

Reinforcement Learning

Introduction to Reinforcement Learning and
Computer Vision (Day 2)

Jeiyoon Park

Department of Computer Science and Engineering



고려대학교
KOREA UNIVERSITY



Natural Language
Processing
& Artificial Intelligence

IPA Korea IT Business Promotion Association
한국IT비즈니스진흥협회

Overview

진단 평가

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

RL Basics

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?

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?

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
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)
- Advantage Actor-Critic (A2C)
- Asynchronous Advantage Actor-Critic (A3C)
- Applications

Outline

- Overview
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)
- Advantage Actor-Critic (A2C)
- Asynchronous Advantage Actor-Critic (A3C)
- Applications

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 해결하였지만 (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

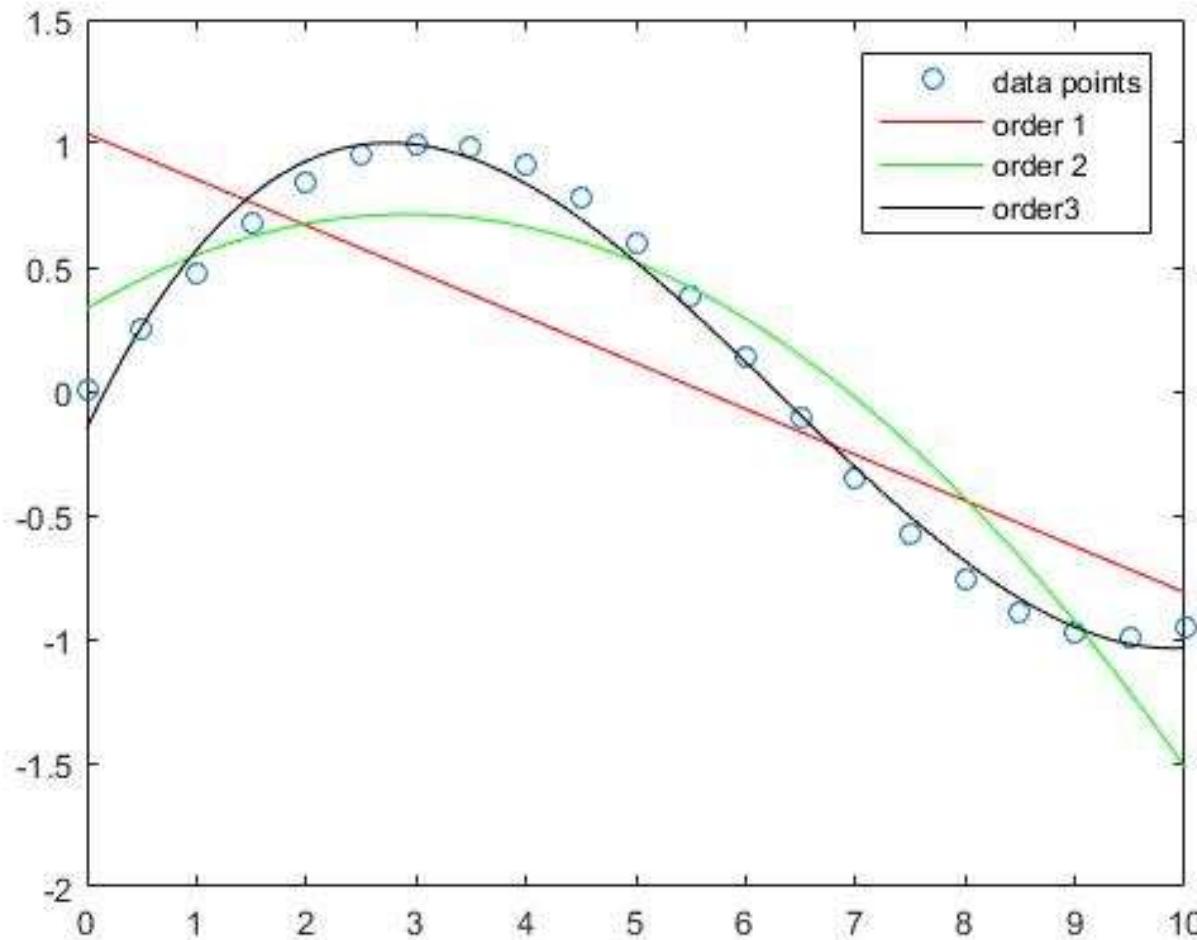
1. 몬테카를로, 살사, 큐러닝의 한계?

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

Deep Reinforcement Learning

2. 큐함수의 매개변수화 & 근사화

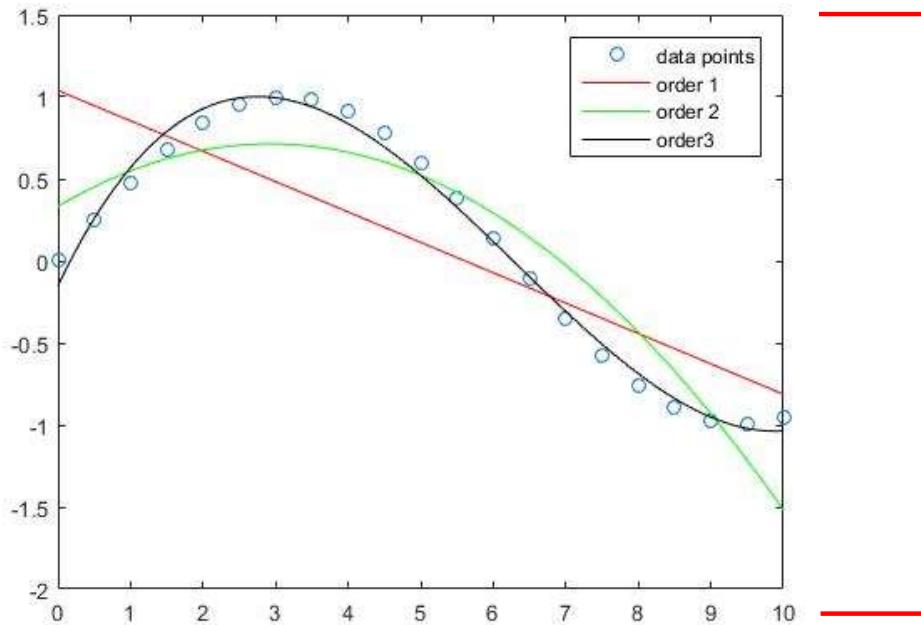
(1) 근사화?



Deep Reinforcement Learning

2. 큐함수의 매개변수화 & 근사화

(2) 매개변수화?



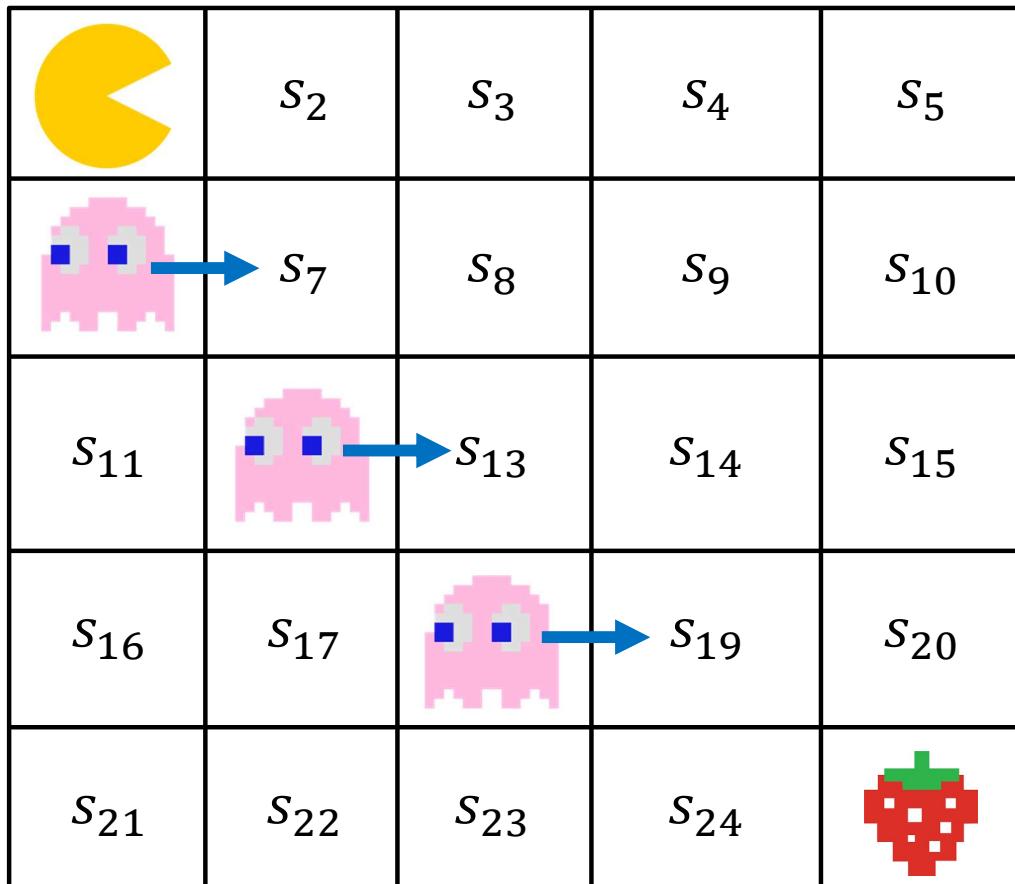
$ax^3 + bx^2 + cx + d$
→ (a, b, c, d) : 매개변수

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

Deep Reinforcement Learning

3. Deep-SARSA

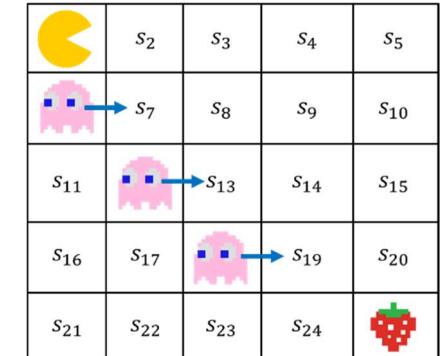
(1) 문제: 팩맨은 좌우로 움직이는 귀신 3마리를 피해서 딸기를 먹어야함



Deep Reinforcement Learning

3. Deep-SARSA

(2) 복잡한 순차적 의사결정 문제 → MDP



사람도 어떠한 장애물을 피할 때 정확하지는 않아도 물체가 내쪽으로 오고 있는지 혹은 멀어지고 있는지, 그리고 속도가 어느 정도 되는지 알아야 한다.

MDP를 정의할 때도 에이전트가 충분히 잘 학습할 수 있도록 상태 정보를 잘 정의해주어야 한다.

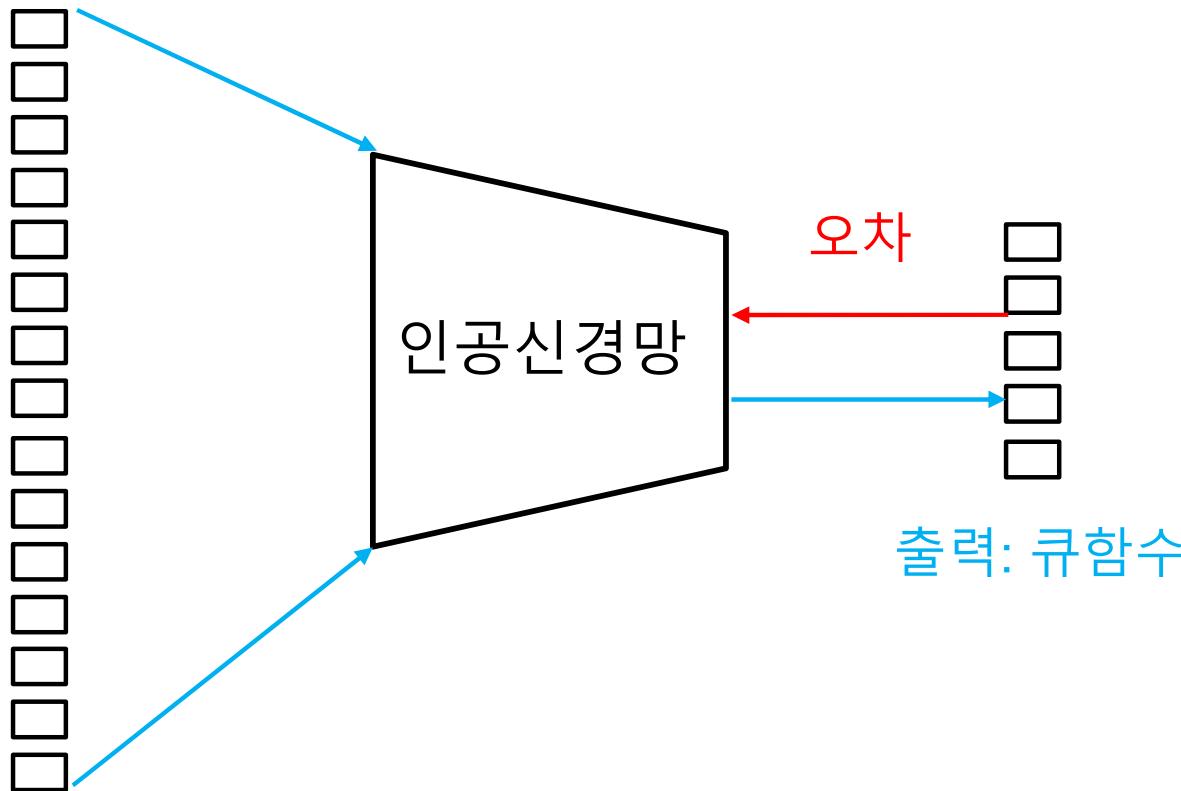
상태에 들어가야 하는 정보: 도착지점의 상대위치, 도착지점의 레이블, 장애물의 상대위치, 장애물의 레이블, 장애물의 속도

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

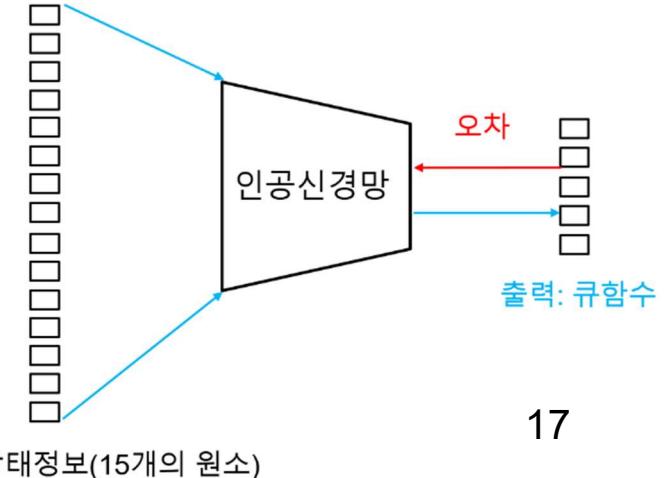
3. Deep-SARSA

(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

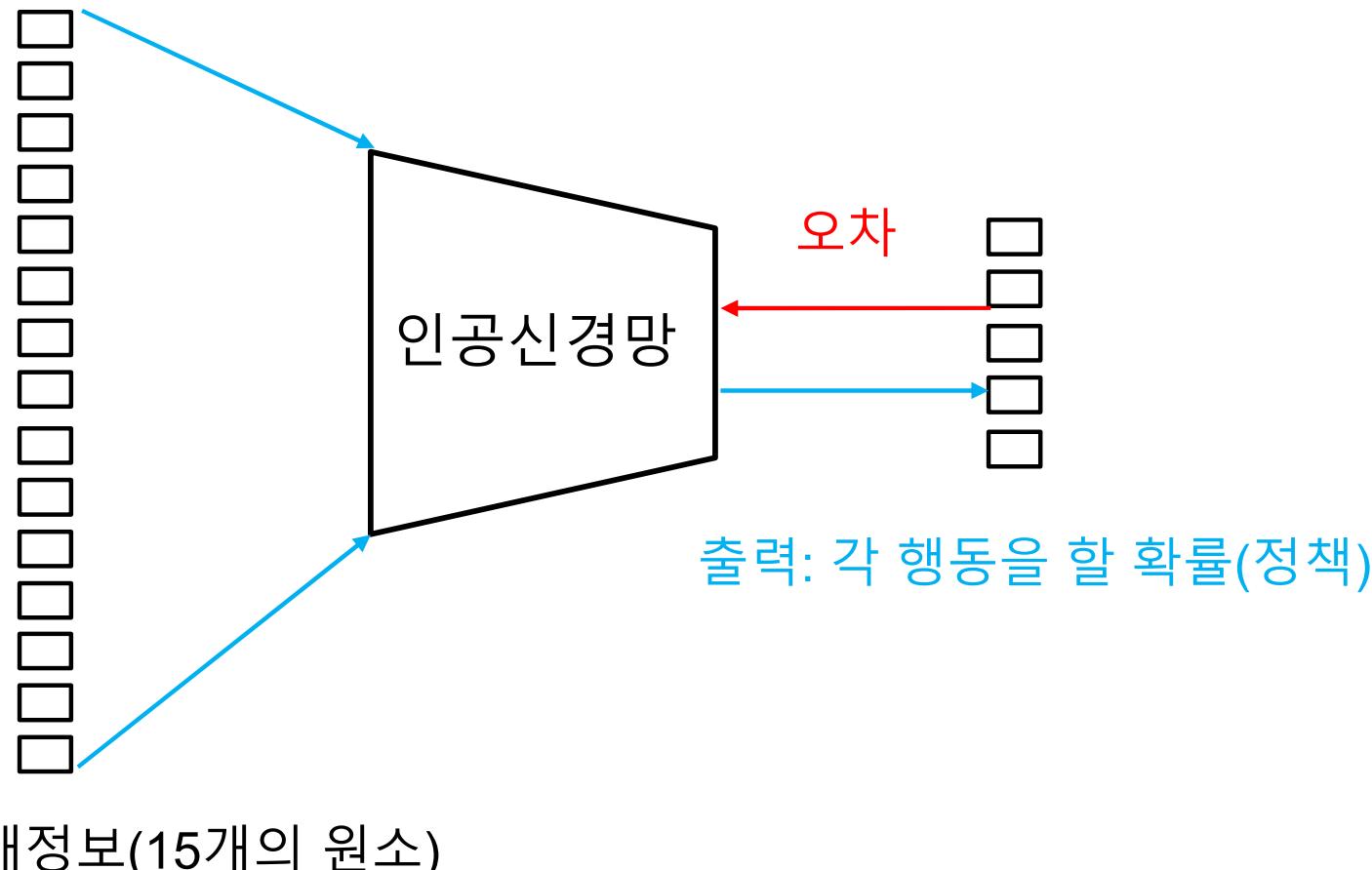
(1) 지금 까지 배웠던 강화학습 방법들은 모두 가치 기반 강화학습(Value-based RL)임. 즉, 가치함수, 큐함수(행동가치함수)를 업데이트 하며 학습함

(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

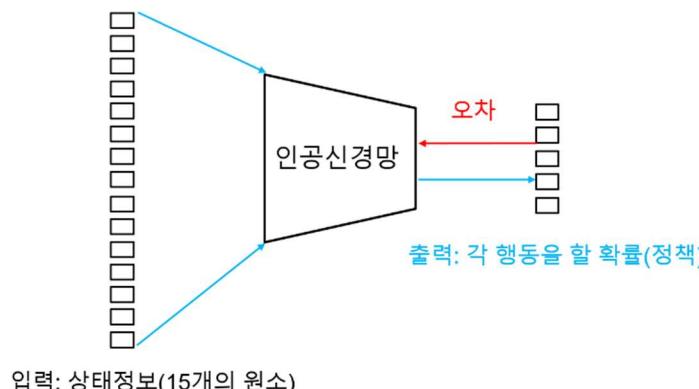
4. Policy Gradient

(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

5. REINFORCE

- 코드

<http://bitly.kr/PgeEobT8WVFE>

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
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- **Deep Q-Networks (DQN)**
- Advantage Actor-Critic (A2C)
- Asynchronous Advantage Actor-Critic (A3C)
- Applications

Deep Q-Networks (DQN)

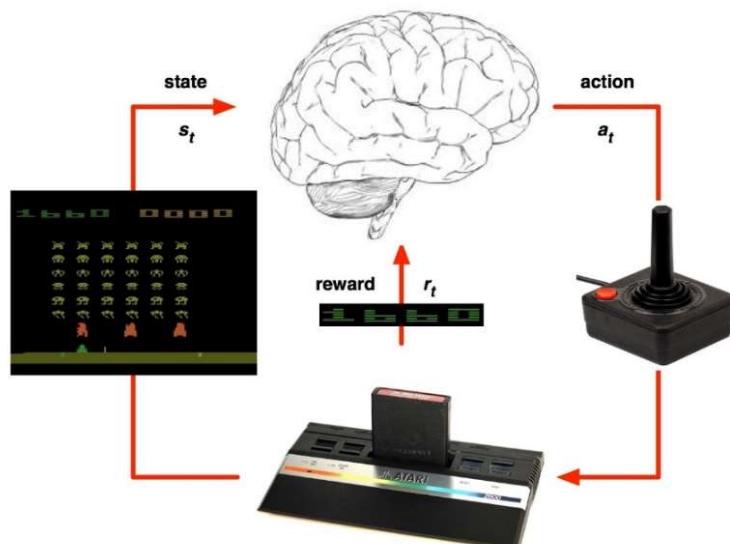
1. Deep Q-Networks

- Playing Atari with Deep Reinforcement Learning

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

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

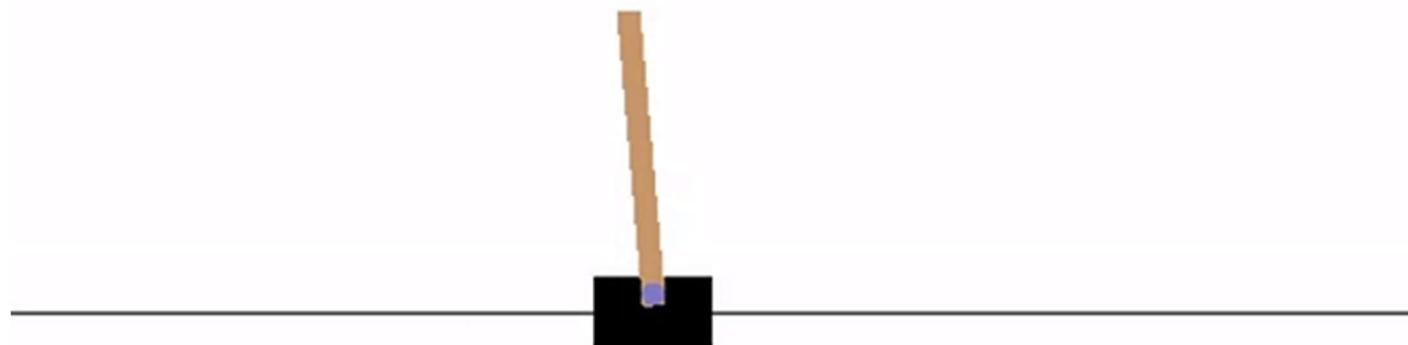
- 여기서 Q값을 인공신경망을 이용하여 딥러닝 방식으로 구한 것



Deep Q-Networks (DQN)

2. Cartpole

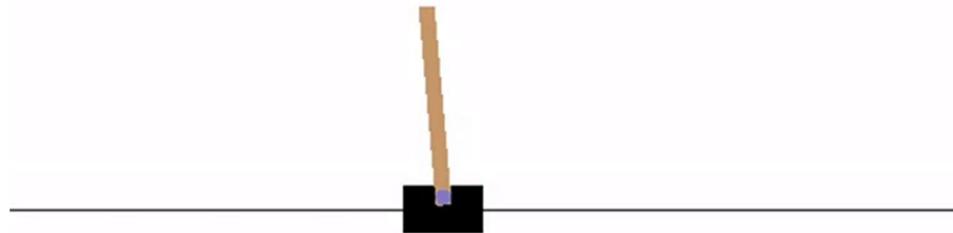
- OpenAI Gym에서 제공하는 실험환경



Deep Q-Networks (DQN)

2. Cartpole

- Markov Decision Process (MDP)



1) 상태(state) : 카트의 위치, 속도, 폴의 각도, 각속도

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

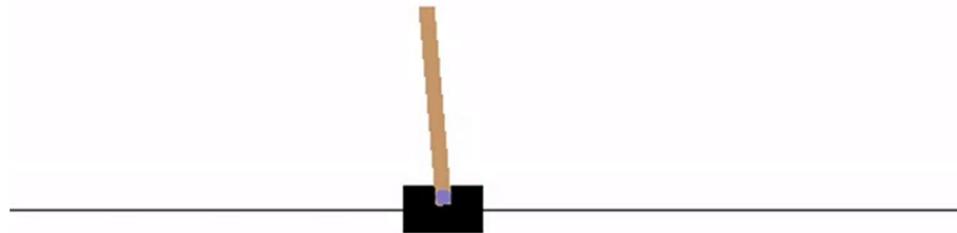
2) 행동(action) : 왼쪽(0), 오른쪽(1)

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

Deep Q-Networks (DQN)

2. Cartpole

- Markov Decision Process (MDP)



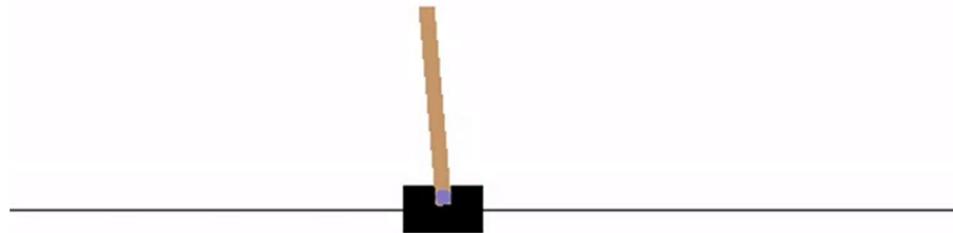
3) 보상(reward) : 카트폴이 쓰러지지 않고 버티는 시간

- 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)

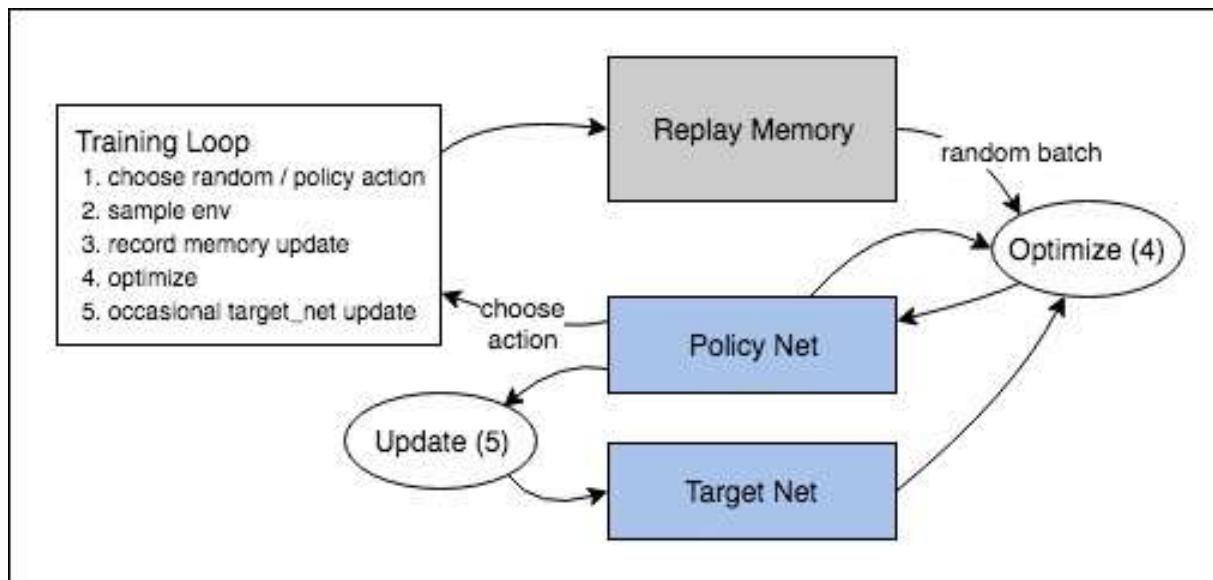


4) 감가율(discount factor) : Q함수에 대한 discount

- 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

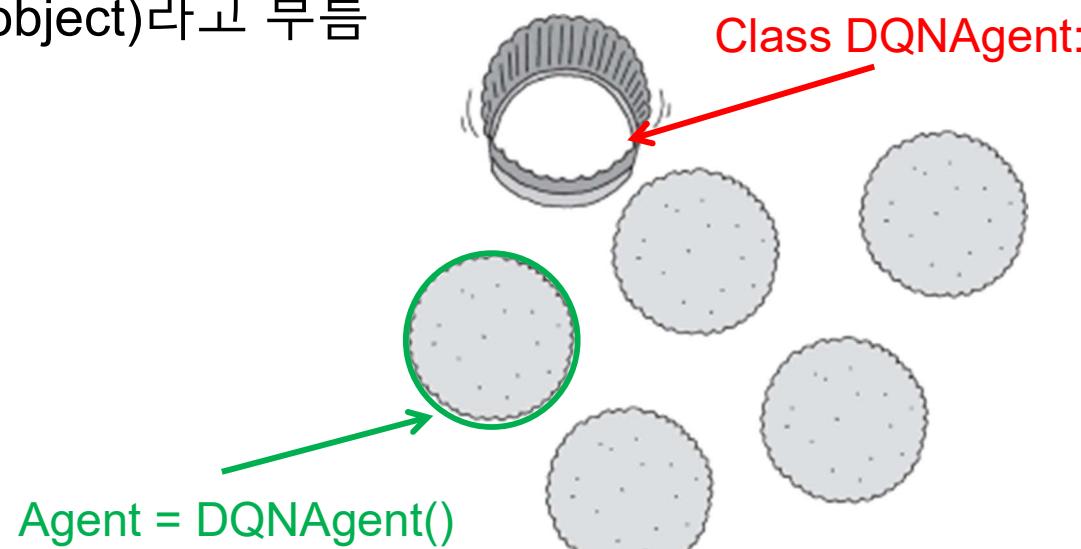
- Training

```
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")
```

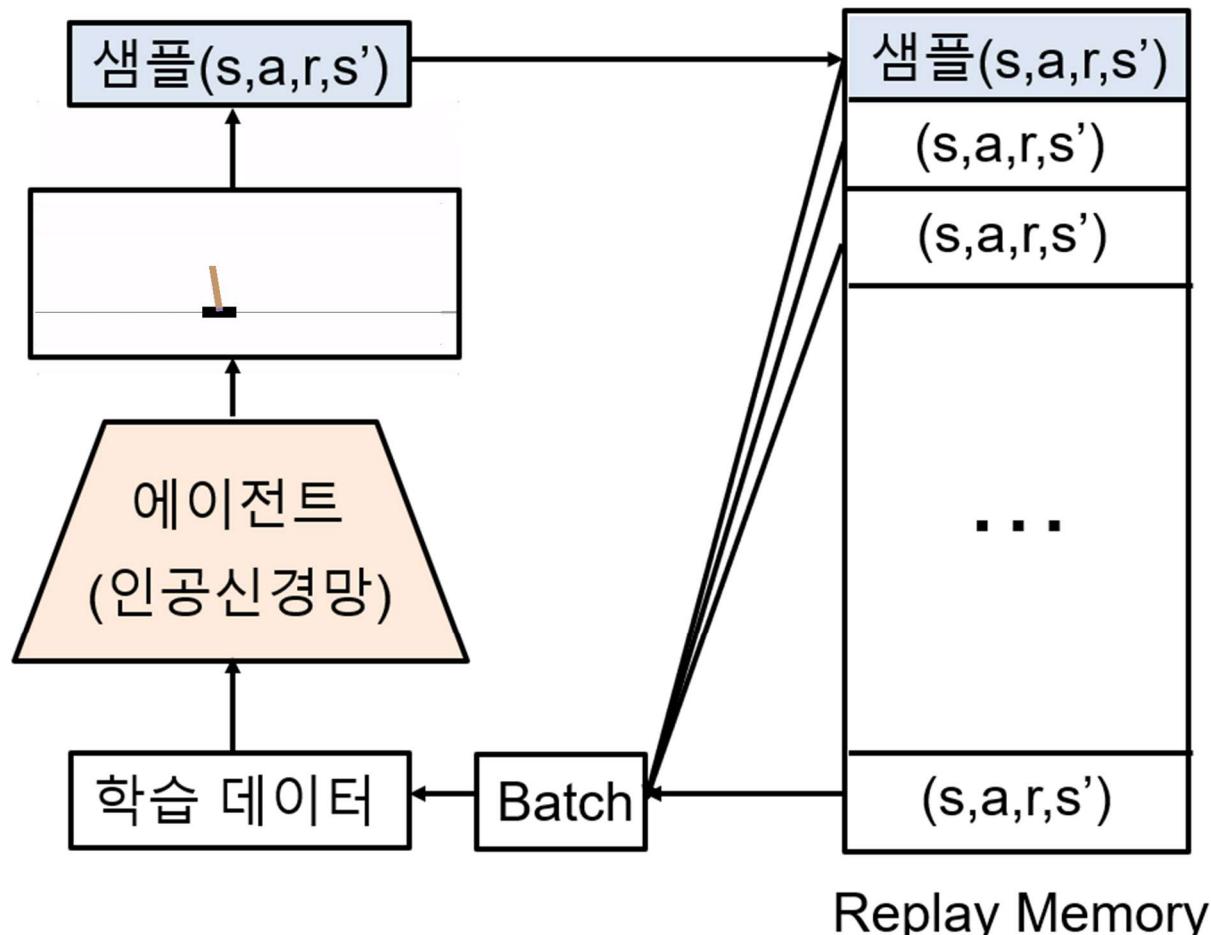
Deep Q-Networks (DQN)

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)

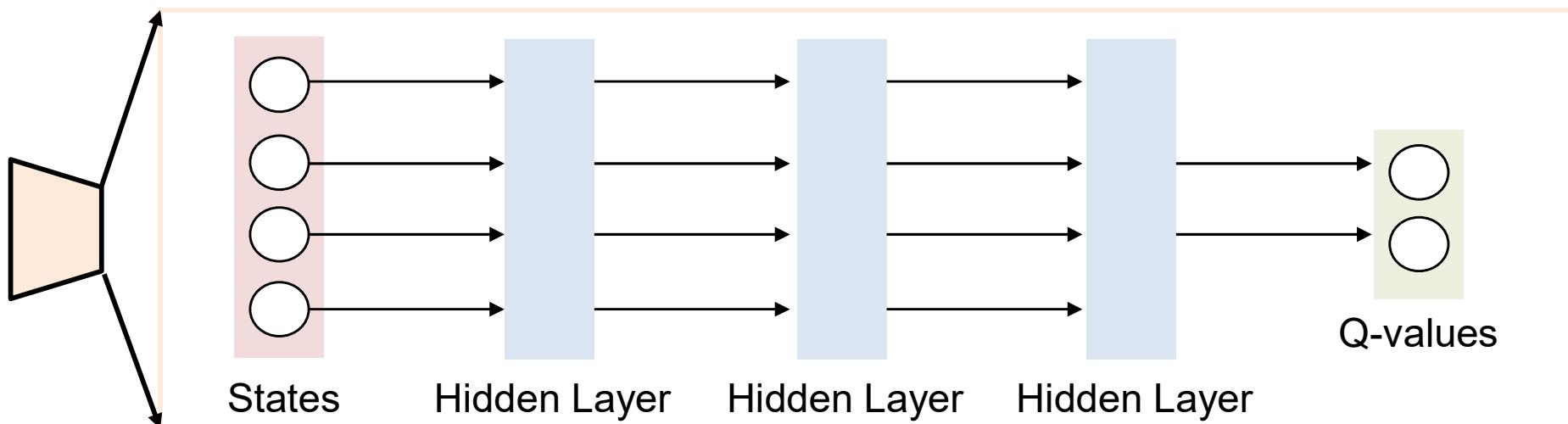
3. 코드 설명

- 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, ...]

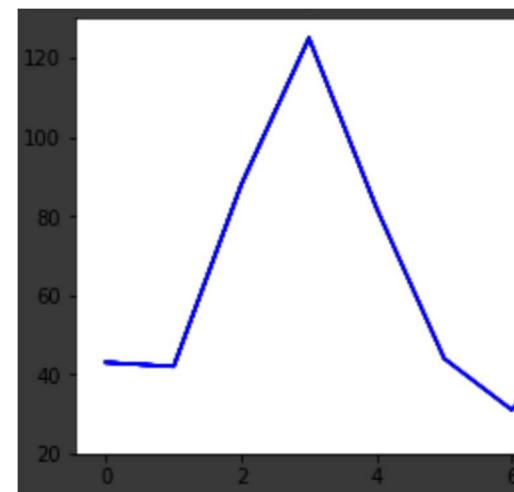
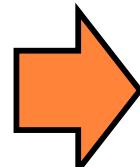
Deep Q-Networks (DQN)

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

Deep Q-Networks (DQN)

3. 코드 설명

- Training -> Agent

S	A	R	S'	d
....

Mini Batch

X	X'	θ	θ'
....

States

X	X'	θ	θ'
....

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'
	:	Ω

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'	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)
```

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

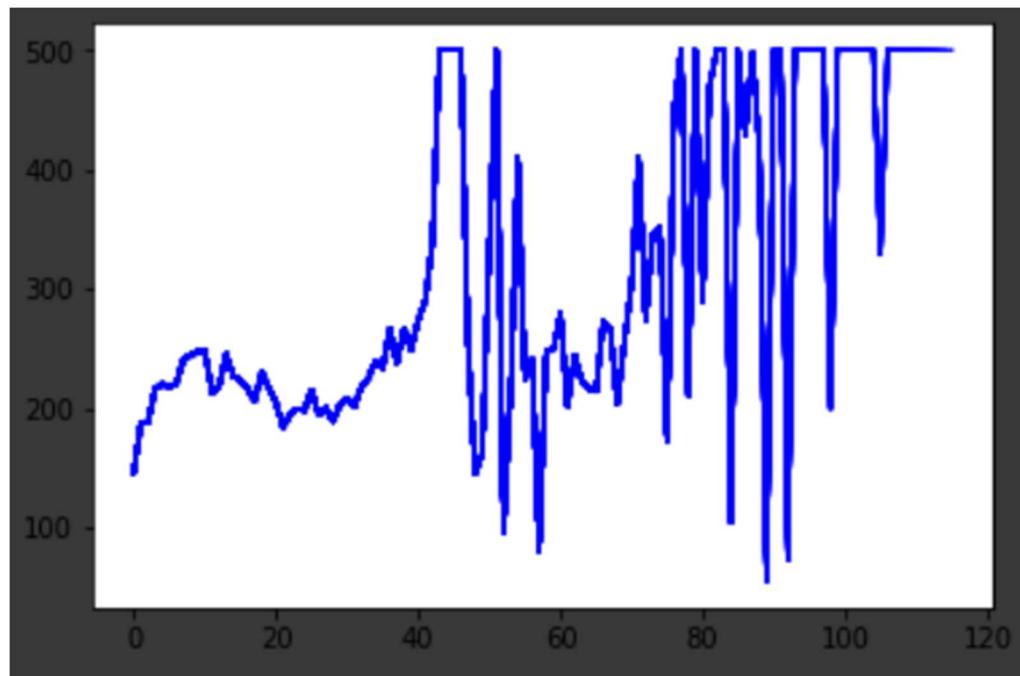
- Training

```
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



Outline

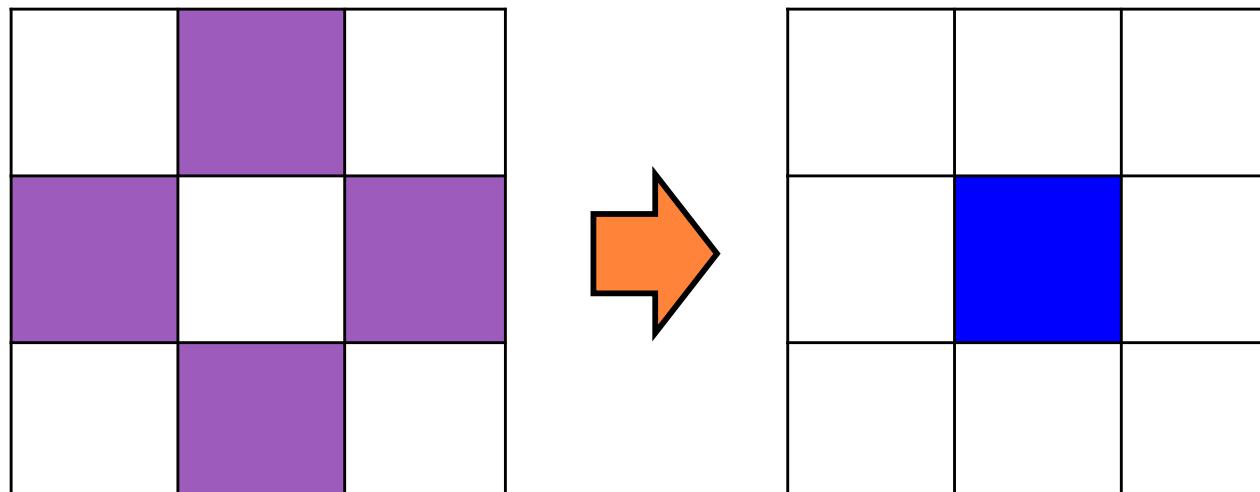
- Overview
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)
- **Advantage Actor-Critic (A2C)**
- Asynchronous Advantage Actor-Critic (A3C)
- Applications

Advantage Actor-Critic (A2C)

A2C = 정책 이터레이션 + 폴리시 그래디언트

1) 정책 이터레이션 (정책 평가 + 정책 발전)

- 정책 평가: 가치함수 업데이트



Iteration k

Iteration k+1

$$\forall_{s \in S}, V_\pi(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma v_k(s'))$$

Advantage Actor-Critic (A2C)

A2C = 정책 이터레이션 + 폴리시 그래디언트

1) 정책 이터레이션 (정책 평가 + 정책 발전)

- 정책 발전: 큐함수 선택

$$\forall_{s \in S}, V_\pi(s) = \sum_{a \in A} \pi(a|s) (R_s^a + \gamma v_k(s'))$$

$$q_\pi(s, a) = R_s^a + \gamma v_\pi(s')$$

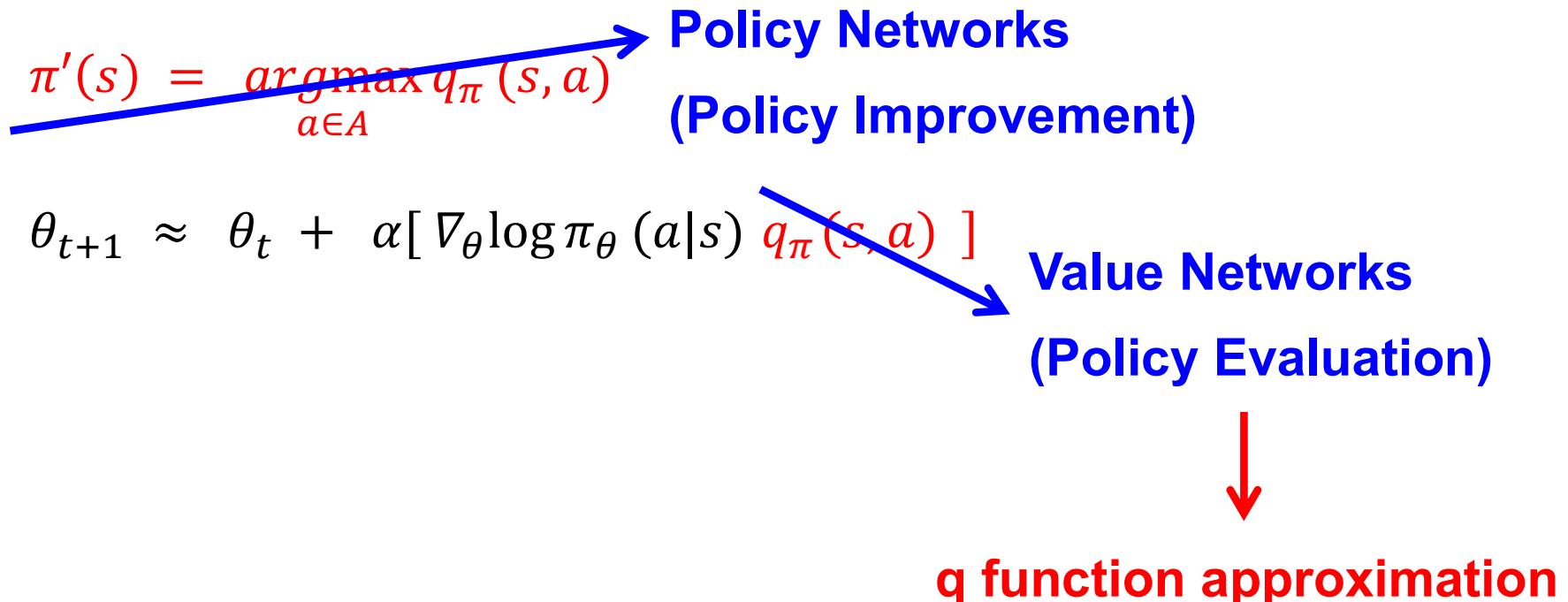
$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}} q_\pi(s, a)$$

Advantage Actor-Critic (A2C)

A2C = 정책 이터레이션 + 폴리시 그레디언트

2) 폴리시 그레디언트

- 정책 발전: 큐함수 선택
- 정책 평가: 가치함수 업데이트



Advantage Actor-Critic (A2C)

A2C = Policy Iteration + Policy Gradient

2) Policy Gradient

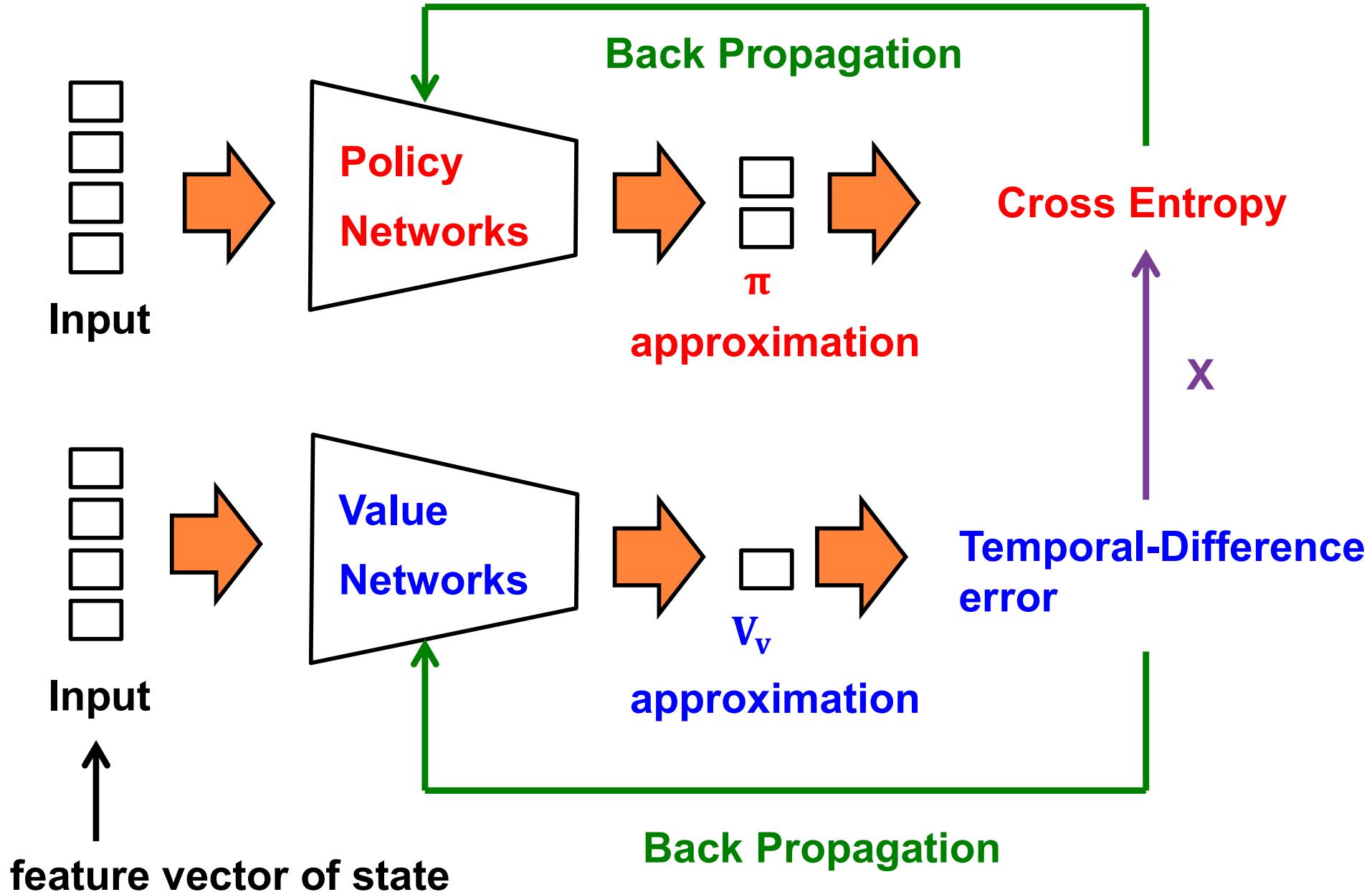
- 정책 발전: 큐함수 선택 (**Actor**)
- 정책 평가: 가치함수 업데이트 (**Critic**)

$$\delta_v = R_{t+1} + \gamma V_v(S_{t+1}) - V_v(s_t), \delta_v : \text{Adventage function}$$

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

- **Actor** : 어떤 행동 할지 선택
- **Critic** : 어떤 행동이 좋았는지 알려주고 조절하게 함

Advantage Actor-Critic (A2C)



Advantage Actor-Critic (A2C)

- Loss function

= Cross Entropy of Policy Network output * q function (Value Network output)

$$MSE = (Answer - prediction)^2 = (R_{t+1} + \gamma V_v(S_{t+1}) - V_v(S_t))^2$$

(for Value Networks)

Advantage Actor-Critic (A2C)

- 코드

<http://bitly.kr/mwAFCjjrAjV>

Outline

- Overview
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)
- Advantage Actor-Critic (A2C)
- **Asynchronous Advantage Actor-Critic (A3C)**
- Applications

Asynchronous Advantage Actor-Critic (A3C)

1. DQN의 단점?

- (1) 딥마인드의 DQN에서는 리플레이 메모리로 1,000,000 크기의 메모리를 사용함.
- (2) 따라서 메모리를 많이 차지하며 학습이 느린 특징을 가짐

Asynchronous Advantage Actor-Critic (A3C)

2. A3C

- (1) DQN처럼 많은 샘플을 모으고 샘플간의 연관성을 없애는게 아니라 애초에 에이전트를 여러 개 사용
- (2) 샘플을 모으는 각 에이전트는 액터러너(Actor-Learner)라고 함
- (3) 따라서 서로다른 액터러너가 모으는 샘플은 연관성이 떨어짐
- (4) 이 과정이 비동기의 형태로 이루어지기 때문에 Asynchronous라고 함
- (5) 각각의 액터러너가 모은 샘플들로 글로벌 신경망을 업데이트하면서 학습하는 과정을 거침

Outline

- Overview
- Basics
- Dynamic Programming
- Q-Learning
- Deep Reinforcement Learning
- Deep Q-Networks (DQN)
- Advantage Actor-Critic (A2C)
- Asynchronous Advantage Actor-Critic (A3C)
- Applications

Applications

Deep Reinforcement Learning

1. Wang *et al.*, “Influence-based Multi-Agent Exploration,” *ICLR*, 2020
2. Dabney *et al.*, “A Distributional Code for Value in Dopamine-based Reinforcement Learning,” *Nature*, 2020
3. Lebensold *et al.*, “Actor Critic with Differentially Private Critic,” *NeurIPS*, 2019
4. Jaderberg *et al.*, “Human-Level Performance in 3D Multiplayer Games with Population-based Reinforcement Learning,” *Science*, 2019
5. Derman *et al.*, “A Bayesian Approach to Robust Reinforcement Learning,” *UAI*, 2019
6. Barati and Chen, “An Actor-Critic-Attention Mechanism for Deep Reinforcement Learning in Multi-view Environments,” *IJCAI*, 2019
7. Liu *et al.*, “Deep Reinforcement Active Learning for Human-In-The-Loop Person Re-Identification,” *ICCV*, 2019
8. Wang *et al.*, “Language-driven Temporal Activity Localization: A Semantic Matching Reinforcement Learning Model,” *CVPR*, 2019
9. Dann *et al.*, “Policy Certificates: Towards Accountable Reinforcement Learning,” *ICML*, 2019
10. Zhou *et al.*, “Deep Model-based Reinforcement Learning via Estimated Uncertainty and Conservative Policy Optimization,” *AAAI*, 2020
11. Byravan *et al.*, “Imagined Value Gradients: Model-based Policy Optimization with Transferable Latent Dynamics Models,” *CoRL*, 2019
12. Hasselt *et al.*, “General Non-linear Bellman Equations,” *RLDM*, 2019
13. Zhang *et al.*, “Feature Aggregation with Reinforcement Learning for Video-based Person Re-Identification,” *IEEE-TNNLS*, 2019
14. Zhang *et al.*, “Reconstruct and Represent Video Contents for Captioning via Reinforcement Learning,” *IEEE-TPAMI*, 2019
15. Oh and Iyengar, “Sequential Anomaly Detection using Inverse Reinforcement Learning,” *KDD*, 2019

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>

Appendix: Genetic Algorithm(GA) vs DP

기본적인 컨셉은 비슷

- Genetic Algorithm

<https://untitledtblog.tistory.com/110>

Appendix: On-Policy vs Off-Policy

어떤게 무조건 좋다? → 그런건 없다.

- On-Policy

행동하는 정책과 목표 정책이 동일. 하나의 정책으로 행동하고 학습함

- Off-Policy

현재 행동하는 정책과는 독립적으로 학습. 즉, 행동하는 정책과 학습하는 정책을 따로 분리하여 에이전트는 행동하는 정책으로 지속적인 탐험을 하고 학습은 따로 목표 정책을 둔다.