

Reinforcement Learning

Deep Q-Networks (DQN)

Jeiyoon Park

Department of Computer Science and Engineering



고려대학교
KOREA UNIVERSITY



Natural Language
Processing
& Artificial Intelligence



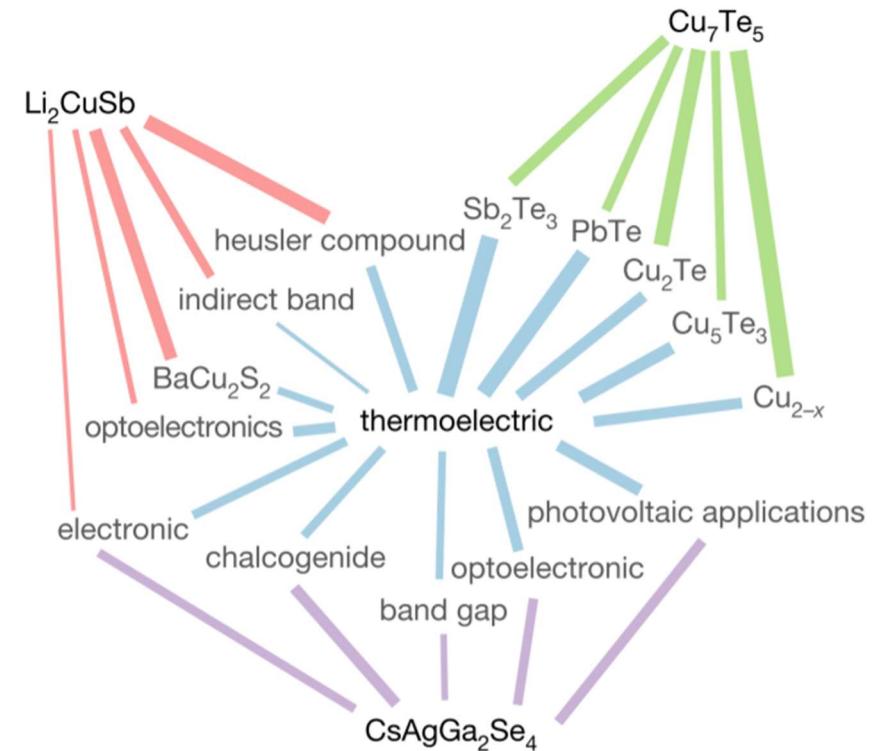
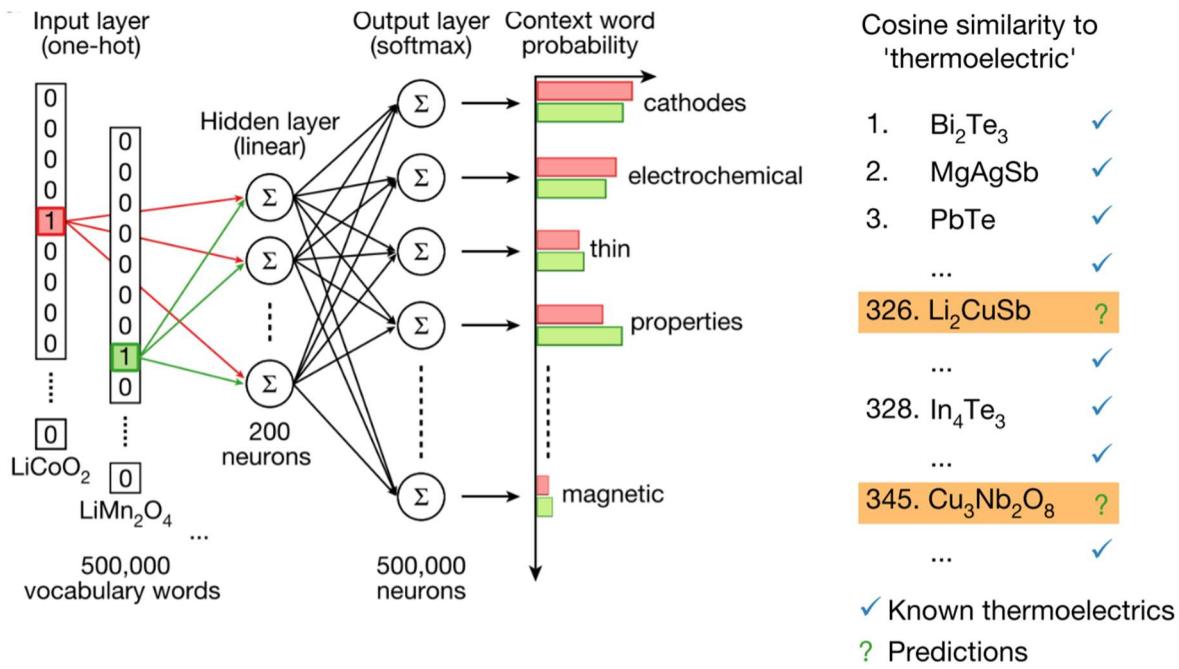
Outline

- Introduction
- Detour: RL basics
- DQN

Introduction (1/4)

1. NLP in chemistry

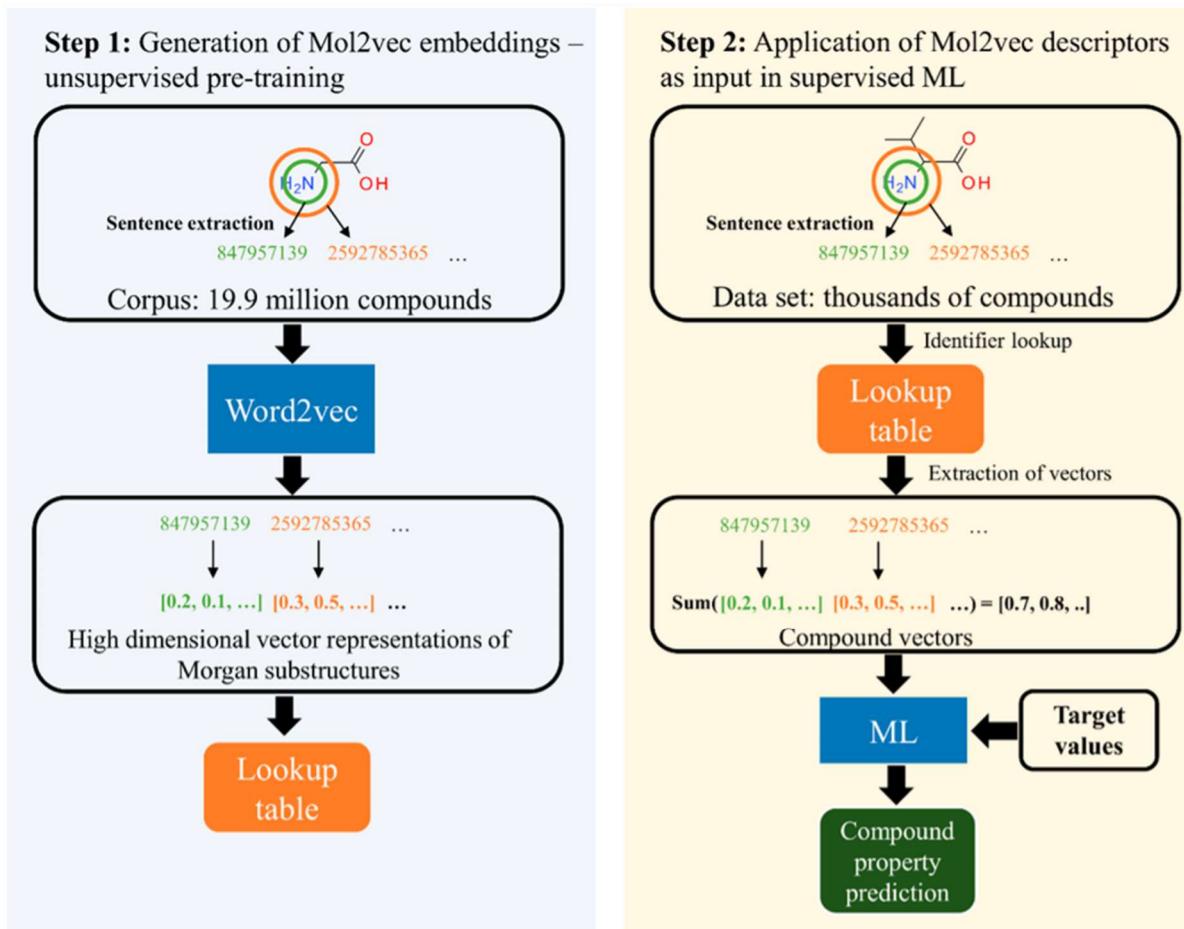
- Unsupervised word embeddings capture latent knowledge from materials science literature (Tshitoyan et al. Nature, 2019.)



Introduction (2/4)

1. NLP in chemistry

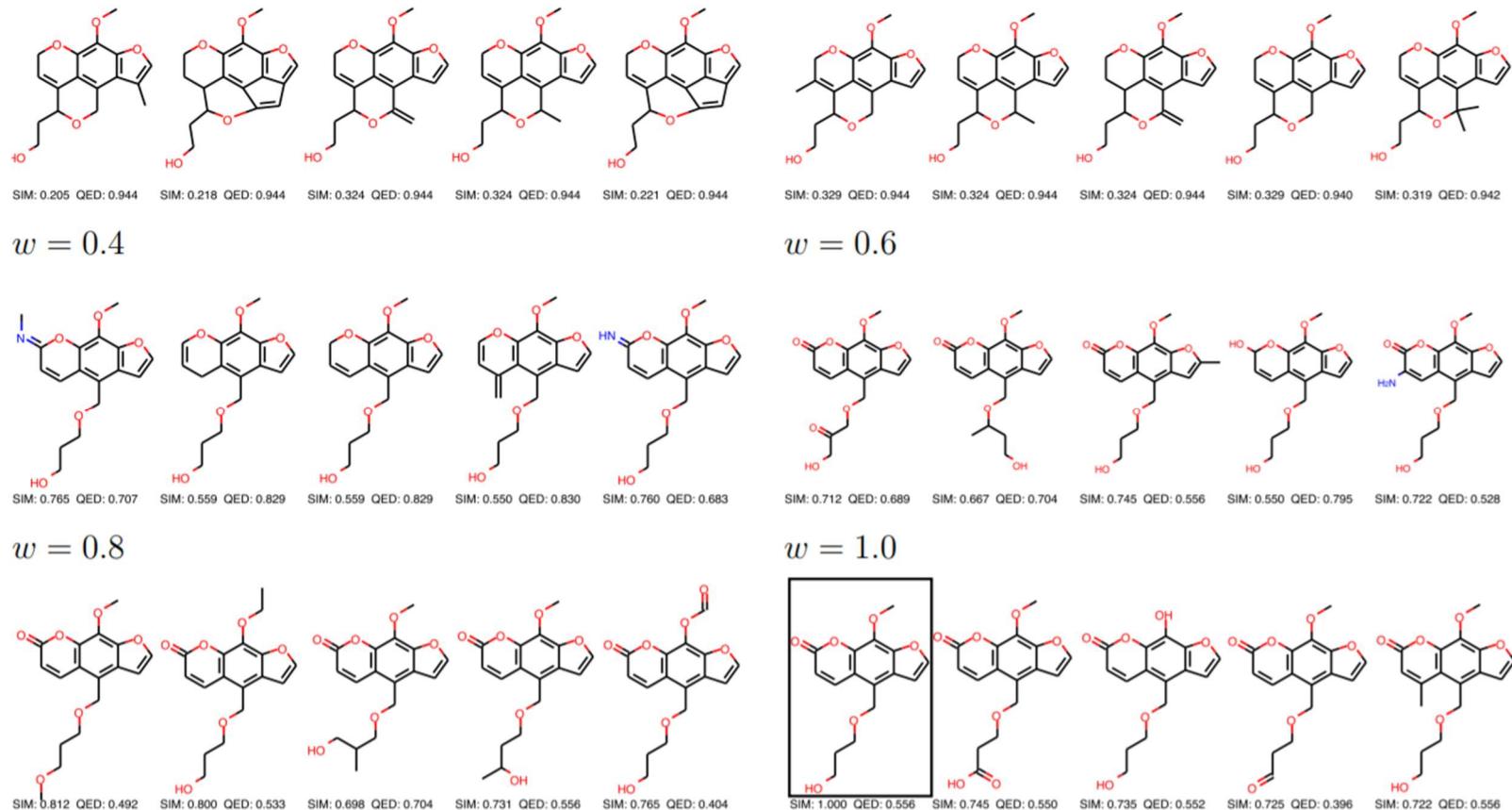
- Mol2vec: Unsupervised Machine Learning Approach with Chemical Intuition (Jaeger et al. JCIM, 2018.)



Introduction (3/4)

2. Reinforcement Learning in chemistry

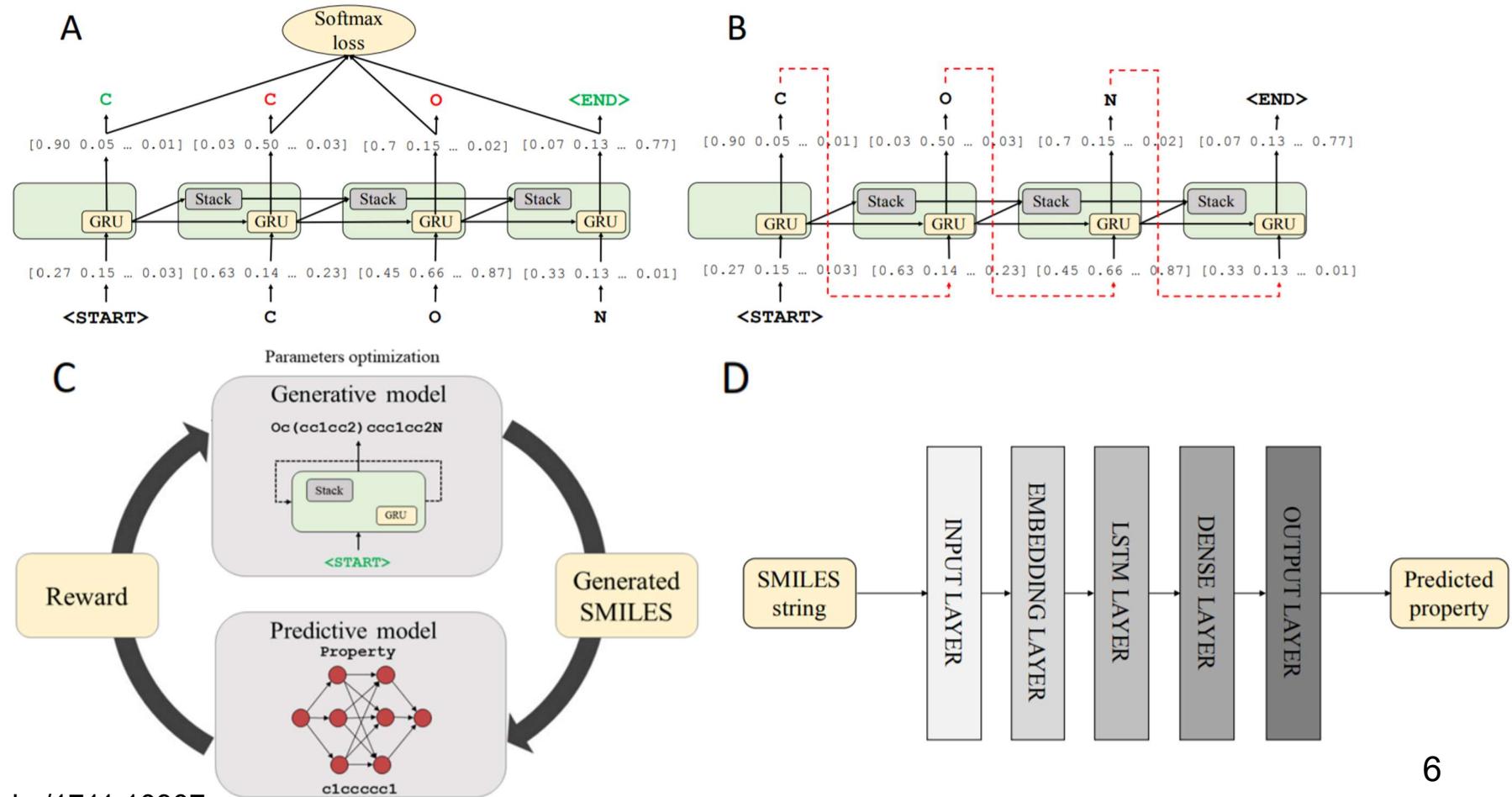
- Optimization of Molecules via Deep Reinforcement Learning
(Zhou et al. 2018.)



Introduction (4/4)

2. Reinforcement Learning in chemistry

- Deep Reinforcement Learning for de-novo Drug Design
(Popova et al. Science Advances, 2018.)



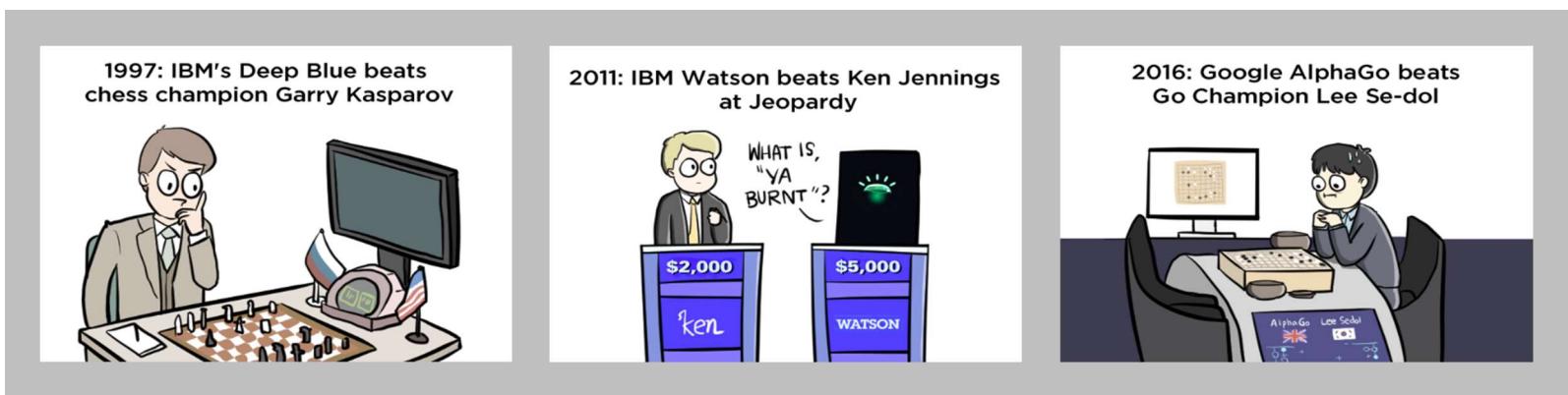
Outline

- Introduction
- Detour: RL basics
- DQN

Detour: RL basics (1/8)

1. Markov Decision Process

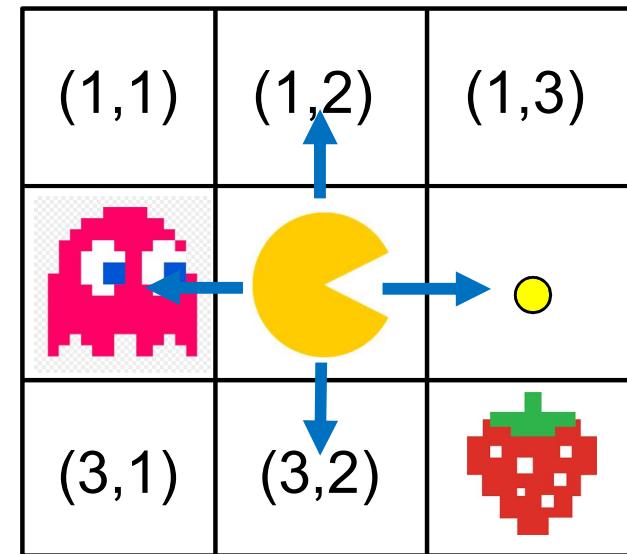
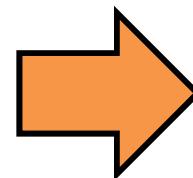
- **Markov Decision Process(MDP)**란 강화학습 같은 순차적으로 행동을 결정하는 문제를 정의할 때 사용하는 방법
- MDP의 구성요소는 크게 다섯가지임. 상태(state), 행동(action), 보상함수(reward), 상태변환확률(state transition probability), 감가율(discount factor)
- **모든 강화학습은 MDP를 “사용자”가 정의하는 것 부터 시작**



Detour: RL basics (1/8)

1. Markov Decision Process

- ex) Pac Man



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

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

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

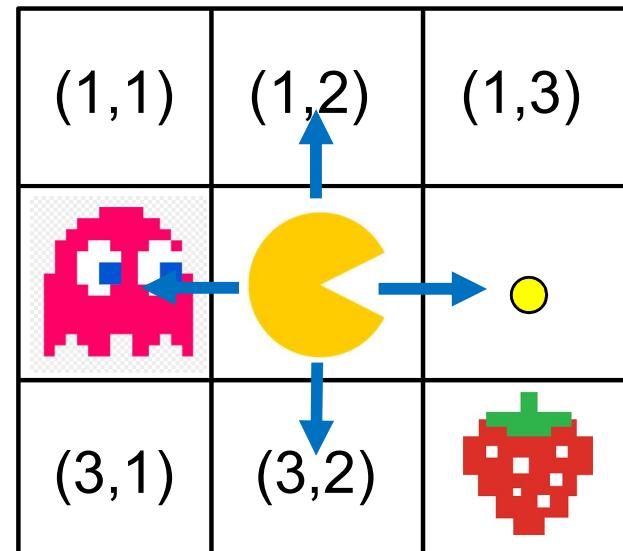
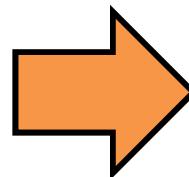
$$R_s^a = E[R_{t+1} | S_t = s, A_t = a]$$



Detour: RL basics (1/8)

1. Markov Decision Process

- ex) Pac Man



- 상태변환확률(state transition probability) : 팩맨이 (2, 2)에 있을 때, (2, 3)에 있는 공을 먹으러 갈 확률
- 감가율(action) : 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. 보통 $\gamma = 0.999$

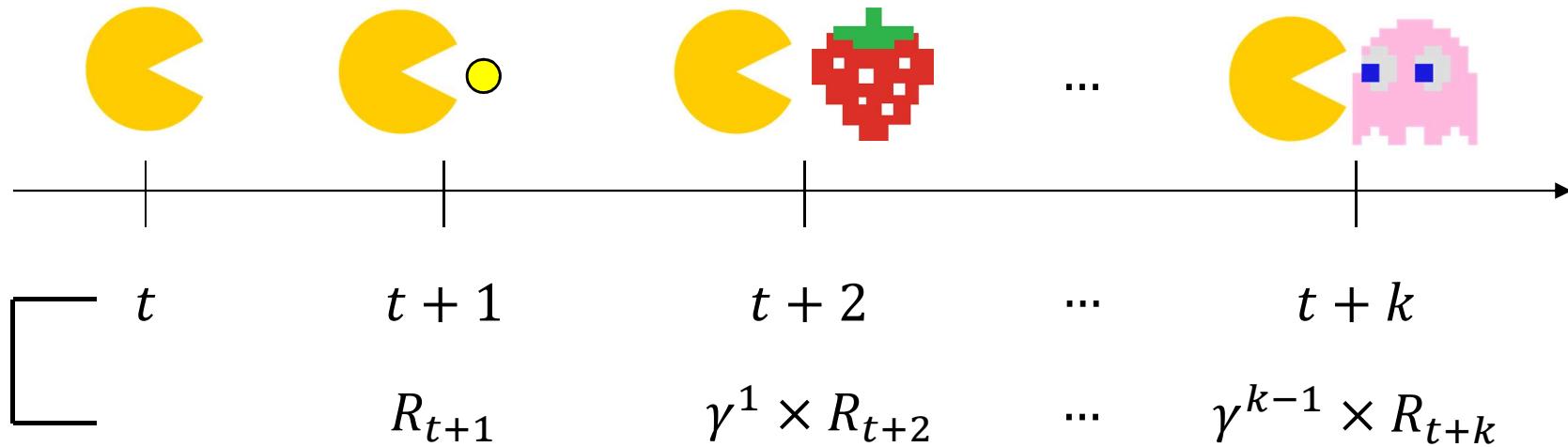
Detour: RL basics (1/8)

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

Detour: RL basics (2/8)

2. Value Function



- 반환값(return)이란 에이전트가 실제로 환경을 탐험하며 받은 보상

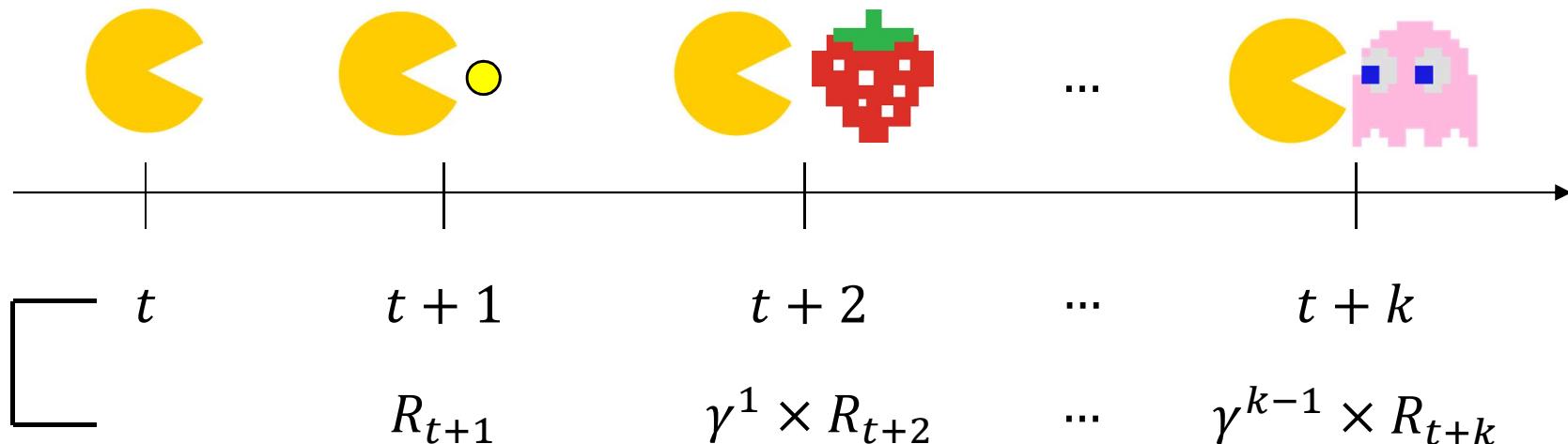
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

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

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

Detour: RL basics (3/8)

3. Bellman Expectation Equation



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

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

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

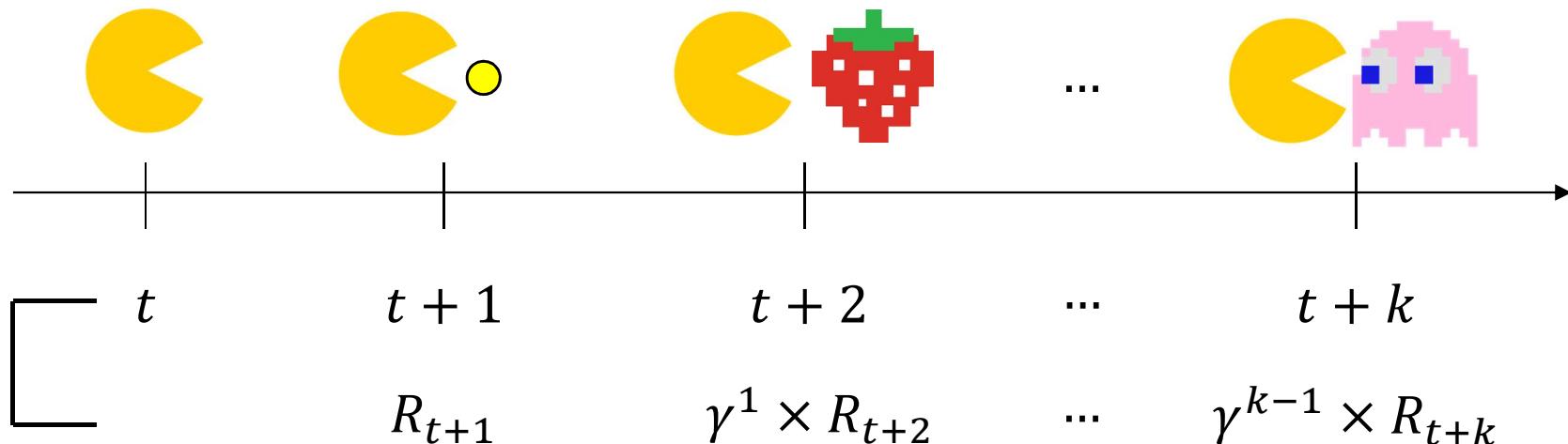
$$= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

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

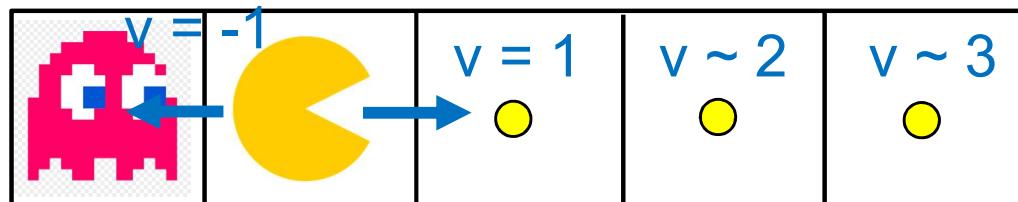
$\therefore v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$: 현재상태의 가치함수와 다음상태의 가치함수 사이의 관계를 말해줌

Detour: RL basics (3/8)

3. Bellman Expectation Equation



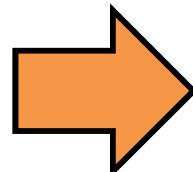
$$\therefore v(s) = E[R_{t+1} + \gamma v(s_{t+1}) | s_t = s]$$



Detour: RL basics (4/8)

4. Policy

- 정책(policy)이란 모든 상태에서 에이전트가 할 행동



| | | |
|---|---|---|
| (1,1) | (1,2) | (1,3) |
| A pink ghost with a blue dot on its chest, positioned in the first cell of the top row. | A yellow Pac-Man character facing up, positioned in the second cell of the top row. | A small yellow circular power pellet, positioned in the third cell of the top row. |
| (3,1) | (3,2) | A red strawberry with a green leaf, positioned in the third cell of the bottom row. |

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

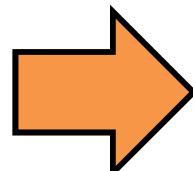
- ex) $\pi(a = \text{왼쪽} | s = (2,2)) = 0.1$

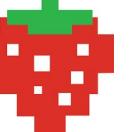
$$\pi(a = \text{오른쪽} | s = (2,2)) = 0.9$$

Detour: RL basics (5/8)

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.



| | | |
|--|--|---|
| (1,1) | (1,2) | (1,3) |
|  |  |  |
| (3,1) | (3,2) |  |

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

Detour: RL basics (6/8)

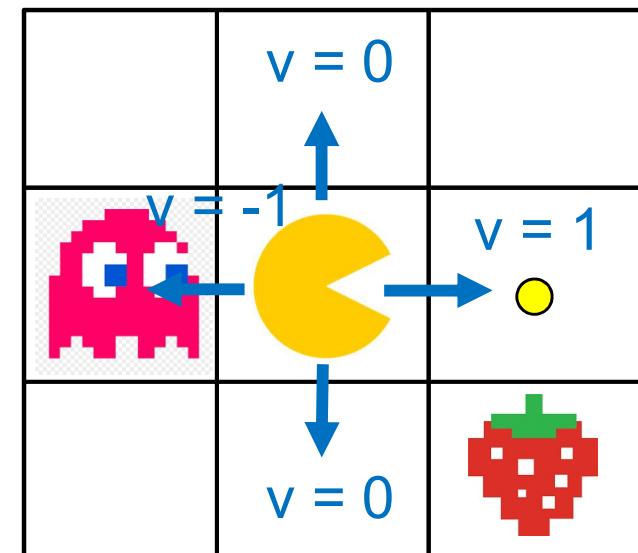
6. Bellman Optimality Equation

- 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?

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

$$\pi_*(s, a) \Leftarrow a = \operatorname{argmax}_{a \in A} q_*(s, a)$$

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



Detour: RL basics

중간 정리 & 자가진단

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

Detour: RL basics (7/8)

7. Temporal Difference Prediction

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

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

$$v_*(s) = \max_a E[R_{t+1} + \gamma v_*(s_{t+1}) | S_t = s, A_t = a]$$

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

$$\pi_*(s, a) \Leftarrow a = \operatorname{argmax}_{a \in A} q_*(s, a)$$

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

Detour: RL basics (7/8)

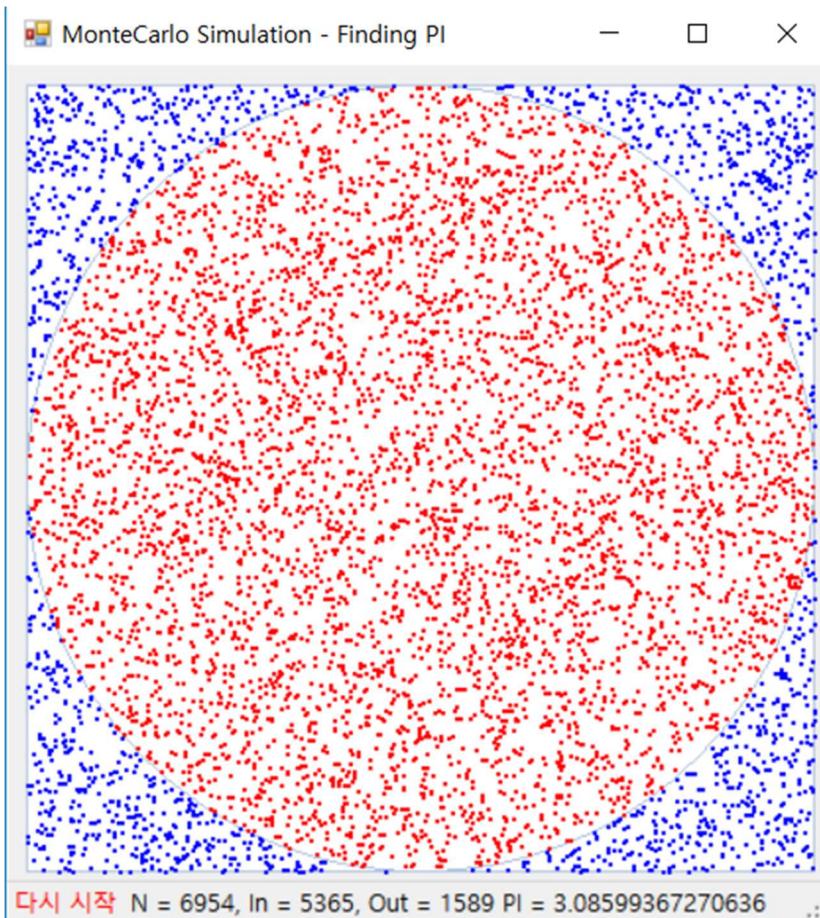
7. Temporal Difference Prediction

- 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?

Detour: RL basics (7/8)

7. Temporal Difference Prediction

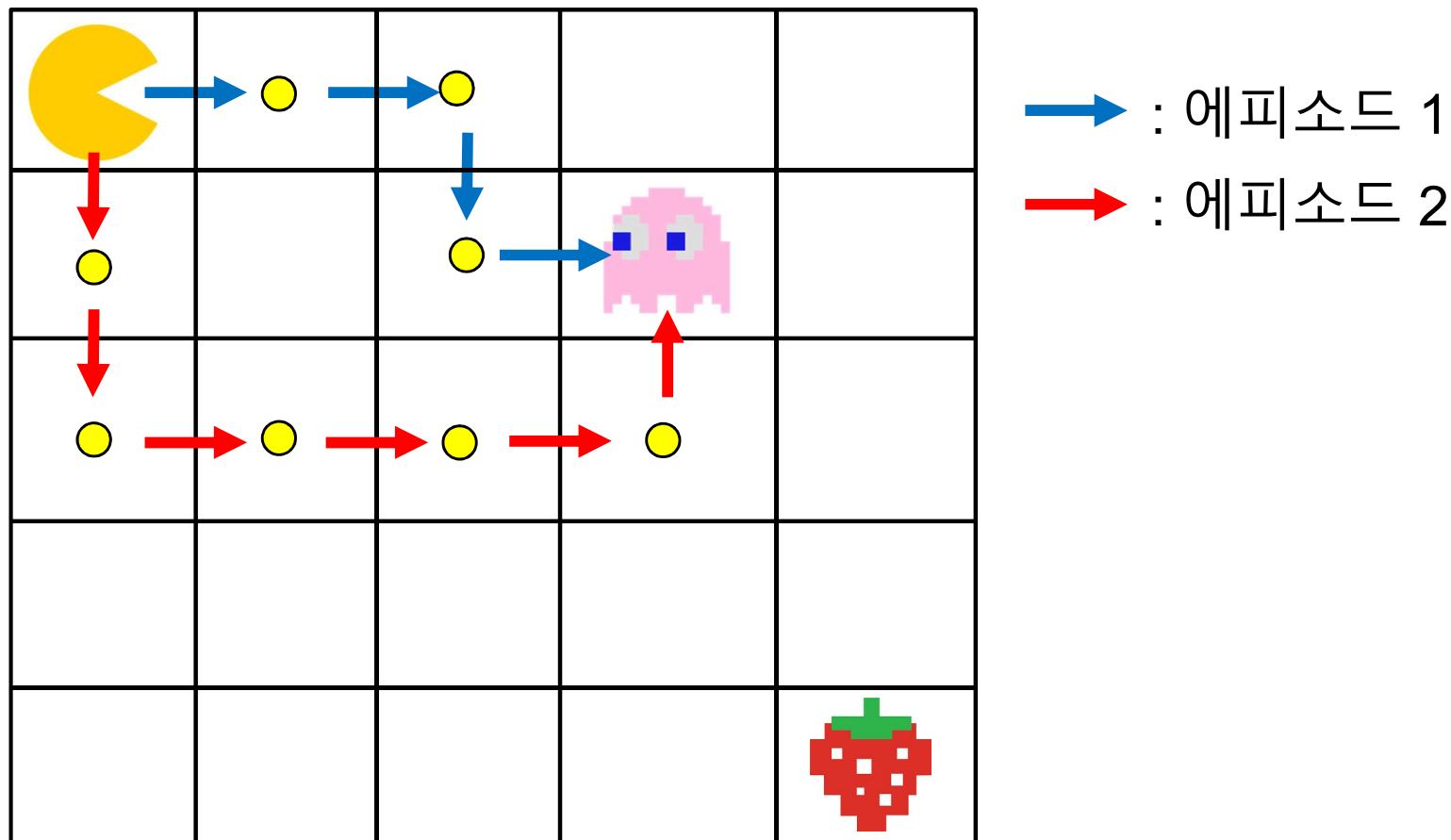
- 원의 넓이를 구하고 싶은데 원의 방정식을 모른다면?
- 점들을 샘플링을 해서 붉은 점의 갯수를 전체 점의 갯수로 나눔



Detour: RL basics (7/8)

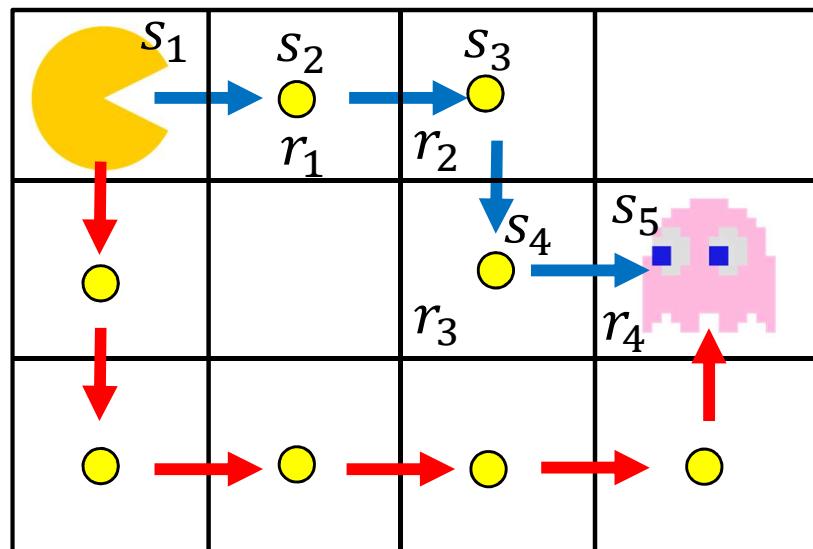
7. Temporal Difference Prediction

- 가치함수를 추정할 때는 에이전트가 환경에서 한 에피소드를 진행한 것을 샘플링함.



Detour: RL basics (7/8)

7. Temporal Difference Prediction



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

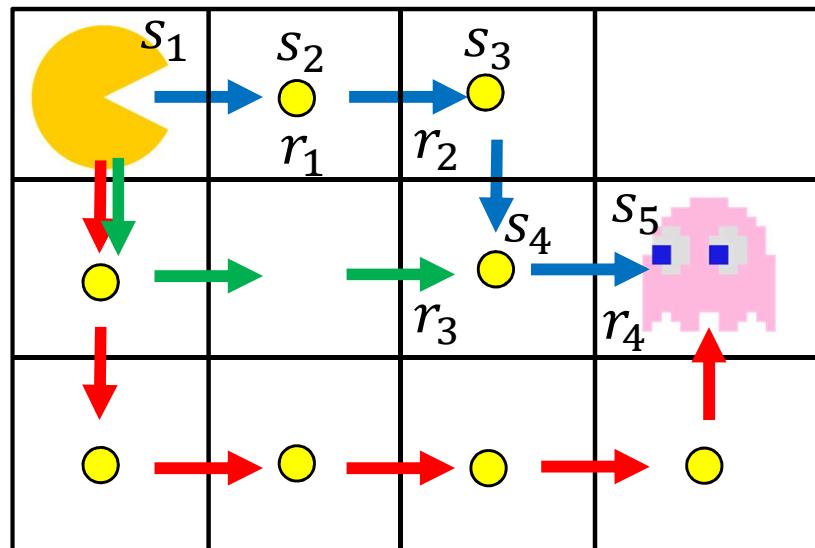
$$G(s_2) = r_2 + \gamma r_3 + \gamma^2 r_4$$

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

$$G(s_4) = r_4$$

Detour: RL basics (7/8)

7. Temporal Difference Prediction



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

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

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

$$G(s_4) = r_4$$

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

$N(s)$ 는 상태 s 를 여러번의 에피소드 동안 방문한 횟수

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

Detour: RL basics (7/8)

7. Temporal Difference Prediction

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

$$\begin{aligned} v_{n+1}(s) &= \frac{1}{n} \sum_{i=1}^n G_i = \left(G_n + \sum_{i=1}^{n-1} G_i \right) \\ &= \frac{1}{n} \left(G_n + (n-1) \frac{1}{(n-1)} \sum_{i=1}^{n-1} G_i \right) \\ &= \frac{1}{n} (G_n + (n-1) \textcolor{red}{v_n}) \\ &= v_n + \frac{1}{n} (G_n - v_n) \end{aligned}$$

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

Detour: RL basics (7/8)

7. 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



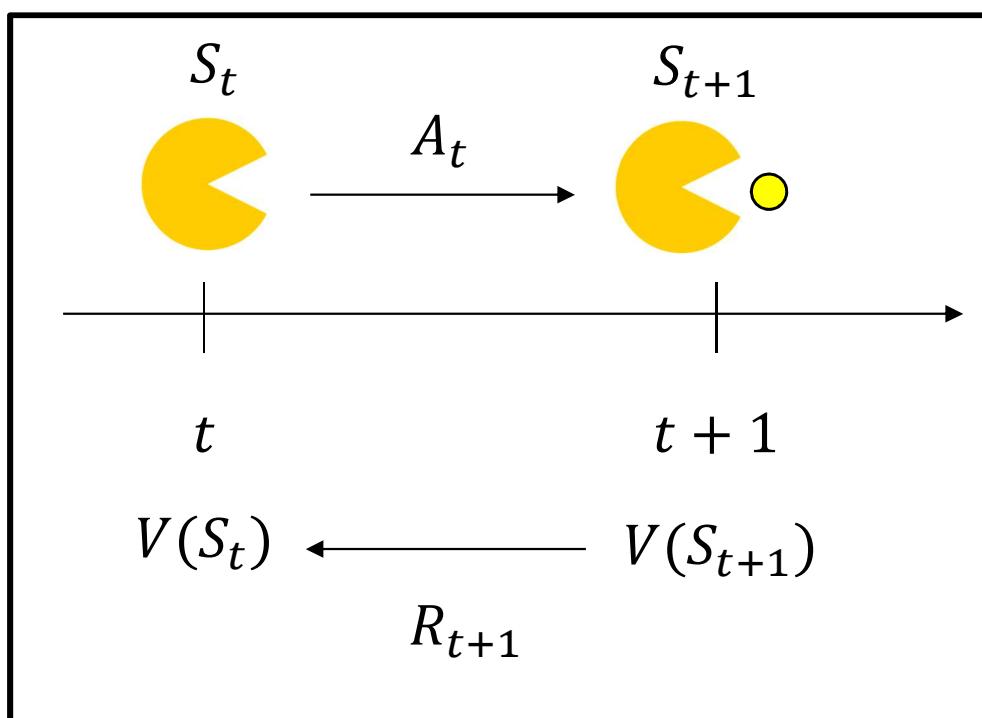
$$V(S_t) \leftarrow V(S_t) + \alpha(R + \gamma V(S_{t+1}) - V(s))$$

Detour: RL basics (7/8)

7. 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

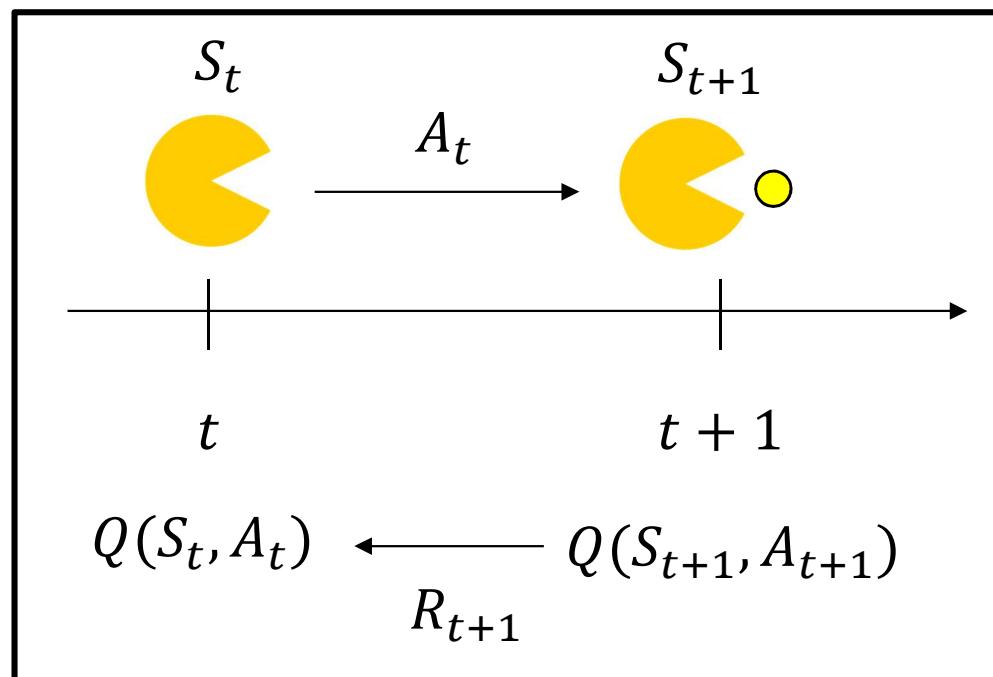
$$V(S_t) \leftarrow V(S_t) + \alpha(R + \gamma V(S_{t+1}) - V(s))$$



Detour: RL basics (8/8)

9. Q-Learning

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



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

Detour: RL basics

중간 정리 & 자가진단

- 1) 시간차 예측이란? 학습 과정은?
- 2) 큐러닝이란?

Outline

- Introduction
- Detour: RL basics
- DQN

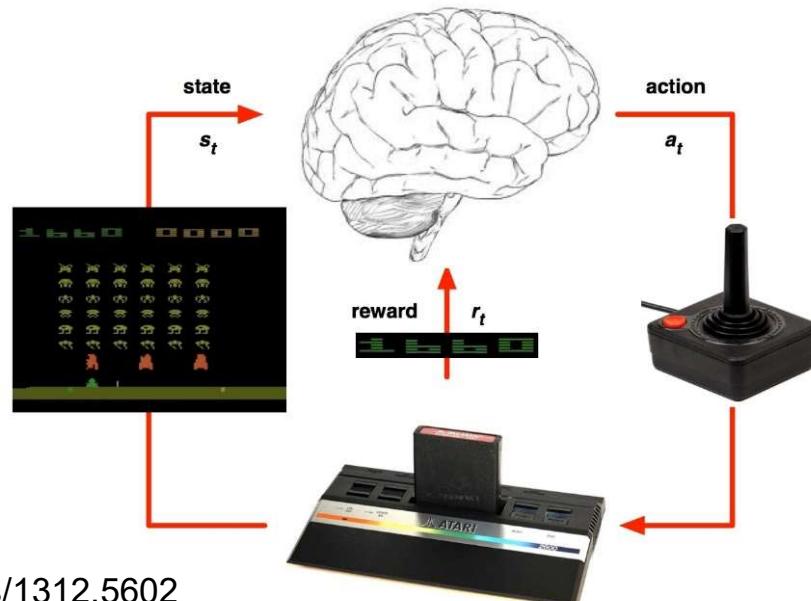
DQN

1. Deep Q-Networks

- Playing Atari with Deep Reinforcement Learning
(Minh et al. NIPS Deep Learning Workshop, 2013.)에 소개된 내용

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

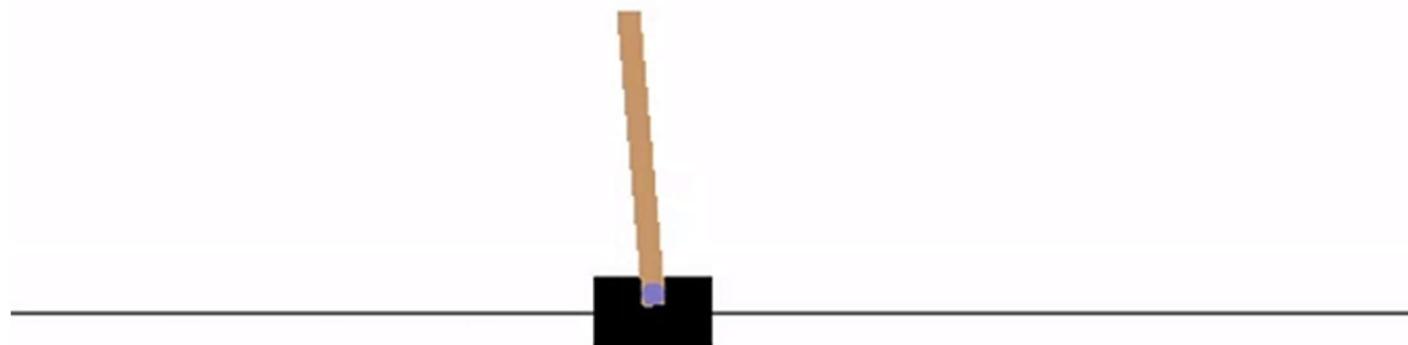
- 여기서 Q값을 인공신경망을 이용하여 딥러닝 방식으로 구한 것



DQN

2. Cartpole

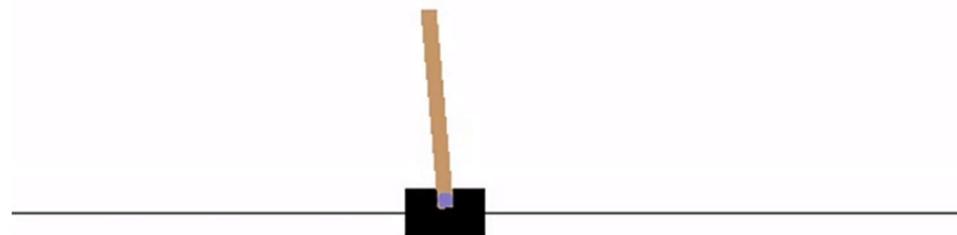
- OpenAI Gym에서 제공하는 실험환경



DQN

2. Cartpole

- Markov Decision Process (MDP)



1) 상태(state) : 카트의 위치, 속도, 폴의 각도, 각속도

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

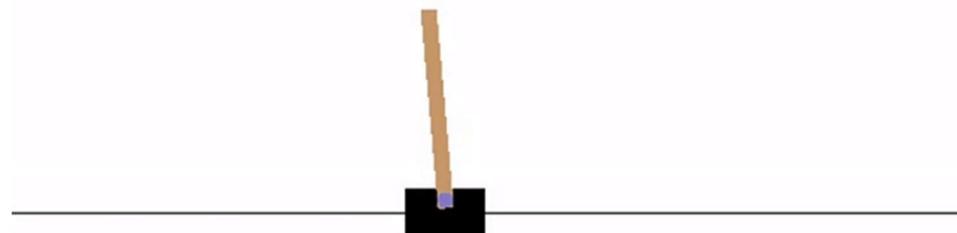
2) 행동(action) : 왼쪽(0), 오른쪽(1)

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

DQN

2. Cartpole

- Markov Decision Process (MDP)



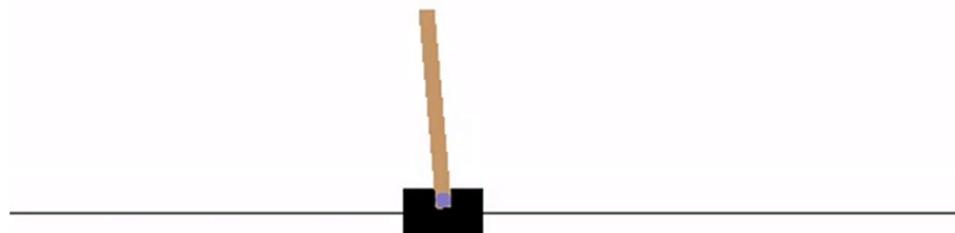
3) 보상(reward) : 카트폴이 쓰러지지 않고 버티는 시간

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

DQN

2. Cartpole

- Markov Decision Process (MDP)



4) 감가율(discount factor) : Q함수에 대한 discount

- 0.99

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)

DQN

3. 코드 설명

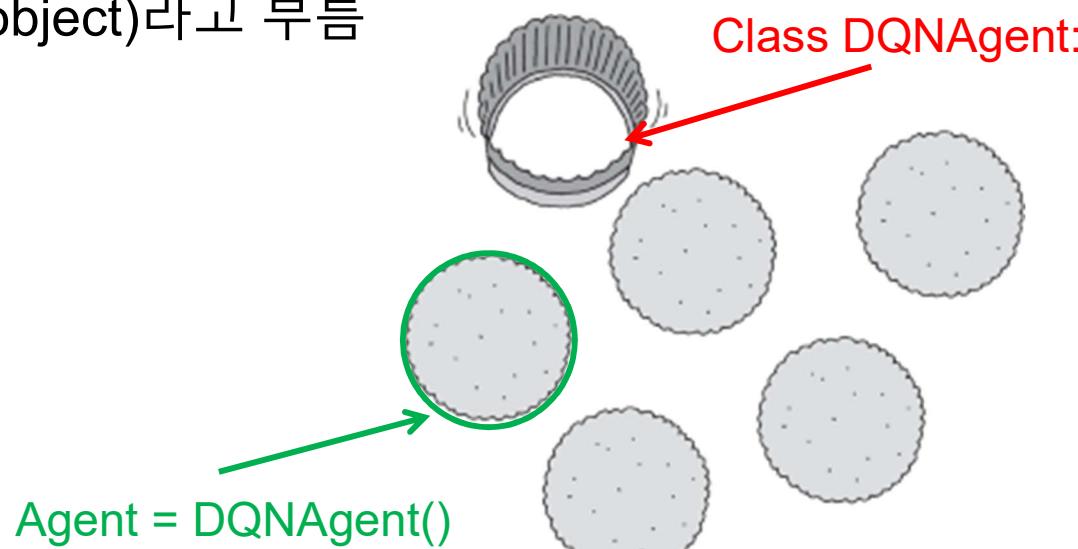
- Training

```
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 ???



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")
```

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")
```

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.

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")
```

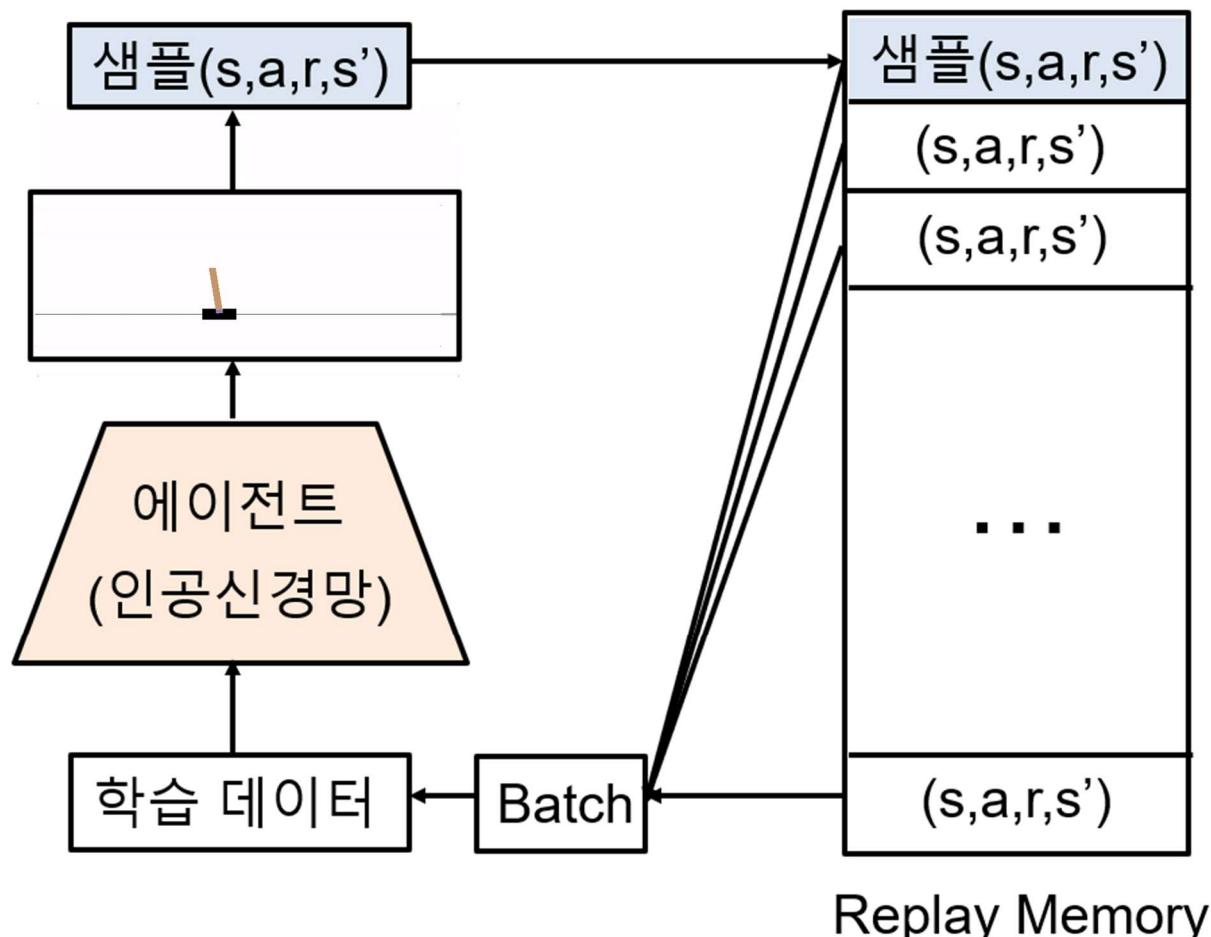
DQN

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)
```



Replay Memory

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")
```

DQN

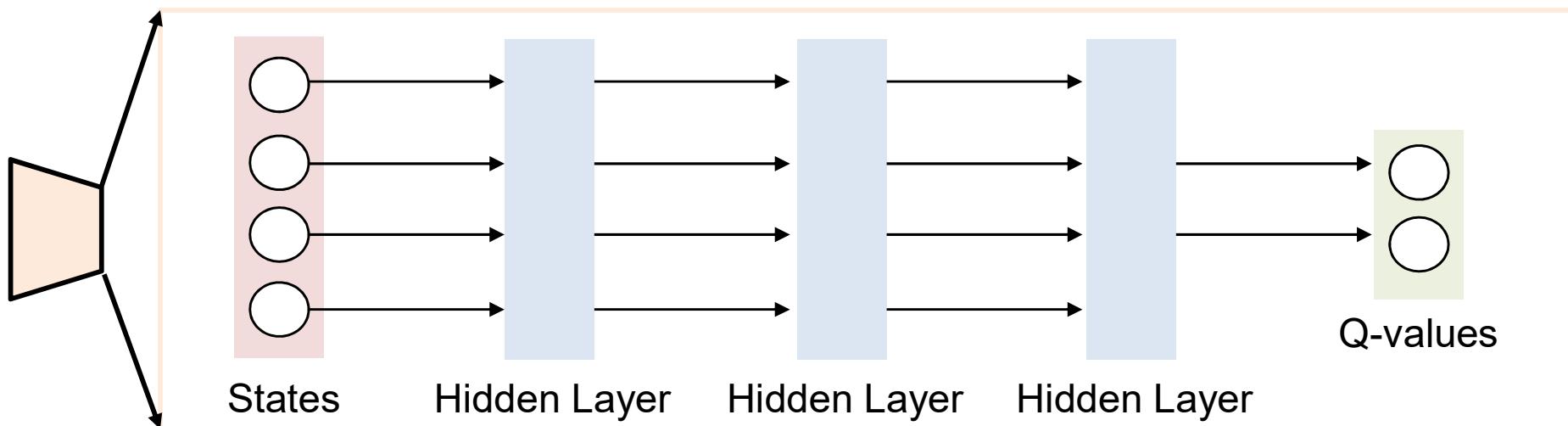
3. 코드 설명

- 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
```



DQN

3. 코드 설명

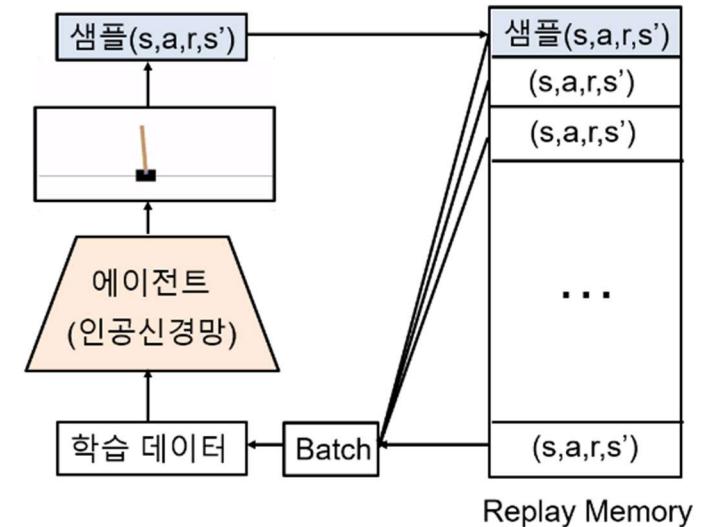
- Training -> Agent

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 예측값**을 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



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")
```

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?

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, ...]

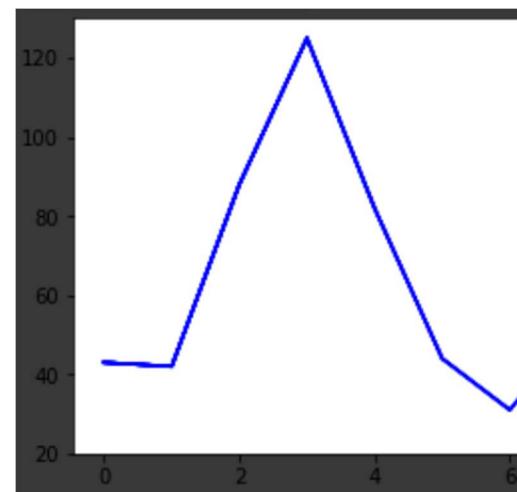
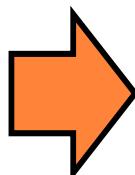
DQN

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
```



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]]

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 ???**

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

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 $\langle s, a, r, s', \text{done} \rangle$ 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)
```

DQN

3. 코드 설명

- Training -> Agent

```
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 = [], [], []
```

| S | A | R | S' | d |
|------|------|------|------|------|
| | | | | |
| | | | | |
| | | | | |

Mini Batch

| X | X' | θ | θ' |
|------|------|------|------|
| | | | |
| | | | |
| | | | |

States

| X | X' | θ | θ' |
|------|------|------|------|
| | | | |
| | | | |
| | | | |

Next 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)
```

DQN

3. 코드 설명

- Training -> Agent

```
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])
```

i++



| | | |
|--|---|----------|
| | : | S |
| | : | A |
| | : | R |
| | : | S |
| | : | Ω |

Mini Batch

Mini_batch[i][0]

Mini_batch[i][1]

Mini_batch[i][2]

Mini_batch[i][3]

Mini_batch[i][4]

| x | x' | θ | θ' |
|-----|------|----------|-----------|
| | | | |
| | | | |
| ... | ... | ... | ... |
| | | | |

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)
```

DQN

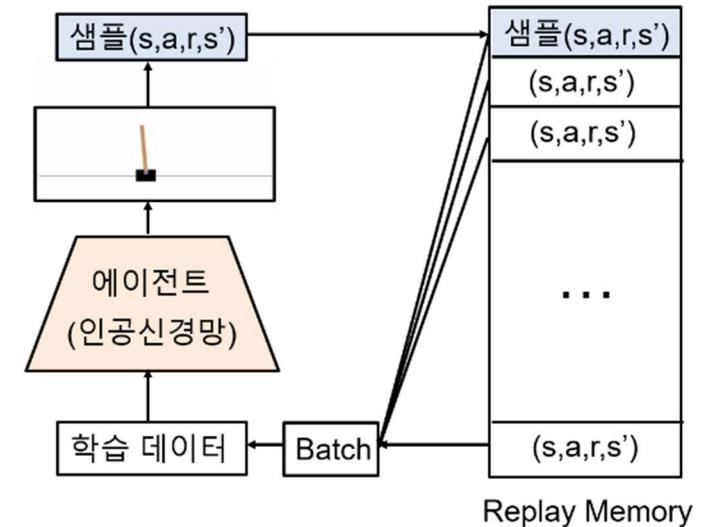
3. 코드 설명

- 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



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)
```

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 $\text{dones}[i] == \text{False}$ / 즉, 한 에피소드가 아직 끝나지 않음

- else 문으로 들어감

| q_1 | q_2 |
|-------|-------|
| | |
| | |
| ... | ... |
| | |
| | |

target

| q_1 | q_2 |
|-------|-------|
| | |
| | |
| ... | ... |
| | |
| | |

target_val

DQN

3. 코드 설명

- Training -> Agent

2) If $\text{dones}[i] == \text{True}$ / 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)
```

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
```

DQN

3. 코드 설명

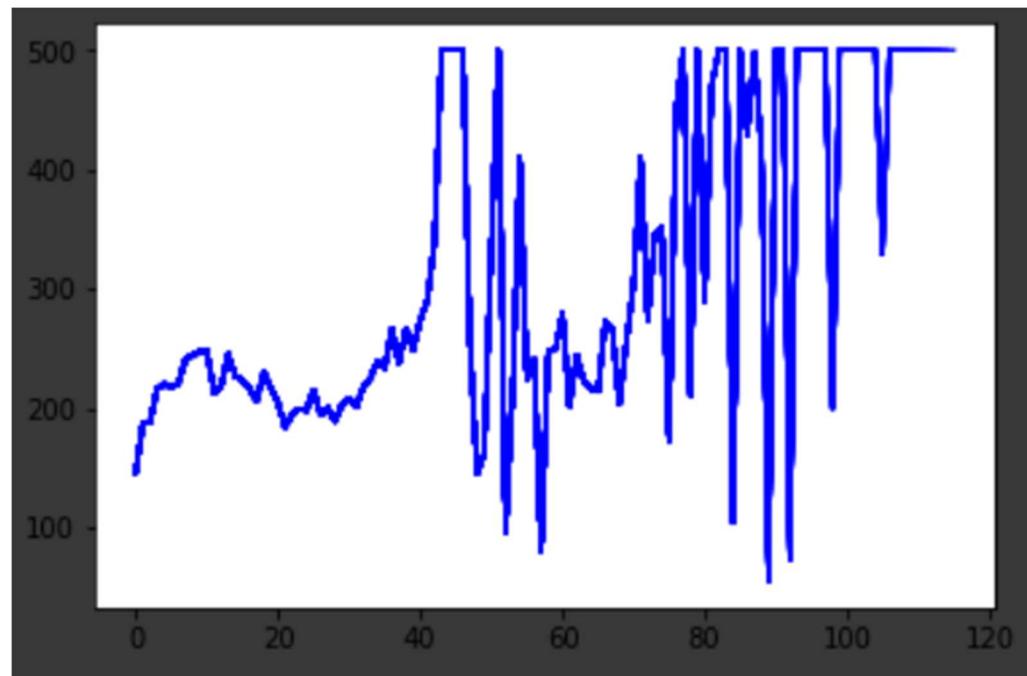
- Training

```
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()
```

DQN

3. 코드 설명

- Result



Thank you :)

k4ke@korea.ac.kr