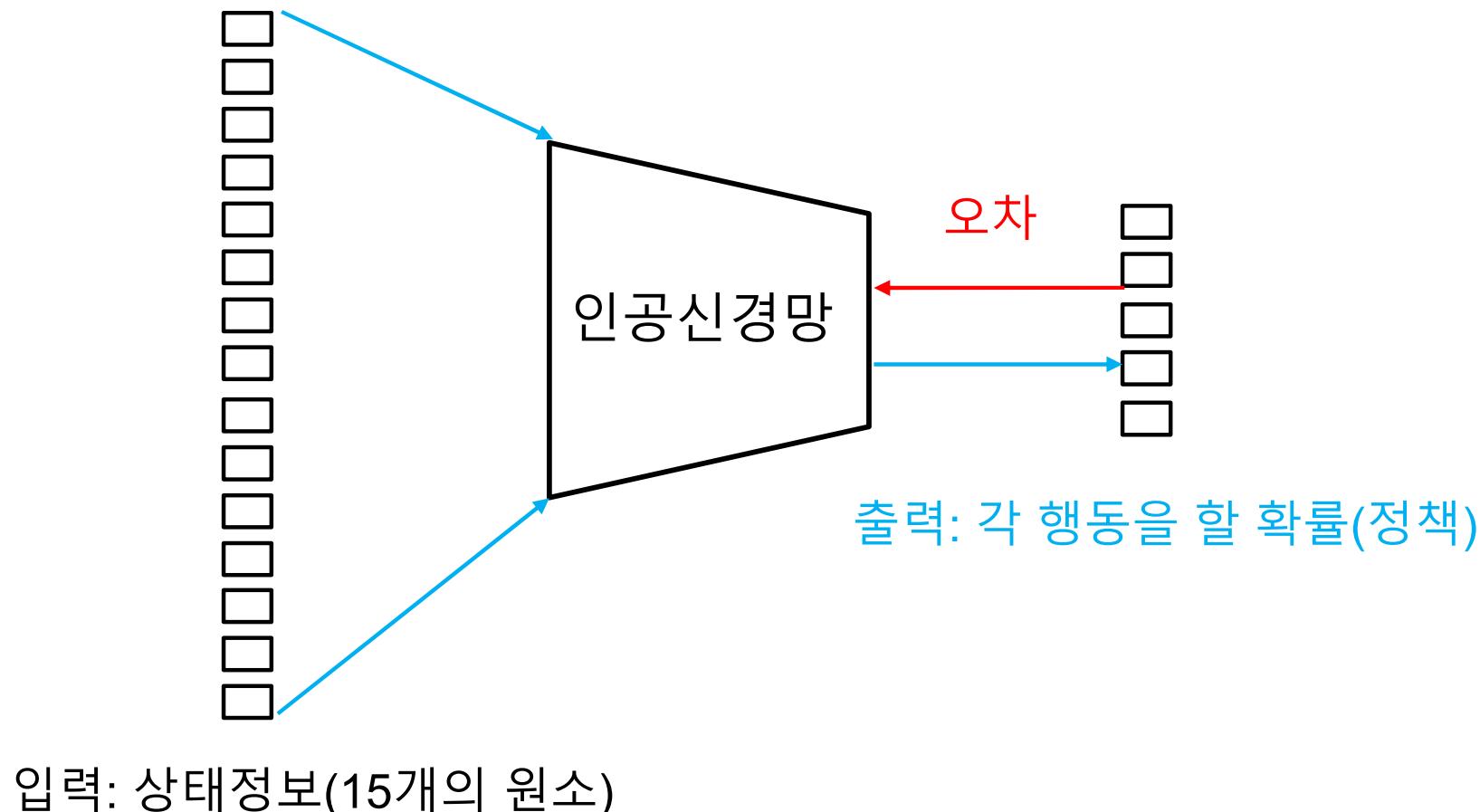
(1) 지금 까지 배웠던 강화학습 방법들은 모두 가치 기반 강화학습(Value-based RL)임. 즉, 가치함수, 큐함수(행동가치함수)를 업데이트 하며 학습함

(2) 다른 방법은 없을까?

# Deep Reinforcement Learning

## 4. Policy Gradient

(3) 정책기반 강화학습은 상태의 가치함수가 아닌 상태에 따라 바로 행동을 선택. 즉, 정책 기반 강화학습은 정책을 직접적으로 근사함.



# Deep Reinforcement Learning

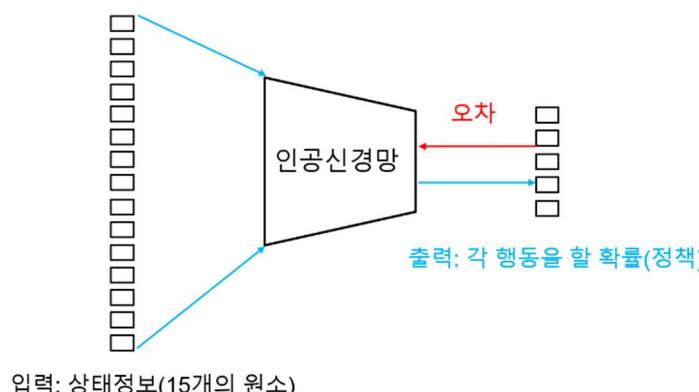
## 4. Policy Gradient

(4) 정책 기반 강화학습에서는 무엇을 목표로 학습을 진행할까?

(5) 강화학습의 목표는 누적 보상을 최대로 하는 정책을 찾는 것!

(6) 즉, 정책신경망을 사용하는 경우 정책 신경망의 계수(가중치, weight)에 따라 에이전트가 받을 누적 보상이 달라짐.

(7) 따라서 정책 기반 강화학습에서는 신경망의 가중치가 변수가 됨.



# Deep Reinforcement Learning

## 4. Policy Gradient

- 정책 =  $\pi_\theta(a|s)$
- 목표함수 =  $L(\theta)$  : 누적 보상과 관련된 인공신경망 계수
- 목표 = maximize  $L(\theta)$
- 목표 함수를 최대화 하기위해 경사상승법(Gradient Ascent)을 사용

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta L(\theta), \quad \alpha : \text{learning rate}$$

- 이렇게 목표함수의 근사된 정책을 경사상승법으로 업데이트 하는 것을 **폴리시 그레디언트(Policy Gradient)**라고 함.

$$\nabla_\theta L(\theta) = \nabla_\theta v_{\pi_\theta}(s_0)$$

$$= \sum_s d_{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(a|s) q_\pi(s, a)$$

# Deep Reinforcement Learning

## 4. Policy Gradient

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} L(\theta), \quad \alpha : \text{learning rate}$$

$$\nabla_{\theta} L(\theta) = \nabla_{\theta} v_{\pi_{\theta}}(s_0)$$

$$= \sum_s d_{\pi_{\theta}}(s) \sum_a \nabla_{\theta} \pi_{\theta}(a|s) q_{\pi}(s, a)$$

$d_{\pi_{\theta}}(s)$ :  $s$ 라는 상태에 에이전트가 있을 확률

(8) 즉, 가능한 모든 상태에 대해 각 상태에서 특정 행동을 했을 때 받을 큐함수의 기댓값

(9) 에이전트가 에피소드 동안 내릴 선택에 대한 좋고 나쁨의 지표

# Deep Reinforcement Learning

## 4. Policy Gradient

$$\begin{aligned}\nabla_{\theta} L(\theta) &= \sum_s d_{\pi_{\theta}}(s) \sum_a \nabla_{\theta} \pi_{\theta}(a|s) q_{\pi}(s, a) \\&= \sum_s d_{\pi_{\theta}}(s) \sum_a \pi_{\theta}(a|s) \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} q_{\pi}(s, a) \\&= \boxed{\sum_s d_{\pi_{\theta}}(s) \sum_a \pi_{\theta}(a|s)} \nabla_{\theta} \log \pi_{\theta}(a|s) q_{\pi}(s, a) \\&\quad \text{기댓값} \downarrow \\&= E_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(a|s) q_{\pi}(s, a)]\end{aligned}$$

$$(\because) \theta_{t+1} \approx \theta_t + \alpha [\nabla_{\theta} \log \pi_{\theta}(a|s) q_{\pi}(s, a)]$$

# Deep Reinforcement Learning

## 5. REINFORCE

(1) REINFORCE 알고리즘이란 큐함수 자리에 반환값을 사용한 것

$$\theta_{t+1} \approx \theta_t + \alpha [\nabla_\theta \log \pi_\theta(a|s) G_t]$$

(2) 오류함수(loss function)

$$Cross\ Entropy\ H(X) = - \sum_i y_i \log p_i, \quad p_i \rightarrow y_i$$

(3) 오류역전파(Back Propagation)를 통해 크로스 엔트로피를 줄이는 방향으로 가중치를 업데이트 했다면, 그 업데이트 값은 행동의 좋고 나쁨의 정보를 가지고 있는 반환값과 곱해짐.

# Detour: Cross Entropy

$$\text{Entropy } H(X) = - \sum_X P(X = x) \log_b P(X = x)$$

엔트로피 높다 → 더 불확실 하다

$$\text{Cross Entropy } H(X) = - \sum_i y_i \log p_i , p_i \rightarrow y_i$$

크로스 엔트로피 낮다 →  $p_i$ 가 정답인  $y_i$ 를 고를 가능성이 높아진다.

# Deep Reinforcement Learning

---

## # 진단 평가

- 1) 몬테카를로, 살사, 큐러닝의 한계는?
- 2) Deep SARSA란 무엇인가?
- 3) 정책 기반 강화학습에서는 무엇을 학습하는가?
- 4) REINFORCE란 무엇인가?
- 5) 크로스 엔트로피가 낮은건 무엇을 의미하는가?

# Outline

---

- Overview
- Applications
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)

# Deep Q-Networks (DQN)

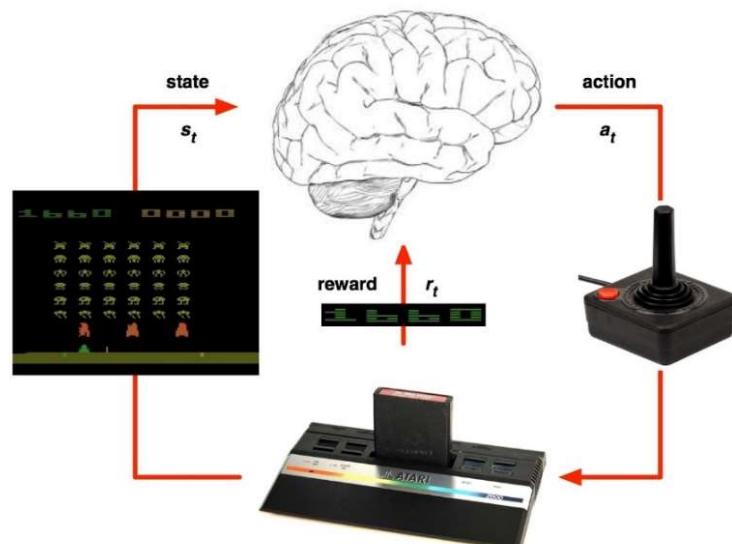
## 1. Deep Q-Networks

- Playing Atari with Deep Reinforcement Learning

([Minh et al. NIPS Deep Learning Workshop, 2013.](#))에 소개된 내용

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left( R + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right)$$

- 여기서 Q값을 인공신경망을 이용하여 딥러닝 방식으로 구한 것

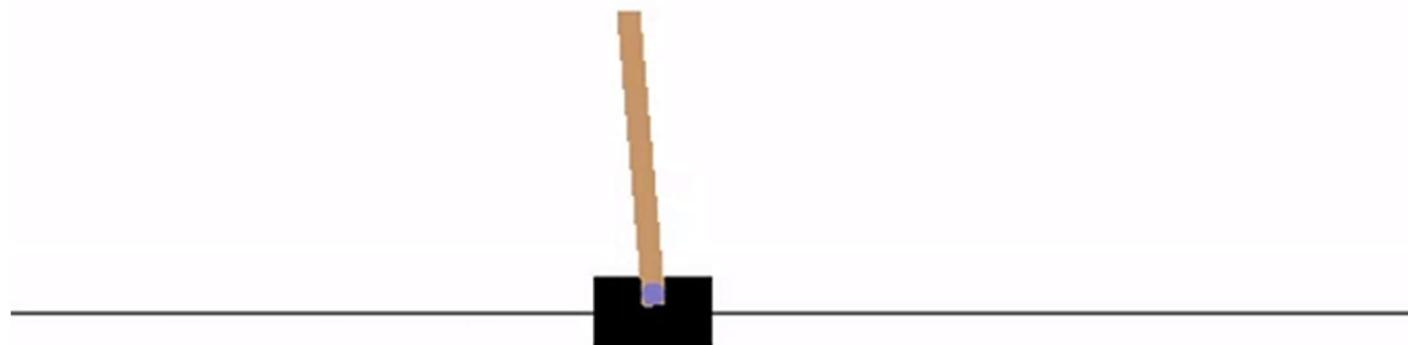


# **Deep Q-Networks (DQN)**

---

## 2. Cartpole

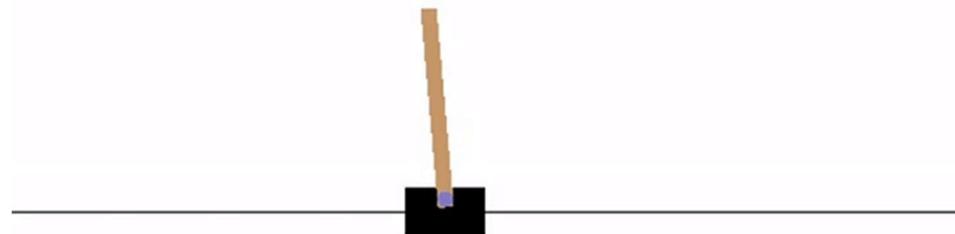
- OpenAI Gym에서 제공하는 실험환경



# Deep Q-Networks (DQN)

## 2. Cartpole

- Markov Decision Process (MDP)



1) 상태(state) : 카트의 위치, 속도, 폴의 각도, 각속도

$$= [x, \dot{x}, \theta, \dot{\theta}]$$

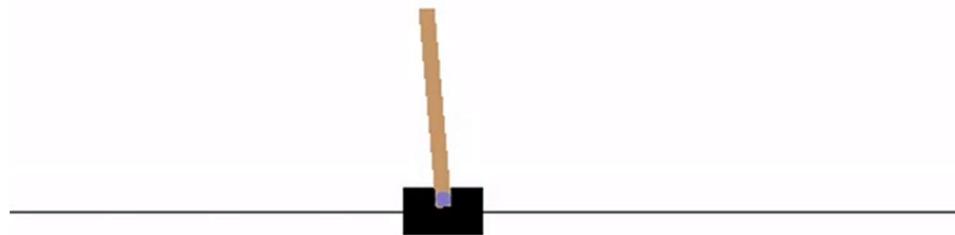
2) 행동(action) : 왼쪽(0), 오른쪽(1)

$$= [\leftarrow, \rightarrow]$$

# Deep Q-Networks (DQN)

## 2. Cartpole

- Markov Decision Process (MDP)



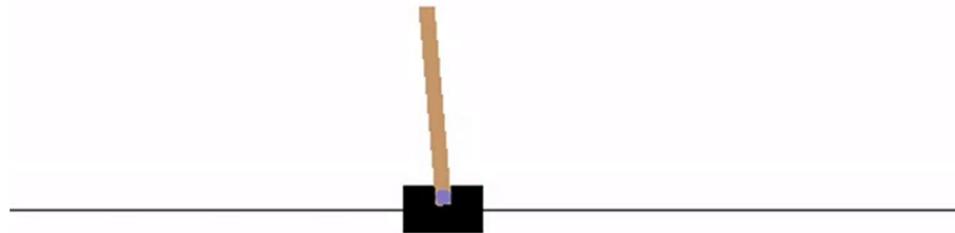
3) 보상(reward) : 카트폴이 쓰러지지 않고 버티는 시간

- 예를들어, 10초를 버티면 보상은 +10
- 여기선 단위가 초가 아니라 타임스텝
- 최대 500타임스텝까지 버틸 수 있음. 보상은 +500
- 중간에 카트폴이 쓰러지면 -100

# Deep Q-Networks (DQN)

## 2. Cartpole

- Markov Decision Process (MDP)

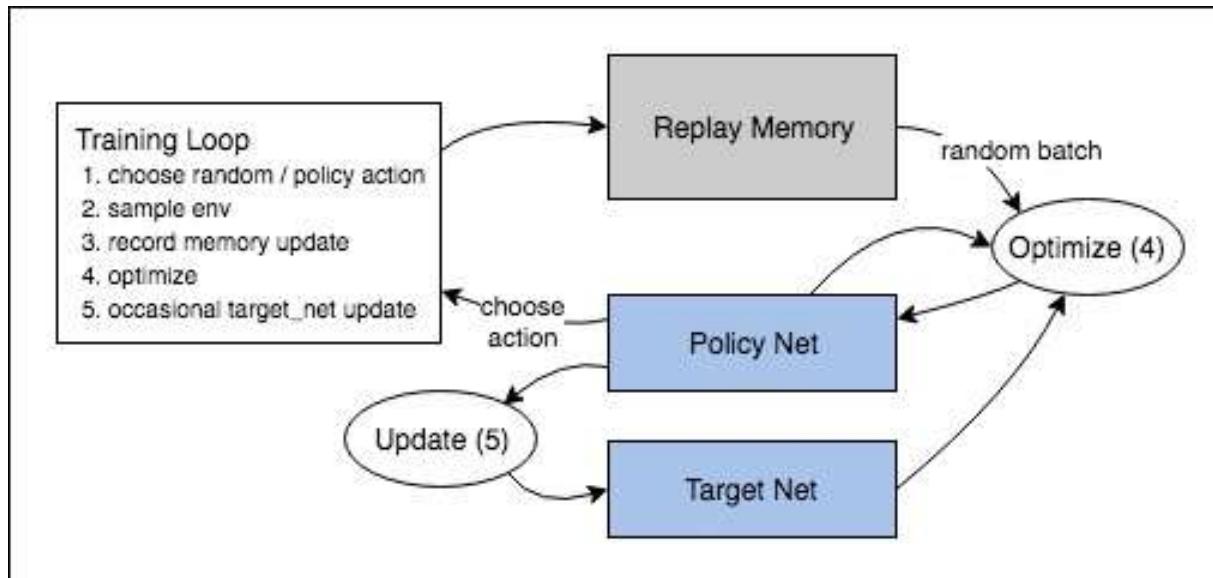


4) 감가율(discount factor) : Q함수에 대한 discount

- 0.99

# Deep Q-Networks (DQN)

## 3. 코드 설명



<http://bitly.kr/e2VNQp0FHnM>

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Environments

```
1 # CartPole-v1 환경, v1은 최대 타임스텝 500, v0는 최대 타임스텝 200
2 env = gym.make('CartPole-v1')
3 state_size = env.observation_space.shape[0] # 4
4 action_size = env.action_space.n # 2
5 print("state_size:", state_size)
6 print("action_size:", action_size)
```

- CartPole-v0 과 v1의 차이는 최대 타임스텝의 수 (각각 200, 500)
- state\_size = 4 (카트의 위치, 속도, 폴의 각도, 각속도)
- action\_size = 2 (왼쪽으로 움직이기, 오른쪽으로 움직이기)

# Deep Q-Networks (DQN)

## 3. 코드 설명

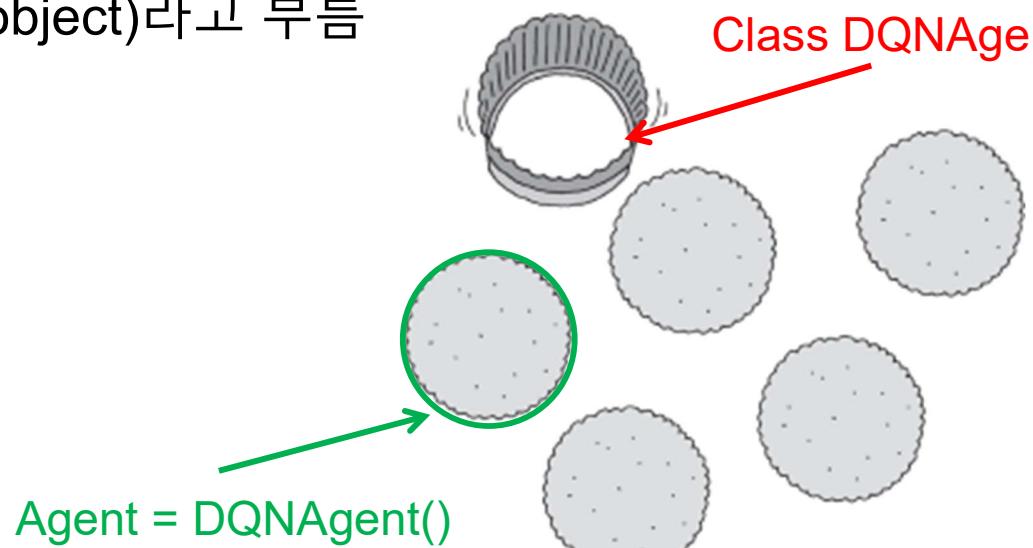
- Training

```
1 # DQN 에이전트 생성  
2 agent = DQNAgent(state_size, action_size)
```

1 class DQNAgent:

복사본을 만들어내고  
agent라는 이름을 붙였다.

- 클래스(class)란? 똑같은 무언가를 계속해서 만들어 낼 수 있는 설계도면 (예를들면 제과점에서 과자를 찍는 틀)
- DQN 속성을 가지는 agent를 찍어내고 agent라는 이름을 붙임
- 이 때 agent를 객체(object)라고 부름
- DQN Agent ???



# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

**def \_\_init\_\_():** 클래스를 사용할 때  
자동으로 실행됨

```
1 # DQN 에이전트 생성
2 agent = DQNAgent(state_size, action_size)
```

```
1 class DQNAgent:
2     def __init__(self, state_size, action_size):
3         # render : True이면 학습 진행 영상을 볼 수 있음, 싫으면 False
4         self.render = False
5         self.load_model = False
6
7         # 상태와 행동의 크기 정의
8         self.state_size = state_size # 4
9         self.action_size = action_size # 2
10
11        # DQN hyperparameter
12        # epsilon decay : 1.0에서 0.999씩 곱해지며 decay 됨
13        # epsilon min : decay되는 최솟값
14        # train start : 만큼의 샘플 갯수가 메모리에 모이면 그때부터 학습 시작
15        self.discount_factor = 0.99
16        self.learning_rate = 0.001
17        self.epsilon = 1.0 # epsilon이 1이면 무조건 무작위로 행동을 선택한다.
18        self.epsilon_decay = 0.999
19        self.epsilon_min = 0.01 # 지속적인 탐험을 위해 epsilon을 0으로 만들지 않고 하한선을 설정함.
20        self.batch_size = 64
21        self.train_start = 1000
22
23        # 리플레이 메모리, 최대크기 2000
24        # deque : 큐의 양쪽에서 삽입 삭제가 가능
25        self.memory = deque(maxlen = 2000)
26
27        # 모델과 타겟 모델 생성
28        # DQN의 특징 중 하나는 타겟신경망(모델)을 사용한다는 것
29        # 가중치가 무작위로 초기화 되기 때문에 현재 두 모델이 같지 않음
30        self.model = self.build_model()
31        self.target_model = self.build_model()
32
33        # 타겟 모델 초기화
34        self.update_target_model()
35
36        if self.load_model:
37            self.model.load_weights("./save_model/cartpole_dqn.h5")
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
1 class DQNAgent:  
2     def __init__(self, state_size, action_size):  
3         # render : True이면 학습 진행 영상을 볼 수 있음, 싫으면 False  
4         self.render = False  
5         self.load_model = False  
6  
7         # 상태와 행동의 크기 정의  
8         self.state_size = state_size # 4  
9         self.action_size = action_size # 2  
10  
11        # DQN hyperparameter  
12        # epsilon decay : 1.0에서 0.999씩 곱해지며 decay됨  
13        # epsilon min : decay되는 최솟값  
14        # train start : 만큼의 샘플 갯수가 메모리에 모이면 그때부터 학습 시작  
15        self.discount_factor = 0.99  
16        self.learning_rate = 0.001  
17        self.epsilon = 1.0 # epsilon이 1이면 무조건 무작위로 행동을 선택한다.  
18        self.epsilon_decay = 0.999  
19        self.epsilon_min = 0.01 # 지속적인 탐험을 위해 epsilon을 0으로 만들지 않고 하한선을 설정함.  
20        self.batch_size = 32  
21        self.train_start = 1000  
22  
23        # 리플레이 메모리, 최대크기 2000  
24        # deque : 큐의 양쪽에서 삽입 삭제가 가능  
25        self.memory = deque(maxlen = 2000)  
26  
27        # 모델과 타겟 모델 생성  
28        # DQN의 특징 중 하나는 타겟신경망(모델)을 사용한다는 것  
29        # 가중치가 무작위로 초기화 되기 때문에 현재 두 모델이 같지 않음  
30        self.model = self.build_model()  
31        self.target_model = self.build_model()  
32  
33        # 타겟 모델 초기화  
34        self.update_target_model()  
35  
36        if self.load_model:  
37            self.model.load_weights("./save_model/cartpole_dqn.h5")
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
17     self.epsilon = 1.0 # epsilon이 1이면 무조건 무작위로 행동을 선택한다.  
18     self.epsilon_decay = 0.999  
19     self.epsilon_min = 0.01 # 지속적인 탐험을 위해 epsilon을 0으로 만들지 않고 하한선을 설정함.
```

### 1) Detour : Epsilon Greedy Algorithm

- 강화학습에서 정말 중요한건 최적값을 위한 탐험(exploration)
- 탐험이 잘 이루어지지 않는다면 처음에 하던 행동만 계속 강화함
- 예를들어 처음 폴을 세울때 [ $\leftarrow, \rightarrow, \leftarrow, \rightarrow, \leftarrow, \rightarrow$ ]로 가장 오래 버텼다면 그 다음 에피소드는 [ $\leftarrow, \rightarrow, \leftarrow, \rightarrow, \leftarrow, \rightarrow$ ]를 반복한 뒤 다음 행동을 함.
- 이런식으로 학습이 진행되면 안되기 때문에 학습 초반에는 epsilon 값을 1로 줘서 계속 무작위 행동을 하게 하고
- epsilon에 epsilon\_decay 값을 계속 곱해서 epsilon 값을 작게 한다.
- 그렇게 하면 점점 무작위 행동 빈도는 줄고 최적 행동은 늘어난다.
- epsilon 값은 epsilon\_min 이하로 떨어지지 않는다.

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
1 class DQNAgent:  
2     def __init__(self, state_size, action_size):  
3         # render : True이면 학습 진행 영상을 볼 수 있음, 싫으면 False  
4         self.render = False  
5         self.load_model = False  
6  
7         # 상태와 행동의 크기 정의  
8         self.state_size = state_size # 4  
9         self.action_size = action_size # 2  
10  
11        # DQN hyperparameter  
12        # epsilon decay : 1.0에서 0.999씩 곱해지며 decay됨  
13        # epsilon min : decay되는 최솟값  
14        # train start : 만큼의 샘플 갯수가 메모리에 모이면 그때부터 학습 시작  
15        self.discount_factor = 0.99  
16        self.learning_rate = 0.001  
17        self.epsilon = 1.0 # epsilon이 1이면 무조건 무작위로 행동을 선택한다.  
18        self.epsilon_decay = 0.999  
19        self.epsilon_min = 0.01 # 지속적인 탐험을 위해 epsilon을 0으로 만들지 않고 하한선을 설정함  
20        self.batch_size = 64  
21        self.train_start = 1000  
22  
23        # 리플레이 메모리, 최대크기 2000  
24        # deque : 큐의 양쪽에서 삽입 삭제가 가능  
25        self.memory = deque(maxlen = 2000)  
26  
27        # 모델과 타겟 모델 생성  
28        # DQN의 특징 중 하나는 타겟신경망(모델)을 사용한다는 것  
29        # 가중치가 무작위로 초기화 되기 때문에 현재 두 모델이 같지 않음  
30        self.model = self.build_model()  
31        self.target_model = self.build_model()  
32  
33        # 타겟 모델 초기화  
34        self.update_target_model()  
35  
36        if self.load_model:  
37            self.model.load_weights("./save_model/cartpole_dqn.h5")
```

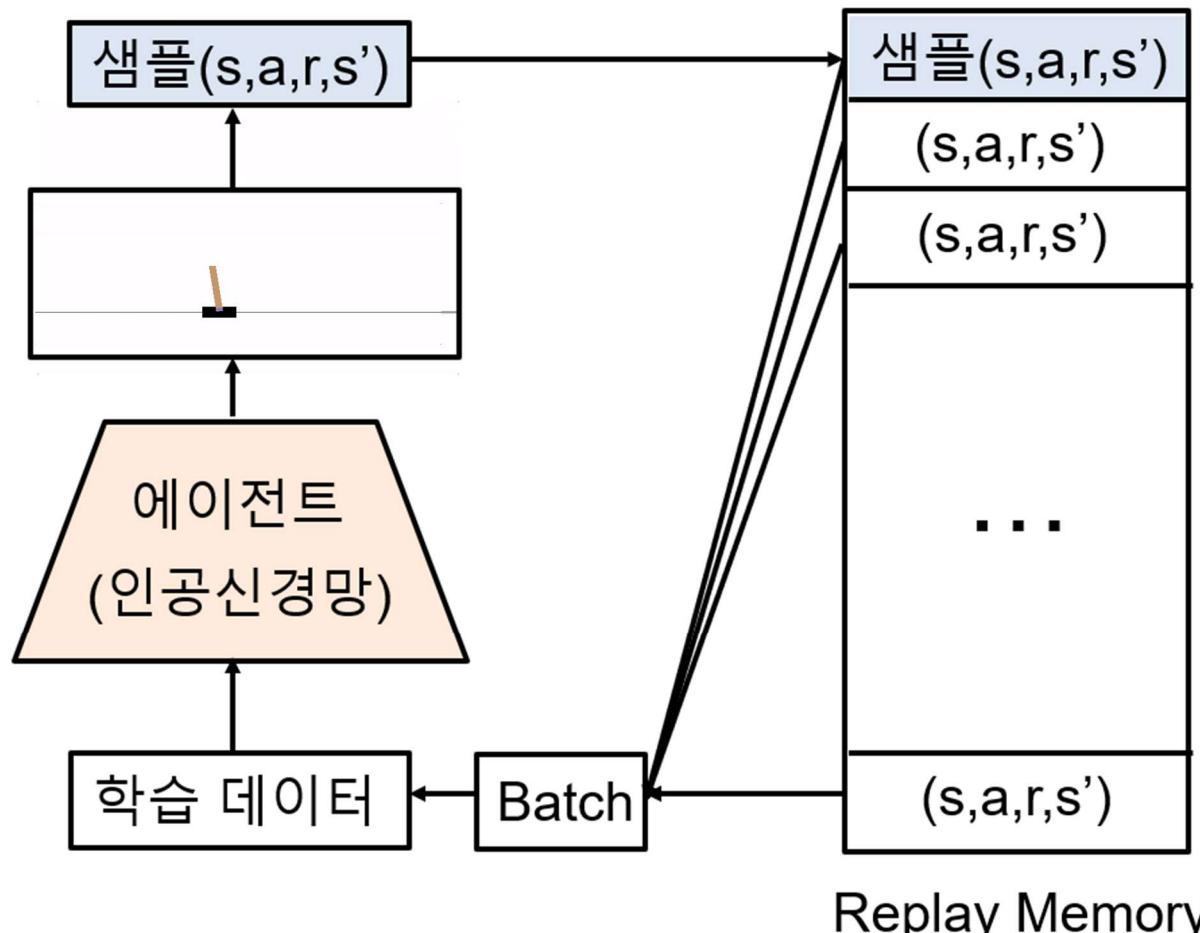
# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

### 2) Detour : Replay Memory

```
20         self.batch_size = 64
21         self.train_start = 1000
22
23         # 리플레이 메모리, 최대크기 2000
24         # deque : 큐의 양쪽에서 삽입 삭제가 가능
25         self.memory = deque(maxlen = 2000)
```



# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
1 class DQNAgent:  
2     def __init__(self, state_size, action_size):  
3         # render : True이면 학습 진행 영상을 볼 수 있음, 싫으면 False  
4         self.render = False  
5         self.load_model = False  
6  
7         # 상태와 행동의 크기 정의  
8         self.state_size = state_size # 4  
9         self.action_size = action_size # 2  
10  
11        # DQN hyperparameter  
12        # epsilon decay : 1.0에서 0.999씩 곱해지며 decay 됨  
13        # epsilon min : decay되는 최솟값  
14        # train start : 만큼의 샘플 갯수가 메모리에 모이면 그때부터 학습 시작  
15        self.discount_factor = 0.99  
16        self.learning_rate = 0.001  
17        self.epsilon = 1.0 # epsilon이 1이면 무조건 무작위로 행동을 선택한다.  
18        self.epsilon_decay = 0.999  
19        self.epsilon_min = 0.01 # 지속적인 탐험을 위해 epsilon을 0으로 만들지 않고 하한선을 설정함.  
20        self.batch_size = 64  
21        self.train_start = 1000  
22  
23        # 리플레이 메모리, 최대크기 2000  
24        # deque : 큐의 양쪽에서 삽입 삭제가 가능  
25        self.memory = deque(maxlen = 2000)  
26  
27        # 모델과 타겟 모델 생성  
28        # DQN의 특징 중 하나는 타겟신경망(모델)을 사용한다는 것  
29        # 가중치가 무작위로 초기화 되기 때문에 현재 두 모델이 같지 않음  
30        self.model = self.build_model()  
31        self.target_model = self.build_model()  
32  
33        # 타겟 모델 초기화  
34        self.update_target_model()  
35  
36        if self.load_model:  
37            self.model.load_weights("./save_model/cartpole_dqn.h5")
```

# Deep Q-Networks (DQN)

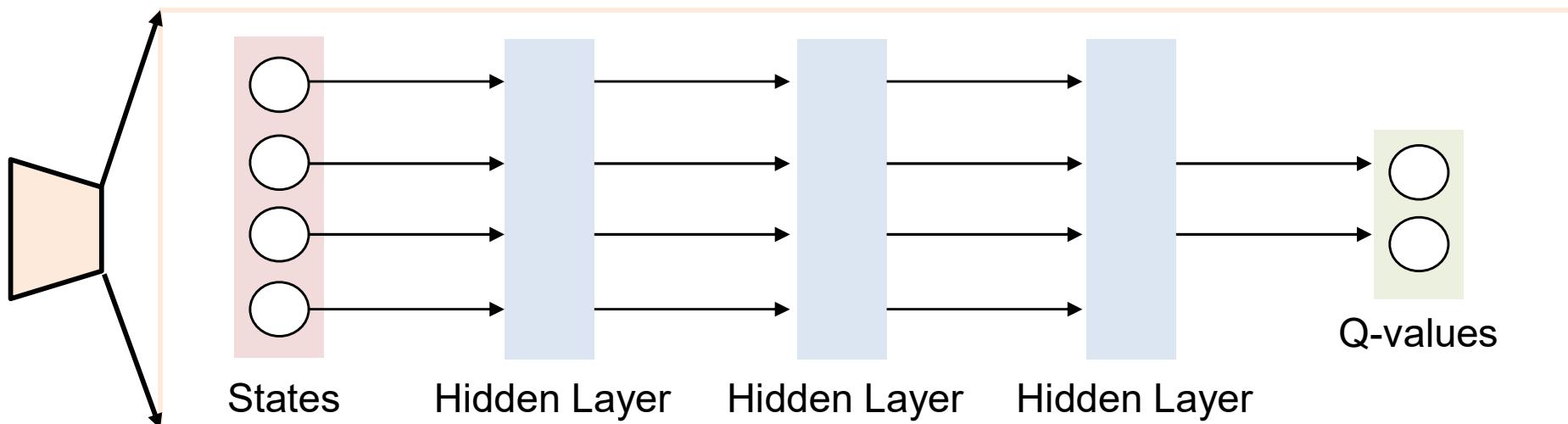
## 3. 코드 설명

- Training -> Agent

### 3) Detour : Target Network

```
27     # 모델과 타겟 모델 생성  
28     # DQN의 특징 중 하나는 타겟신경망(모델)을 사용한다는 것  
29     # 가중치가 무작위로 초기화 되기 때문에 현재 두 모델이 같지 않음  
30     self.model = self.build_model()  
31     self.target_model = self.build_model()
```

```
39     # 상태가 입력, 큐함수가 출력인 인공신경망 생성  
40     # he_uniform : 가중치 초기화 방법 / https://reniew.github.io/13/  
41     # 가중치 초기화 방법도 성능향상에 영향을 미친다.  
42     def build_model(self):  
43         model = Sequential()  
44         model.add(Dense(24, input_dim = self.state_size, activation = 'relu', kernel_initializer = 'he_uniform'))  
45         model.add(Dense(24, activation = 'relu', kernel_initializer = 'he_uniform'))  
46         model.add(Dense(self.action_size, activation = 'linear', kernel_initializer = 'he_uniform'))  
47         model.summary()  
48         model.compile(loss = 'mse', optimizer = Adam(lr = self.learning_rate))  
49         return model
```



# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

### 3) Detour : Target Network

$$- Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$

- Q함수의 업데이트는 다음상태 예측값을 통해 현재 상태를 예측  
(부트스트랩 방식)
- 부트스트랩의 문제점은 업데이트 목표가 계속 바뀜
  - 이를 방지하기 위해 정답을 만들어 내는 신경망을 한 에피소드동안 유지함
  - 즉, 타겟 신경망을 따로 만들어서 정답에 해당하는 값을 구함
  - 그 다음 구한 정답을 통해 다른 인공신경망을 계속 학습시키며 타겟 신경망은 한 에피소드 마다 학습된 인공신경망으로 업데이트 함

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
1 class DQNAgent:  
2     def __init__(self, state_size, action_size):  
3         # render : True이면 학습 진행 영상을 볼 수 있음, 싫으면 False  
4         self.render = False  
5         self.load_model = False  
6  
7         # 상태와 행동의 크기 정의  
8         self.state_size = state_size # 4  
9         self.action_size = action_size # 2  
10  
11        # DQN hyperparameter  
12        # epsilon decay : 1.0에서 0.999씩 곱해지며 decay됨  
13        # epsilon min : decay되는 최솟값  
14        # train start : 만큼의 샘플 갯수가 메모리에 모이면 그때부터 학습 시작  
15        self.discount_factor = 0.99  
16        self.learning_rate = 0.001  
17        self.epsilon = 1.0 # epsilon이 1이면 무조건 무작위로 행동을 선택한다.  
18        self.epsilon_decay = 0.999  
19        self.epsilon_min = 0.01 # 지속적인 탐험을 위해 epsilon을 0으로 만들지 않고 하한선을 설정함.  
20        self.batch_size = 64  
21        self.train_start = 1000  
22  
23        # 리플레이 메모리, 최대크기 2000  
24        # deque : 큐의 양쪽에서 삽입 삭제가 가능  
25        self.memory = deque(maxlen = 2000)  
26  
27        # 모델과 타겟 모델 생성  
28        # DQN의 특징 중 하나는 타겟신경망(모델)을 사용한다는 것  
29        # 가중치가 무작위로 초기화 되기 때문에 현재 두 모델이 같지 않음  
30        self.model = self.build_model()  
31        self.target_model = self.build_model()  
32  
33        # 타겟 모델 초기화  
34        self.update_target_model()  
35  
36        if self.load_model:  
37            self.model.load_weights("./save_model/cartpole_dqn.h5")
```

# **Deep Q-Networks (DQN)**

---

## 3. 코드 설명

- Training

- #. 중간 정리 & 자가진단

- a. 클래스란? 필요한 이유는?
    - b. Epsilon Greedy Algorithm이란? 필요한 이유는?
    - c. Replay Memory란? 필요한 이유는?
    - d. Target Networks란? 필요한 이유는?

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training

```
1 scores, episodes = [], []
2
3 N_EPISODES = 300
4 for e in range(N_EPISODES):
5     done = False
6     score = 0
7
8     # env 초기화
9     # state의 모양 : 4
10    state = env.reset()
11    # state의 모양 : 4 -> 1 x 4
12    state = np.reshape(state, [1, state_size])
```

- scores : reward를 한 에피소드 e마다 저장하는 변수
- reward 변수는 계속 바뀌기 때문에 편의를 위해 scores 변수를 사용
- episodes : 에피소드 번호를 저장
- scores와 episodes를 리스트(list)로 선언하는 이유는 뒤에서 그래프를 그리기 위해서임.
- ex) scores: [ 43, 42, 88, 125, 44, ... ] / episodes: [0, 1, 2, ...]

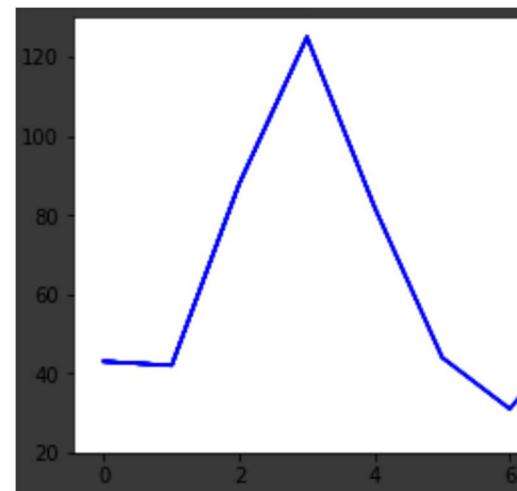
# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training

```
1 scores, episodes = [], []
2
3 N_EPISODES = 300
4 for e in range(N_EPISODES):
5     done = False
6     score = 0
7
8     # env 초기화
9     # state의 모양 : 4
10    state = env.reset()
11    # state의 모양 : 4 -> 1 x 4
12    state = np.reshape(state, [1, state_size])
```

```
episode: 0  score: 43.0
episode: 1  score: 42.0
episode: 2  score: 88.0
episode: 3  score: 125.0
episode: 4  score: 82.0
episode: 5  score: 44.0
```



# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training

```
1 scores, episodes = [], []
2
3 N_EPISODES = 300
4 for e in range(N_EPISODES):
5     done = False
6     score = 0
7
8     # env 초기화
9     # state의 모양 : 4
10    state = env.reset()
11    # state의 모양 : 4 -> 1 x 4
12    state = np.reshape(state, [1, state_size])
```

- state = env.reset() : 학습을 위한 state값 초기화. 모양은 4
- 학습의 편의를 위해 state의 모양을 4 -> 1 x 4로 변환함.  
(뒤에서 자세히 설명함)
- 모양이 4 : [1, 2, 3, 4]
- 모양이 1 x 4 : [ [1, 2, 3, 4] ]

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training

```
14      # done : false 였다가 한 에피소드가 끝나면 True로 바뀜
15      while not done:
16          # render = True 이면 학습영상 보여줌
17          if agent.render:
18              env.render()
19
20          # 현재 상태로 행동을 선택
21          action = agent.get_action(state)
```

- done : 에피소드 동안은 계속 False. 에피소드가 끝나면 True
- while not done : done은 False, not done은 True. while True는 무한반복
- 무한반복 하다가 done이 True로 바뀌면 빠져나오고 다음 에피소드 진행
- render : True면 학습영상 볼 수 있음. colab에선 지원하지 않는 기능ㅠㅠ  
(Pycharm!)
- 현재 state에서 get\_action 함수를 통해 행동값을 결정
- **agent.get\_action ???**

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
56     def get_action(self, state):
57         # 2 <= 3 : 첫번째 숫자가 두번째 보다 같거나 더 작은가? -> True or False
58         # np.random.rand() : 0~1 사이 실수 1개 / np.random.rand(5) : 0~1 사이 실수 5개
59         # random.randrange(5) : 0~4 임의의 정수 / random.randrange(-5,5) : -5 ~ 4 임의의 정수
60         if np.random.rand() <= self.epsilon:
61             return random.randrange(self.action_size)
62         else:
63             # q_value = [[-1.3104991 -1.6175464]]
64             # q_value[0] = [-1.3104991 -1.6175464]
65             # np.argmax(q_value[0]) = -1.3104991
66             q_value = self.model.predict(state)
67             return np.argmax(q_value[0])
```

- np.random.rand() : 0~1 사이 임의의 실수 1개
- self.epsilon : 아까 말했던 Epsilon Greedy Algorithm의 epsilon 값
- 즉, epsilon값이 더 크면 무작위 행동(왼쪽 or 오른쪽으로 움직이기)
- 그게 아니라면 계산한 두개의 Q값들 중 더 큰 값을 반환

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training

```
20     self.batch_size = 64
21     self.train_start = 1000
22
23     # 리플레이 메모리, 최대크기 2000
24     # deque : 큐의 양쪽에서 삽입 삭제가 가능
25     self.memory = deque(maxlen = 2000)
```

```
33     # 리플레이 메모리에 샘플 <s,a,r,s'> 저장
34     agent.append_sample(state, action, reward, next_state, done)
35
36     # 매 타임스텝마다 학습문
37     # self.train_start = 1000
38     # 이렇게 하는 이유는 DQN에서는 배치로 학습하기 때문에 샘플이 어느정도 모일때 까지 기다려야 하기때문.
39     if len(agent.memory) >= agent.train_start:
40         agent.train_model()
41
42     score += reward
43     state = next_state
```

- 이렇게 얻은 샘플 <s, a, r, s', done>을 리플레이 메모리에 추가
- 앞에 말했던 것처럼 샘플 1000개가 모이면 학습 시작
- **train\_model() ???**

### 3.

```
74 # 리플레이 메모리에서 배치 사이즈 만큼 무작위로 추출해서 학습하는 함수
75 def train_model(self):
76     if self.epsilon > self.epsilon_min:
77         self.epsilon *= self.epsilon_decay
78
79     # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80     # mini_batch의 모양: 64 x 5
81     # np.shape(mini_batch)
82     mini_batch = random.sample(self.memory, self.batch_size)
83
84     # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85     # model.fit(states, target)에 들어가는 states는 배치여야함
86     # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87     # np.zeros( (2, 3) ) : 2x3 영행렬
88     states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89     next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90     actions, rewards, dones = [], [], []
91
92     # def append_sample(self, state, action, reward, next_state, done):
93     # mini_batch의 모양: 64 x 5
94     # actions의 모양 : np.shape(actions)
95     for i in range(self.batch_size):
96         states[i] = mini_batch[i][0]
97         actions.append(mini_batch[i][1])
98         rewards.append(mini_batch[i][2])
99         next_states[i] = mini_batch[i][3]
100        dones.append(mini_batch[i][4])
101
102    # target 은 현재 상태에 대한 모델의 큐함수
103    # target_val 은 다음 상태에 대한 타겟 모델의 큐함수
104    # self.model = self.build_model()
105    # self.target_model = self.build_model()
106    # target 의 size: 64 x 2
107    # target_val 의 size : 64 x 2
108    target = self.model.predict(states)
109    target_val = self.target_model.predict(next_states)
110
111    # 벨만 최적 방정식을 이용한 업데이트 타겟
112    # amax 함수는 array 의 최댓값을 반환하는 함수
113    for i in range(self.batch_size): # i: 0 ~ 63
114        # actions[i] : 0 or 1
115        # dones[i] : False or True
116        if dones[i]:
117            target[i][actions[i]] = rewards[i]
118        else:
119            target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
120
121    # nb_epoch 로 에포크(epoch) 횟수 설정
122    # verbose는 학습 중 출력되는 문구를 설정하는 것으로,
123    # 주피터노트북(Jupyter Notebook)을 사용할 때는
124    # verbose=2로 설정하여 진행 막대(progress bar)가 나오지 않도록 설정한다.
125    self.model.fit(states, target, batch_size = self.batch_size, epochs = 1, verbose = 0)
```

### 3.

```
74 # 리플레이 메모리에서 배치 사이즈 만큼 무작위로 추출해서 학습하는 함수
75 def train_model(self):
76     if self.epsilon > self.epsilon_min:
77         self.epsilon *= self.epsilon_decay
78
79     # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80     # mini_batch의 모양: 64 x 5
81     # np.shape(mini_batch)
82     mini_batch = random.sample(self.memory, self.batch_size)
83
84     # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85     # model.fit(states, target)에 들어가는 states는 배치여야함
86     # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87     # np.zeros( (2, 3) ) : 2x3 영행렬
88     states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89     next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90     actions, rewards, dones = [], [], []
91
92     # def append_sample(self, state, action, reward, next_state, done):
93     # mini_batch의 모양: 64 x 5
94     # actions의 모양 : np.shape(actions)
95     for i in range(self.batch_size):
96         states[i] = mini_batch[i][0]
97         actions.append(mini_batch[i][1])
98         rewards.append(mini_batch[i][2])
99         next_states[i] = mini_batch[i][3]
100        dones.append(mini_batch[i][4])
101
102    # target 은 현재 상태에 대한 모델의 큐함수
103    # target_val 은 다음 상태에 대한 타겟 모델의 큐함수
104    # self.model = self.build_model()
105    # self.target_model = self.build_model()
106    # target 의 size: 64 x 2
107    # target_val 의 size : 64 x 2
108    target = self.model.predict(states)
109    target_val = self.target_model.predict(next_states)
110
111    # 벨만 최적 방정식을 이용한 업데이트 타겟
112    # amax 함수는 array 의 최댓값을 반환하는 함수
113    for i in range(self.batch_size): # i: 0 ~ 63
114        # actions[i] : 0 or 1
115        # dones[i] : False or True
116        if dones[i]:
117            target[i][actions[i]] = rewards[i]
118        else:
119            target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
120
121    # nb_epoch 로 에포크(epoch) 횟수 설정
122    # verbose는 학습 중 출력되는 문구를 설정하는 것으로,
123    # 주피터노트북(Jupyter Notebook)을 사용할 때는
124    # verbose=2로 설정하여 진행 막대(progress bar)가 나오지 않도록 설정한다.
125    self.model.fit(states, target, batch_size = self.batch_size, epochs = 1, verbose = 0)
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

S	A	R	S'	d
....	....	....	....	....

Mini Batch

X	X'	$\theta$	$\theta'$
....	....	....	....

States

X	X'	$\theta$	$\theta'$
....	....	....	....

Next States

```
79      # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80      # mini_batch의 모양: 64 x 5
81      # np.shape(mini_batch)
82      mini_batch = random.sample(self.memory, self.batch_size)
83
84      # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85      # model.fit(states, target)에 들어가는 states는 배치여야함
86      # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87      # np.zeros( (2, 3) ) : 2x3 영행렬
88      states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89      next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90      actions, rewards, dones = [], [], []
```

### 3.

```
74 # 리플레이 메모리에서 배치 사이즈 만큼 무작위로 추출해서 학습하는 함수
75 def train_model(self):
76     if self.epsilon > self.epsilon_min:
77         self.epsilon *= self.epsilon_decay
78
79     # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80     # mini_batch의 모양: 64 x 5
81     # np.shape(mini_batch)
82     mini_batch = random.sample(self.memory, self.batch_size)
83
84     # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85     # model.fit(states, target)에 들어가는 states는 배치여야함
86     # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87     # np.zeros( (2, 3) ) : 2x3 영행렬
88     states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89     next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90     actions, rewards, dones = [], [], []
91
92     # def append_sample(self, state, action, reward, next_state, done):
93     # mini_batch의 모양: 64 x 5
94     # actions의 모양 : np.shape(actions)
95     for i in range(self.batch_size):
96         states[i] = mini_batch[i][0]
97         actions.append(mini_batch[i][1])
98         rewards.append(mini_batch[i][2])
99         next_states[i] = mini_batch[i][3]
100        dones.append(mini_batch[i][4])
101
102    # target 은 현재 상태에 대한 모델의 큐함수
103    # target_val 은 다음 상태에 대한 타겟 모델의 큐함수
104    # self.model = self.build_model()
105    # self.target_model = self.build_model()
106    # target 의 size: 64 x 2
107    # target_val 의 size : 64 x 2
108    target = self.model.predict(states)
109    target_val = self.target_model.predict(next_states)
110
111    # 벨만 최적 방정식을 이용한 업데이트 타겟
112    # amax 함수는 array 의 최댓값을 반환하는 함수
113    for i in range(self.batch_size): # i: 0 ~ 63
114        # actions[i] : 0 or 1
115        # dones[i] : False or True
116        if dones[i]:
117            target[i][actions[i]] = rewards[i]
118        else:
119            target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
120
121    # nb_epoch 로 에포크(epoch) 횟수 설정
122    # verbose는 학습 중 출력되는 문구를 설정하는 것으로,
123    # 주피터노트북(Jupyter Notebook)을 사용할 때는
124    # verbose=2로 설정하여 진행 막대(progress bar)가 나오지 않도록 설정한다.
125    self.model.fit(states, target, batch_size = self.batch_size, epochs = 1, verbose = 0)
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

i++



	:	$S$
	:	$A$
	:	$R$
	:	$S$
	:	$\Omega$

Mini Batch

```
92     # def append_sample(self, state, action, reward, next_state, done):  
93     # mini_batch의 모양: 64 x 5  
94     # actions의 모양 : np.shape(actions)  
95     for i in range(self.batch_size):  
96         states[i] = mini_batch[i][0]  
97         actions.append(mini_batch[i][1])  
98         rewards.append(mini_batch[i][2])  
99         next_states[i] = mini_batch[i][3]  
100        dones.append(mini_batch[i][4])
```

Mini\_batch[i][0]

Mini\_batch[i][1]

Mini\_batch[i][2]

Mini\_batch[i][3]

Mini\_batch[i][4]

$x$	$x'$	$\theta$	$\theta'$
...	...	...	...

States

### 3.

```
74 # 리플레이 메모리에서 배치 사이즈 만큼 무작위로 추출해서 학습하는 함수
75 def train_model(self):
76     if self.epsilon > self.epsilon_min:
77         self.epsilon *= self.epsilon_decay
78
79     # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80     # mini_batch의 모양: 64 x 5
81     # np.shape(mini_batch)
82     mini_batch = random.sample(self.memory, self.batch_size)
83
84     # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85     # model.fit(states, target)에 들어가는 states는 배치여야함
86     # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87     # np.zeros( (2, 3) ) : 2x3 영행렬
88     states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89     next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90     actions, rewards, dones = [], [], []
91
92     # def append_sample(self, state, action, reward, next_state, done):
93     # mini_batch의 모양: 64 x 5
94     # actions의 모양 : np.shape(actions)
95     for i in range(self.batch_size):
96         states[i] = mini_batch[i][0]
97         actions.append(mini_batch[i][1])
98         rewards.append(mini_batch[i][2])
99         next_states[i] = mini_batch[i][3]
100        dones.append(mini_batch[i][4])
101
102    # target 은 현재 상태에 대한 모델의 큐함수
103    # target_val 은 다음 상태에 대한 타겟 모델의 큐함수
104    # self.model = self.build_model()
105    # self.target_model = self.build_model()
106    # target 의 size: 64 x 2
107    # target_val 의 size : 64 x 2
108    target = self.model.predict(states)
109    target_val = self.target_model.predict(next_states)
110
111    # 벨만 최적 방정식을 이용한 업데이트 타겟
112    # amax 함수는 array 의 최댓값을 반환하는 함수
113    for i in range(self.batch_size): # i: 0 ~ 63
114        # actions[i] : 0 or 1
115        # dones[i] : False or True
116        if dones[i]:
117            target[i][actions[i]] = rewards[i]
118        else:
119            target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
120
121    # nb_epoch 로 에포크(epoch) 횟수 설정
122    # verbose는 학습 중 출력되는 문구를 설정하는 것으로,
123    # 주피터노트북(Jupyter Notebook)을 사용할 때는
124    # verbose=2로 설정하여 진행 막대(progress bar)가 나오지 않도록 설정한다.
125    self.model.fit(states, target, batch_size = self.batch_size, epochs = 1, verbose = 0)
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

Detour : Target Network

- $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)]$$
- Q함수의 업데이트는 다음상태 예측값을 통해 현재 상태를 예측  
(부트스트랩 방식)
- 부트스트랩의 문제점은 업데이트 목표가 계속 바뀜
  - 이를 방지하기 위해 정답을 만들어 내는 신경망을 한 에피소드동안 유지함
  - 즉, 타겟 신경망을 따로 만들어서 정답에 해당하는 값을 구함
  - 그 다음 구한 정답을 통해 다른 인공신경망을 계속 학습시키며 타겟 신경망은 한 에피소드 마다 학습된 인공신경망으로 업데이트 함

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

Detour : Target Network

```
102     # target 은 현재 상태에 대한 모델의 큐함수  
103     # target_val 은 다음 상태에 대한 타겟 모델의 큐함수  
104     # self.model = self.build_model()  
105     # self.target_model = self.build_model()  
106     # target 의 size: 64 x 2  
107     # target_val 의 size : 64 x 2  
108     target = self.model.predict(states)  
109     target_val = self.target_model.predict(next_states)
```

$q_1$	$q_2$
...	...

target

$q_1$	$q_2$
...	...

target\_val

### 3.

```
74 # 리플레이 메모리에서 배치 사이즈 만큼 무작위로 추출해서 학습하는 함수
75 def train_model(self):
76     if self.epsilon > self.epsilon_min:
77         self.epsilon *= self.epsilon_decay
78
79     # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80     # mini_batch의 모양: 64 x 5
81     # np.shape(mini_batch)
82     mini_batch = random.sample(self.memory, self.batch_size)
83
84     # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85     # model.fit(states, target)에 들어가는 states는 배치여야함
86     # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87     # np.zeros( (2, 3) ) : 2x3 영행렬
88     states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89     next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90     actions, rewards, dones = [], [], []
91
92     # def append_sample(self, state, action, reward, next_state, done):
93     # mini_batch의 모양: 64 x 5
94     # actions의 모양 : np.shape(actions)
95     for i in range(self.batch_size):
96         states[i] = mini_batch[i][0]
97         actions.append(mini_batch[i][1])
98         rewards.append(mini_batch[i][2])
99         next_states[i] = mini_batch[i][3]
100        dones.append(mini_batch[i][4])
101
102    # target 은 현재 상태에 대한 모델의 큐함수
103    # target_val 은 다음 상태에 대한 타겟 모델의 큐함수
104    # self.model = self.build_model()
105    # self.target_model = self.build_model()
106    # target 의 size: 64 x 2
107    # target_val 의 size : 64 x 2
108    target = self.model.predict(states)
109    target_val = self.target_model.predict(next_states)
110
111    # 벨만 최적 방정식을 이용한 업데이트 타겟
112    # amax 함수는 array 의 최댓값을 반환하는 함수
113    for i in range(self.batch_size): # i: 0 ~ 63
114        # actions[i] : 0 or 1
115        # dones[i] : False or True
116        if dones[i]:
117            target[i][actions[i]] = rewards[i]
118        else:
119            target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
120
121    # nb_epoch 로 에포크(epoch) 횟수 설정
122    # verbose는 학습 중 출력되는 문구를 설정하는 것으로,
123    # 주피터노트북(Jupyter Notebook)을 사용할 때는
124    # verbose=2로 설정하여 진행 막대(progress bar)가 나오지 않도록 설정한다.
125    self.model.fit(states, target, batch_size = self.batch_size, epochs = 1, verbose = 0)
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

```
111     # 벨만 최적 방정식을 이용한 업데이트 타겟
112     # amax 함수는 array 의 최댓값을 반환하는 함수
113     for i in range(self.batch_size): # i: 0 ~ 63
114         # actions[i] : 0 or 1
115         # dones[i] : False or True
116         if dones[i]:
117             target[i][actions[i]] = rewards[i]
118         else:
119             target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
```

1) If `dones[i] == False` / 즉, 한 에피소드가 아직 끝나지 않음

- `else` 문으로 들어감

$q_1$	$q_2$
....	....

target

$q_1$	$q_2$
....	....

target\_val

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training -> Agent

2) If `dones[i] == True` / 즉, 한 에피소드가 끝났음

- If문으로 들어감

```
111     # 벨만 최적 방정식을 이용한 업데이트 타겟
112     # amax 함수는 array 의 최댓값을 반환하는 함수
113     for i in range(self.batch_size): # i: 0 ~ 63
114         # actions[i] : 0 or 1
115         # dones[i] : False or True
116         if dones[i]:
117             target[i][actions[i]] = rewards[i]
118         else:
119             target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
```

$q_1$	$q_2$
...	...

target

$q_1$	$q_2$
...	...

target\_val

### 3. 3

```
74 # 리플레이 메모리에서 배치 사이즈 만큼 무작위로 추출해서 학습하는 함수
75 def train_model(self):
76     if self.epsilon > self.epsilon_min:
77         self.epsilon *= self.epsilon_decay
78
79     # 메모리에서 배치 크기만큼 무작위로 샘플 추출
80     # mini_batch의 모양: 64 x 5
81     # np.shape(mini_batch)
82     mini_batch = random.sample(self.memory, self.batch_size)
83
84     # 모델의 업데이트는 배치로 샘플들을 모아서 한 번에 진행하기 때문에
85     # model.fit(states, target)에 들어가는 states는 배치여야함
86     # 따라서 np.zeros를 사용해 states의 형태를 배치 형태로 지정함.
87     # np.zeros((2, 3)) : 2x3 영행렬
88     states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
89     next_states = np.zeros((self.batch_size, self.state_size)) # 64 x 4
90     actions, rewards, dones = [], [], []
91
92     # def append_sample(self, state, action, reward, next_state, done):
93     # mini_batch의 모양: 64 x 5
94     # actions의 모양 : np.shape(actions)
95     for i in range(self.batch_size):
96         states[i] = mini_batch[i][0]
97         actions.append(mini_batch[i][1])
98         rewards.append(mini_batch[i][2])
99         next_states[i] = mini_batch[i][3]
100        dones.append(mini_batch[i][4])
101
102    # target 은 현재 상태에 대한 모델의 큐함수
103    # target_val 은 다음 상태에 대한 타겟 모델의 큐함수
104    # self.model = self.build_model()
105    # self.target_model = self.build_model()
106    # target 의 size: 64 x 2
107    # target_val 의 size : 64 x 2
108    target = self.model.predict(states)
109    target_val = self.target_model.predict(next_states)
110
111    # 벨만 최적 방정식을 이용한 업데이트 타겟
112    # amax 함수는 array 의 최댓값을 반환하는 함수
113    for i in range(self.batch_size): # i: 0 ~ 63
114        # actions[i] : 0 or 1
115        # dones[i] : False or True
116        if dones[i]:
117            target[i][actions[i]] = rewards[i]
118        else:
119            target[i][actions[i]] = rewards[i] + self.discount_factor * (np.amax(target_val[i]))
120
121    # nb_epoch 로 에포크(epoch) 횟수 설정
122    # verbose는 학습 중 출력되는 문구를 설정하는 것으로,
123    # 주피터노트북(Jupyter Notebook)을 사용할 때는
124    # verbose=2로 설정하여 진행 막대(progress bar)가 나오지 않도록 설정한다.
125    self.model.fit(states, target, batch_size = self.batch_size, epochs = 1, verbose = 0)
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Training

```
20     self.batch_size = 64
21     self.train_start = 1000
22
23     # 리플레이 메모리, 최대크기 2000
24     # deque : 큐의 양쪽에서 삽입 삭제가 가능
25     self.memory = deque(maxlen = 2000)
```

```
33     # 리플레이 메모리에 샘플 <s,a,r,s'> 저장
34     agent.append_sample(state, action, reward, next_state, done)
35
36     # 매 타임스텝마다 학습문
37     # self.train_start = 1000
38     # 이렇게 하는 이유는 DQN에서는 배치로 학습하기 때문에 샘플이 어느정도 모일때 까지 기다려야 하기때문.
39     if len(agent.memory) >= agent.train_start:
40         agent.train_model()
41
42     score += reward
43     state = next_state
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

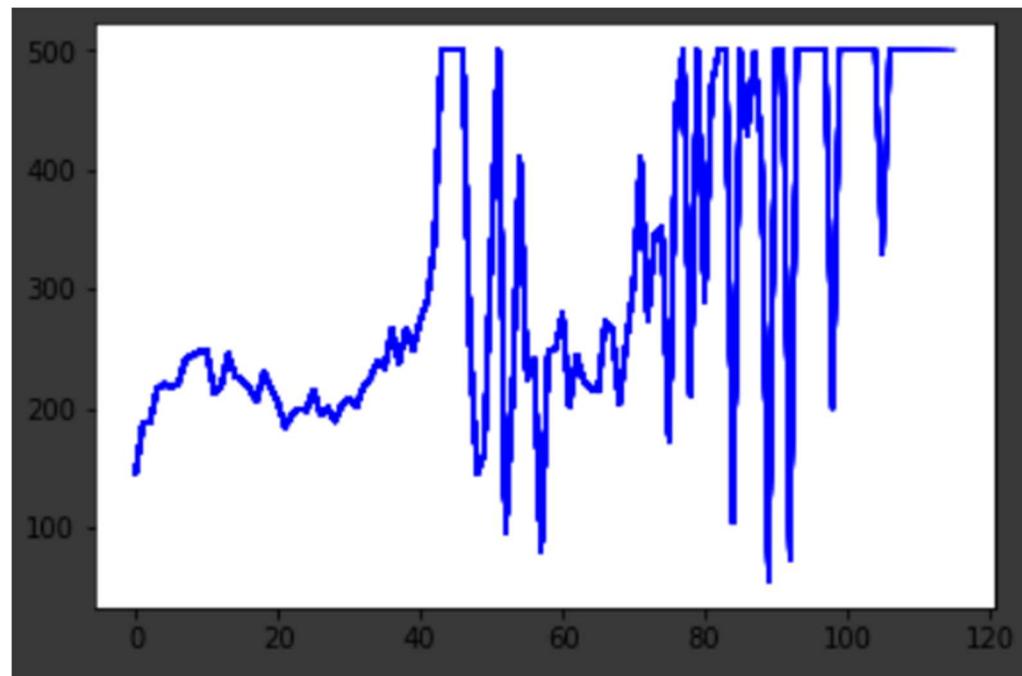
- Training

```
45     if done:  
46         # 각 에피소드마다 타겟 모델을 모델의 가중치로 업데이트  
47         agent.update_target_model()  
48  
49         score = score if score == 500 else score + 100  
50  
51         # 에피소드마다 학습결과 출력  
52         scores.append(score)  
53         episodes.append(e)  
54         pylab.plot(episodes, scores, 'b')  
55         if not os.path.exists("./save_graph"):  
56             os.makedirs("./save_graph")  
57         pylab.savefig("./save_graph/cartpole_dqn.png")  
58         print("episode:", e, " score:", score, " memory length:", len(agent.memory), " epsilon:", agent.epsilon)  
59  
60         # 이전 10개 에피소드의 점수 평균이 490보다 크면 학습 중단  
61         # np.mean([1, 2, 3]) = 2.0 / np.mean() : 평균  
62         # min([1, 2, 3]) = 1 / min : 가장 작은 값  
63  
64         # a = [ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]  
65         # print(a[-10:])  
66         # b = [1,2,3,4,5,6,7,8,9]  
67         # print(b[-9:])  
68         if np.mean(scores[-min(10, len(scores)):-1]) > 490:  
69             if not os.path.exists("./save_model"):  
70                 os.makedirs("./save_model")  
71             agent.model.save_weights("./save_model/cartpole_dqn.h5")  
72             sys.exit()
```

# Deep Q-Networks (DQN)

## 3. 코드 설명

- Result



# Q & A

---

## About me

### Jeiyoон Park

I'm a master's student in the Department of Computer Science and Engineering at [Korea University](#), advised by the professor [Heuseok Lim](#).

My research interests are dialog systems, reinforcement learning and meta-learning.

[Email](#) / [Github](#) / [Google Scholar](#) / [LinkedIn](#)



<https://jeiyoон.github.io/>

<https://www.youtube.com/channel/UC5dx094F-Se1DMI1vvVJyRw>