커리큘럼 : 전체 흐름

Part 1　강화학습의 소개 및 개요

↓

Part 2　강화학습의 기초

↓

Part 3　딥러닝과 강화학습

1. 강화학습 소개 및 개요

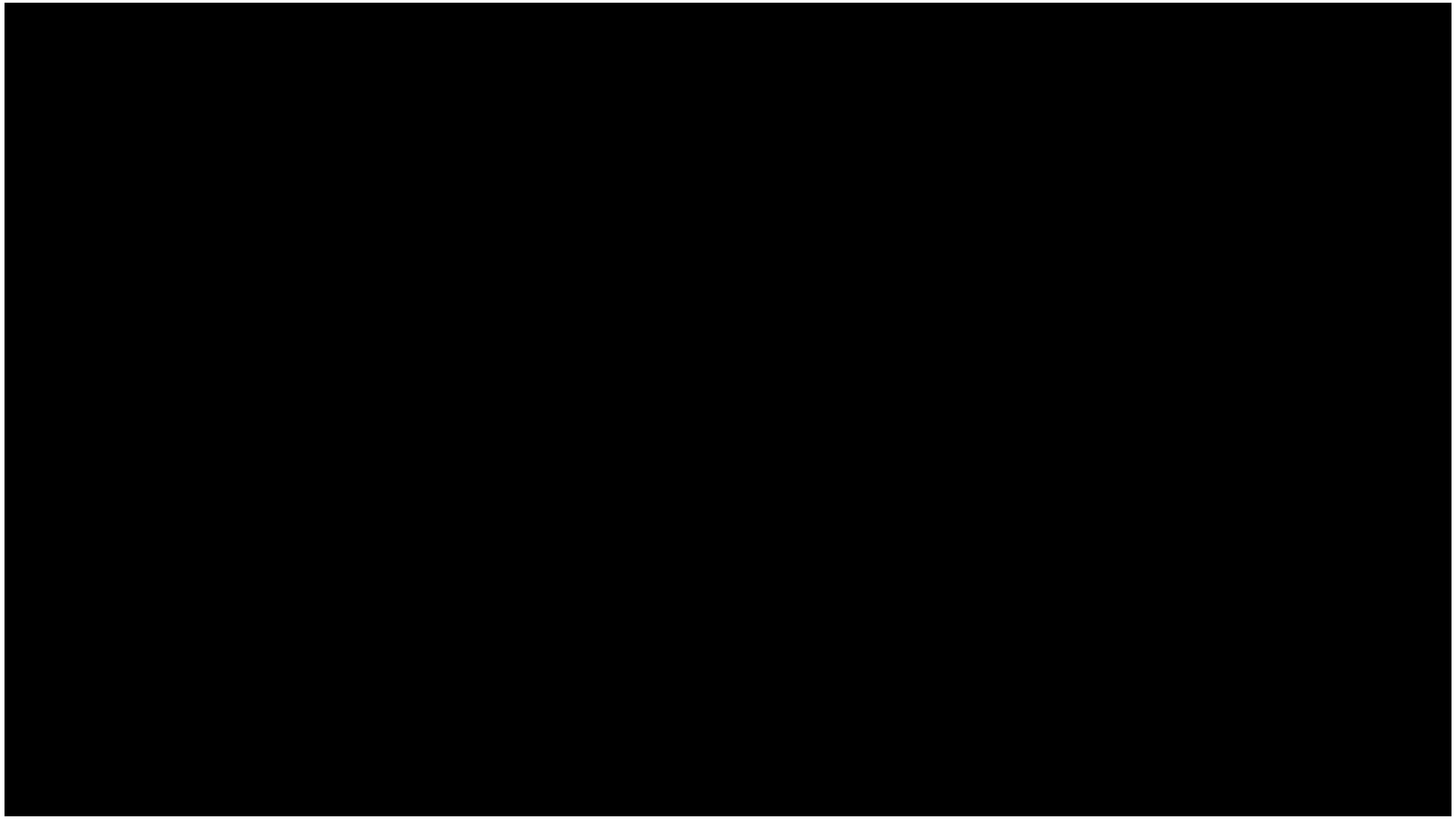
최근 AI의 활용 사례

1. AlphaGo
2. Robot arm manipulation
3. Walking & Running

→ 공통 기술 : Deep Reinforcement Learning

최신 AI의 활용 사례 : AlphaGo

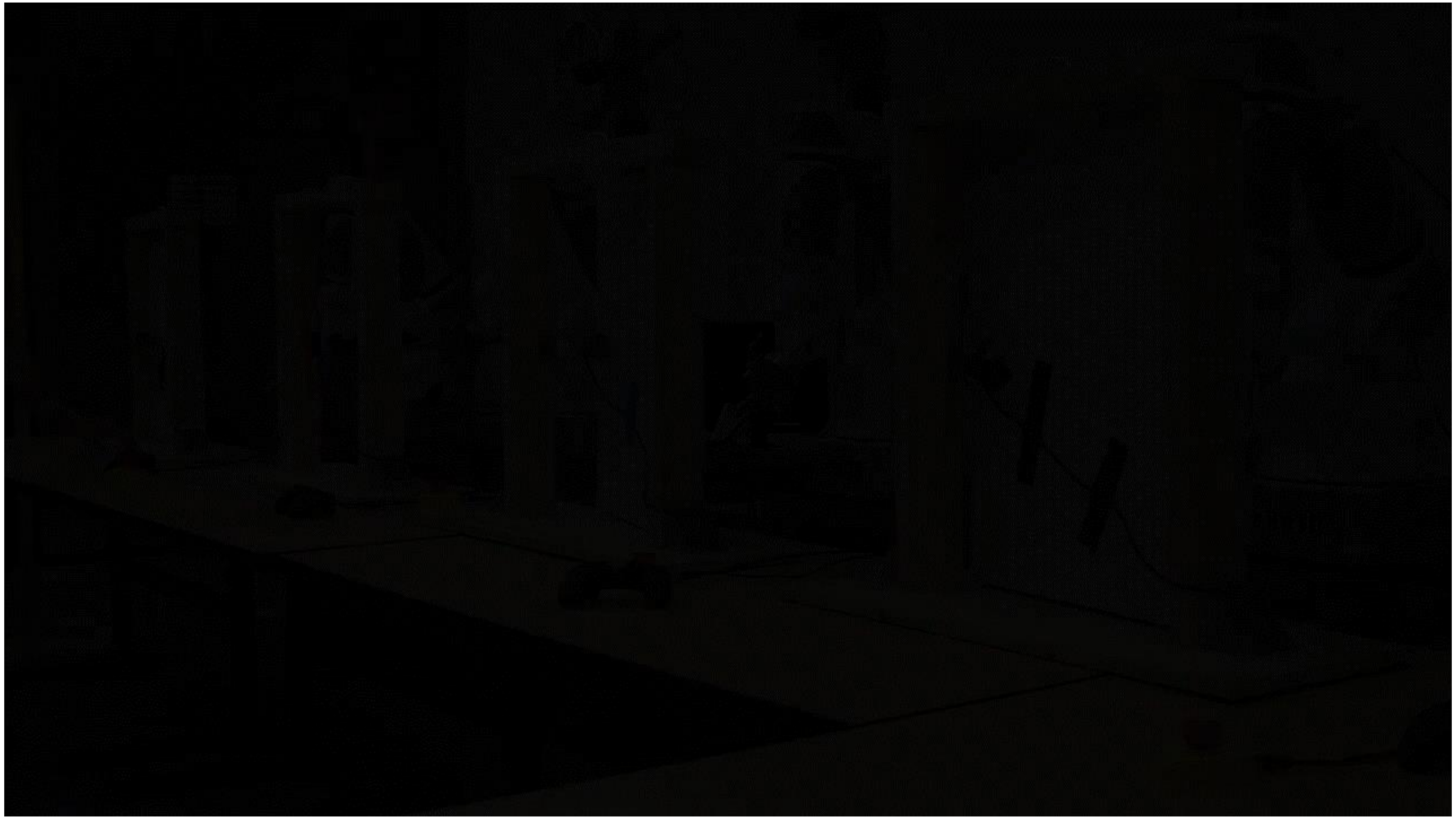
사용된 알고리즘 : Policy Gradient with Monte-Carlo Tree Search



https://www.youtube.com/watch?v=qa7p0GysSlw

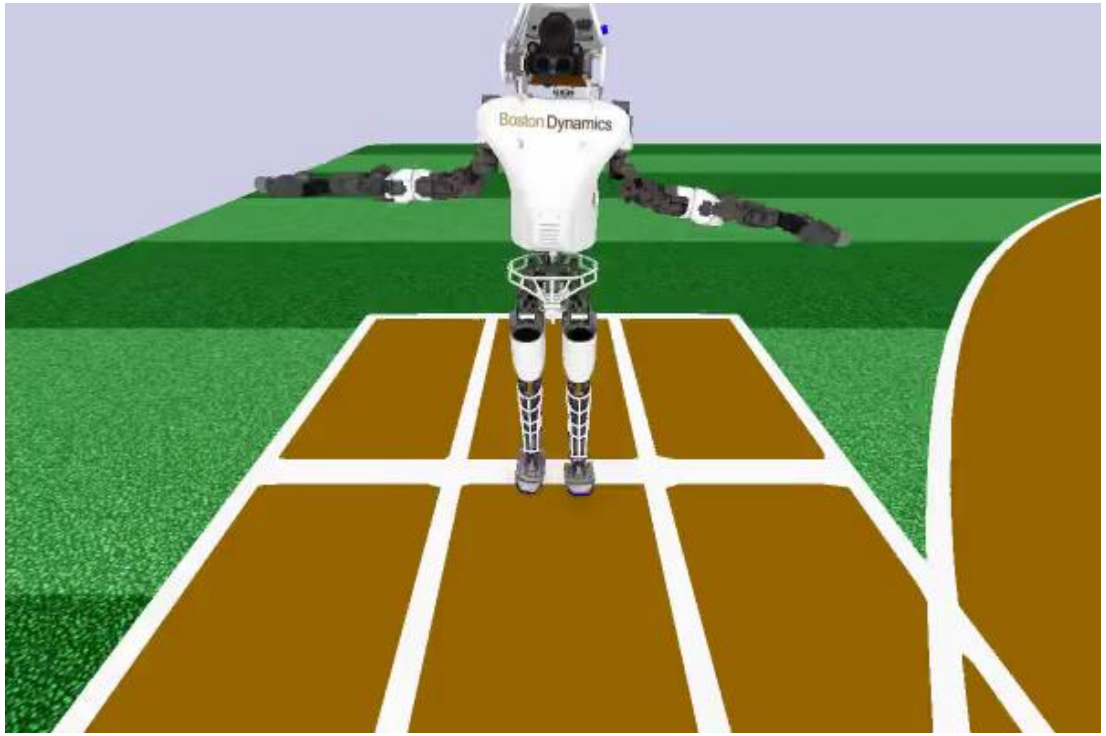
최신 AI의 활용 사례 : Robotic Manipulation

Collective Robot Reinforcement Learning with Distributed Asynchronous Guided Policy Search



https://www.youtube.com/watch?v=ZBFwe1gF0FU

최신 AI의 활용 사례 : Walking & Running



https://blog.openai.com/openai-baselines-ppo/

https://www.youtube.com/watch?v=gn4nRCC9TwQ

강화학습이란

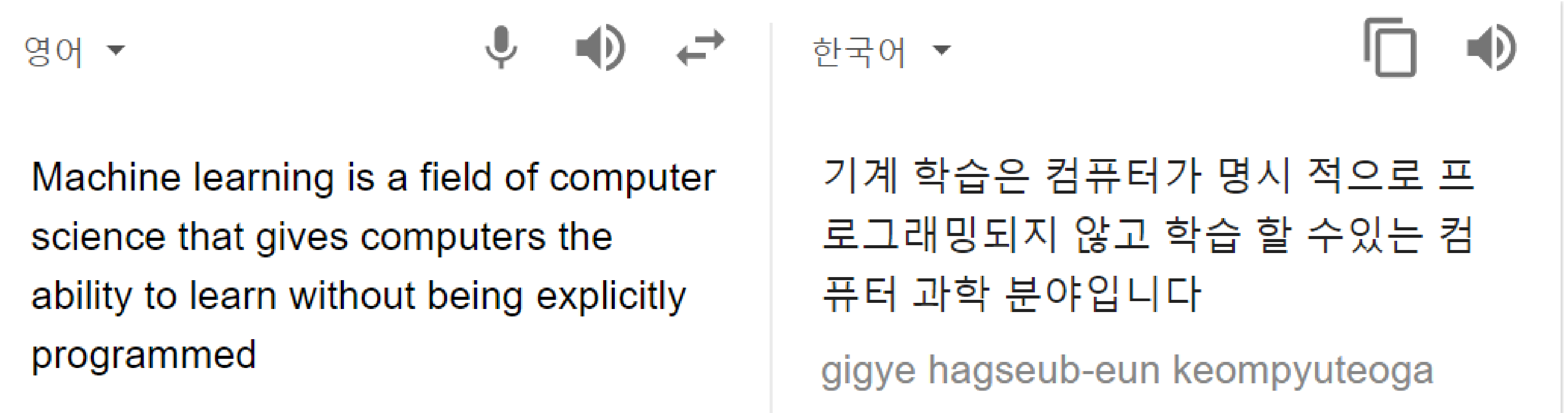
강화학습(Reinforcement Learning)을 이해하는 순서

1. Reinforcement Learning = Reinforcement + Machine Learning
2. Reinforcement 은 무엇인가?
3. Machine Learning 은 무엇인가?

Machine Learning 은 무엇인가

- 위키피디아 정의

Machine learning is a field of computer science that gives computers the ability to learn without being explicitly programmed



Machine Learning 은 무엇인가

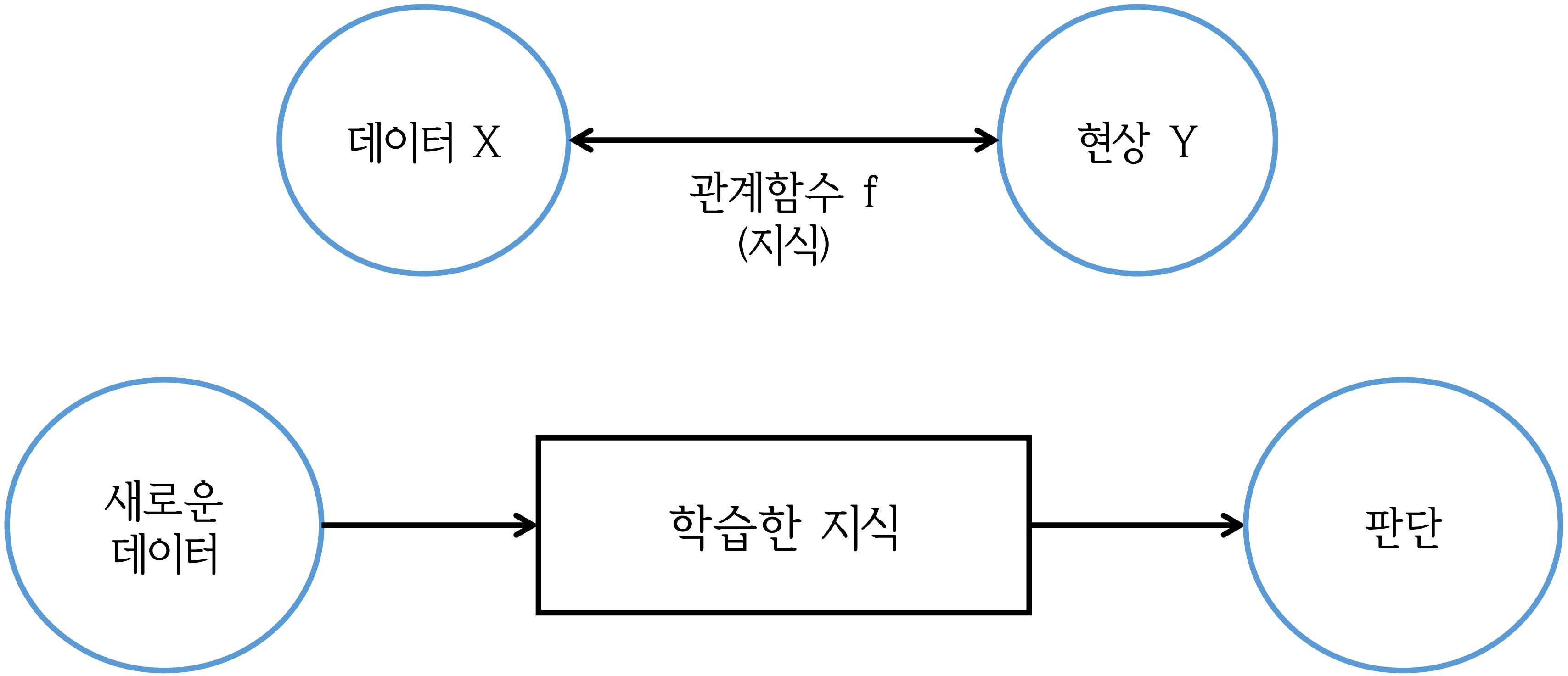
Explicit Programming		Machine Learning
If 배가 고프면, then 밥을 먹어라	→	데이터 기반 → 예측 + 학습
간단한 문제		복잡한 문제

Ex) 스팸필터, 추천 시스템,

날씨와 교통상황 사이의 상관관계

Machine Learning 은 무엇인가

데이터로부터 유용한 **지식**을 추출해 새로운 데이터에 대한 **판단**에 적용



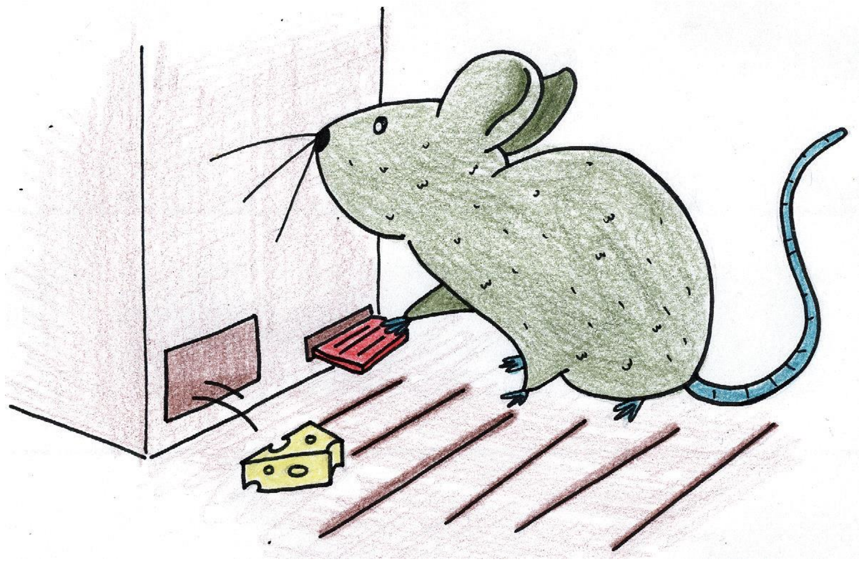
Machine Learning 은 무엇인가

Machine Learning의 종류

1. Supervised Learning : 정답이 있는 데이터 학습
2. Unsupervised Learning : 데이터 자체의 특성 학습
3. Reinforcement Learning : 보상으로부터 학습

Reinforcement 는 무엇인가

1. 행동주의와 Skinner → "눈으로 관찰가능한 행동을 연구"
2. Skinner의 문제상자 : 레버를 누르면 먹이가 나오는 상자 안에 굶긴 쥐를 넣고 실험



Reinforcement 는 무엇인가

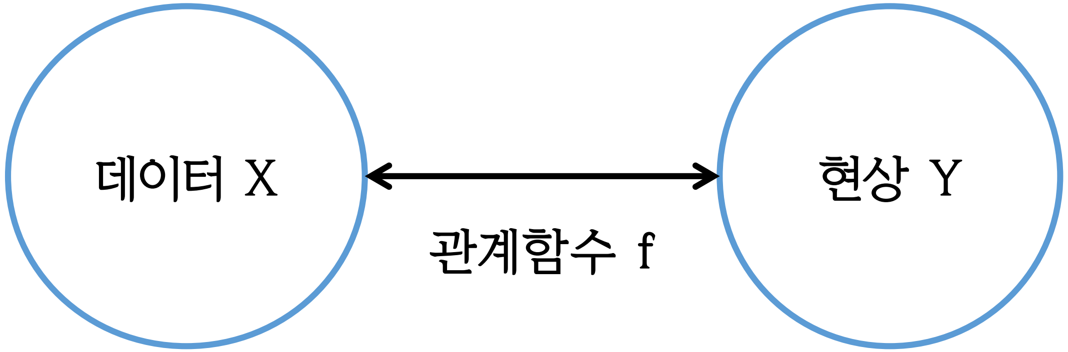
1. 굶긴 쥐를 상자에 넣는다
2. 쥐는 돌아다니다가 우연히 상자 안에 있는 지렛대를 누르게 된다
3. 지렛대를 누르자 먹이가 나온다
4. 지렛대를 누르는 행동과 먹이와의 상관관계를 모르는 쥐는 다시 돌아다닌다
5. 그러다가 우연히 쥐가 다시 지렛대를 누르면 쥐는 이제 먹이와 지렛대 사이의 관계를 알게 되고 점점 지렛대를 자주 누르게 된다
6. 이 과정을 반복하면서 쥐는 지렛대를 누르면 먹이를 먹을 수 있다는 것을 학습한다[1]

https://namu.wiki/w/%ED%96%89%EB%8F%99%EC%A3%BC%EC%9D%98

Reinforcement : 배우지 않았지만 직접 시도하면서 행동과 그 결과로 나타나는 보상 사이의 상관관계를 학습하는 것 → 보상을 많이 받는 행동의 확률을 높이기

강화학습은 무엇인가

1. Reinforcement Learning = Reinforcement + Machine Learning

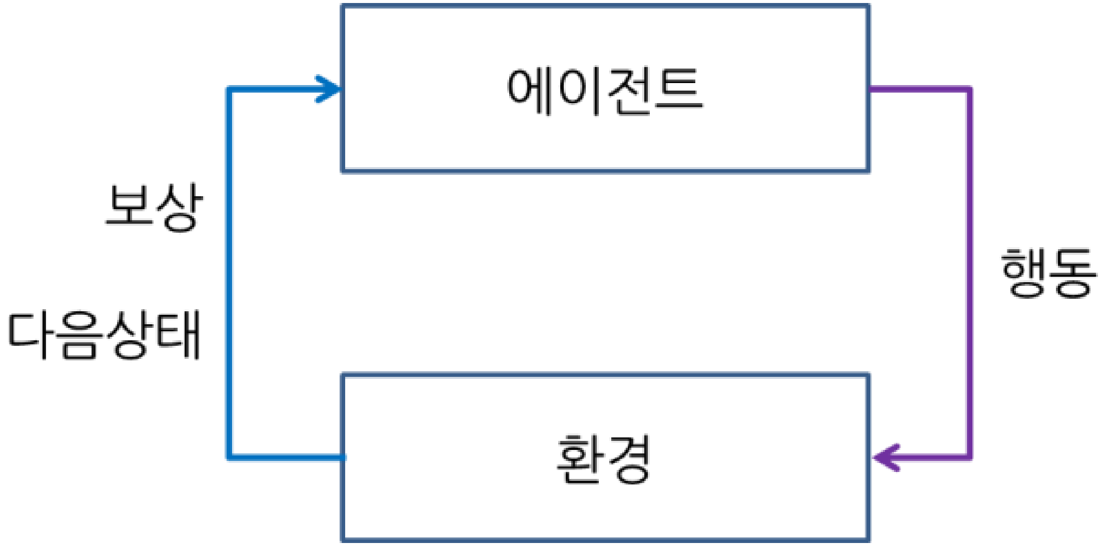


2. 데이터 X : 어떤 상황에서 어떤 행동을 했는지, 현상 Y: 보상을 얼마나 받았는지

→ 어떤 행동을 해야 보상을 많이 받는지를 데이터로부터 학습

강화학습은 무엇인가

1. 에이전트와 환경의 상호작용 → 데이터 생성 (미리 모아 놓을 수 없다)
2. 특정 상태에서 특정 행동을 선택 → 보상 → 학습



강화학습 개요

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. 문제에 대한 수학적 정의 : Markov Decision Process
3. MDP를 계산으로 푸는 방법 : Dynamic Programming
4. MDP를 학습으로 푸는 방법 : Reinforcement Learning
5. 상태공간이 크고 차원이 높을 때 쓰는 방법 : Function Approximation
6. 바둑과 같은 복잡하고 어려운 문제를 푸는 방법 : Deep Reinforcement Learning

2. 강화학습의 기초

2-1. Sequential Decision Problem

강화학습 개요

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. 문제에 대한 수학적 정의 : Markov Decision Process
3. MDP를 계산으로 푸는 방법 : Dynamic Programming
4. MDP를 학습으로 푸는 방법 : Reinforcement Learning
5. 상태공간이 크고 차원이 높을 때 쓰는 방법 : Function Approximation
6. 바둑과 같은 복잡하고 어려운 문제를 푸는 방법 : Deep Reinforcement Learning

강화학습 접근 방식

1. 문제 접근 방식

- 문제 자체에 대한 이해
- 그 문제에 적용되었던 초기 방식들
- 초기 방식의 문제와 해결하기 위한 아이디어
- 아이디어를 구체화한 방법

강화학습 접근 방식

2. 강화학습 문제 접근 방식

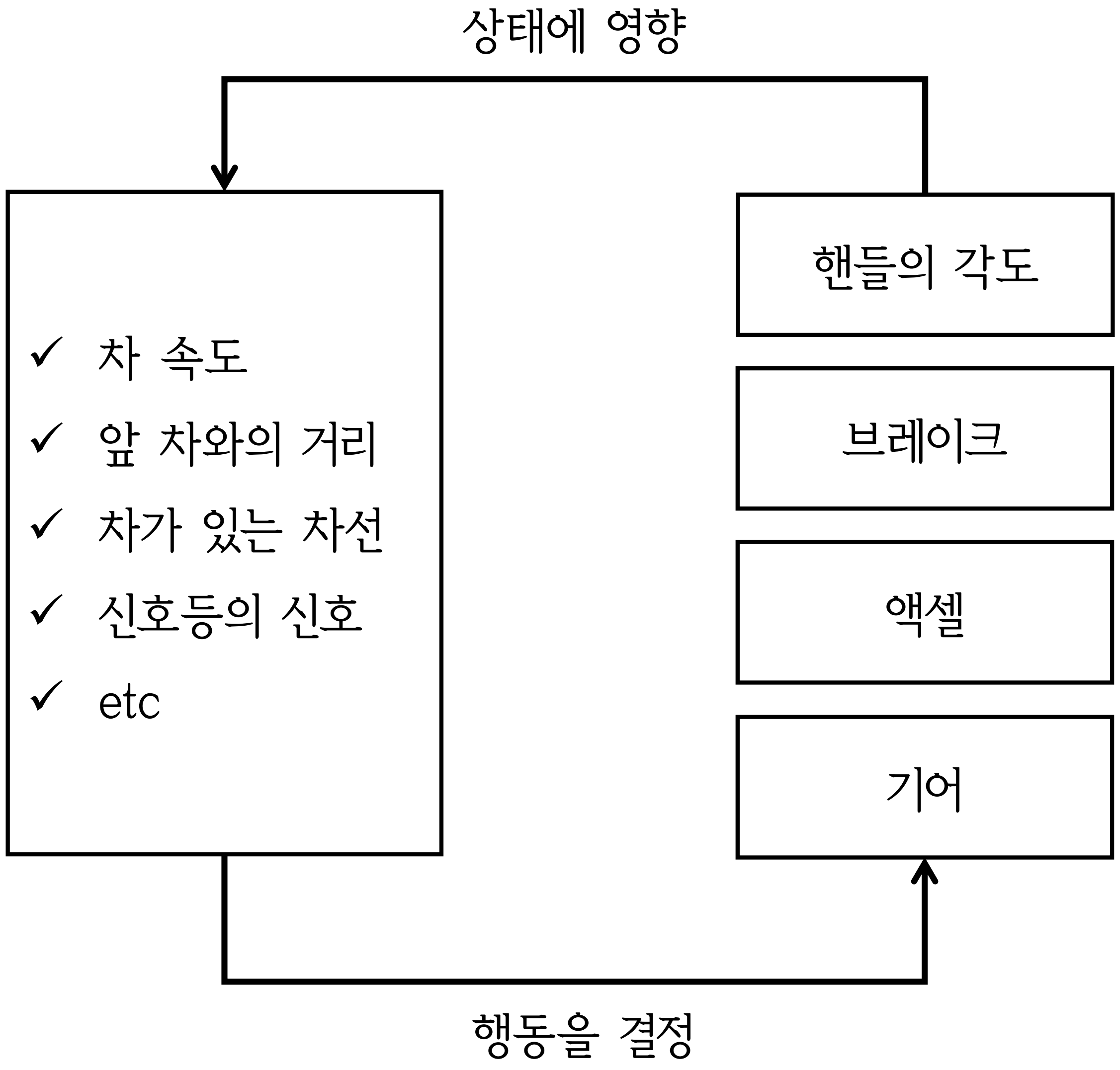
- Sequential Decision Problem과 MDP 이해
- MDP 문제를 풀기 위한 DP(Dynamic Programming)
- DP의 문제와 이를 해결하기 위한 아이디어 (Planning → Learning)
- 아이디어의 알고리즘화 : Q-Learning

Sequential Decision Problem

- 여러 번의 연속적 선택을 하는 문제 : Sequential Decision Problem
  - 집에서 직장까지 차를 운전하는 경우

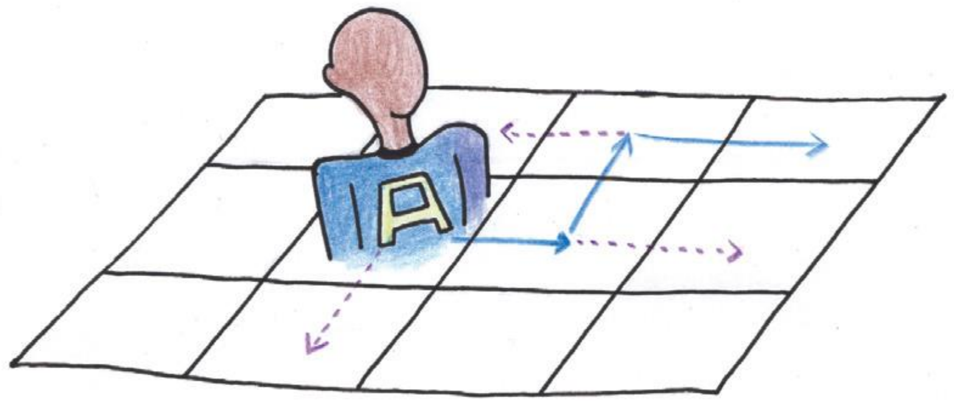
http://www.chevrolet.co.in/vehicles/cars.html

Sequential Decision Problem



Sequential Decision Problem

- 에이전트 : 상태를 관찰, 행동을 선택, 목표지향, “Decision Maker!”



an autonomous, goal-directed entity which observes and acts upon an environment – 위키피디아

환경을 관찰하고 행동하는 자율적이고 목표 지향적인 주체-구글번역기

Sequential Decision Problem

- 환경 : 에이전트를 제외한 나머지

판단하는 아이라는 주체를 빼고 길과 자전거와 아이의 몸 또한 환경이 된다

Markov Decision Process

1. Sequential Decision Problem을 수학적으로 정의
2. MDP(Markov Decision Process)의 목표는 reward를 최대화
3. Markov Process → MRP(Markov Reward Process) → MDP(Markov Decision Process)

Andrei Andreyevich Markov

2-2. Markov Decision Process

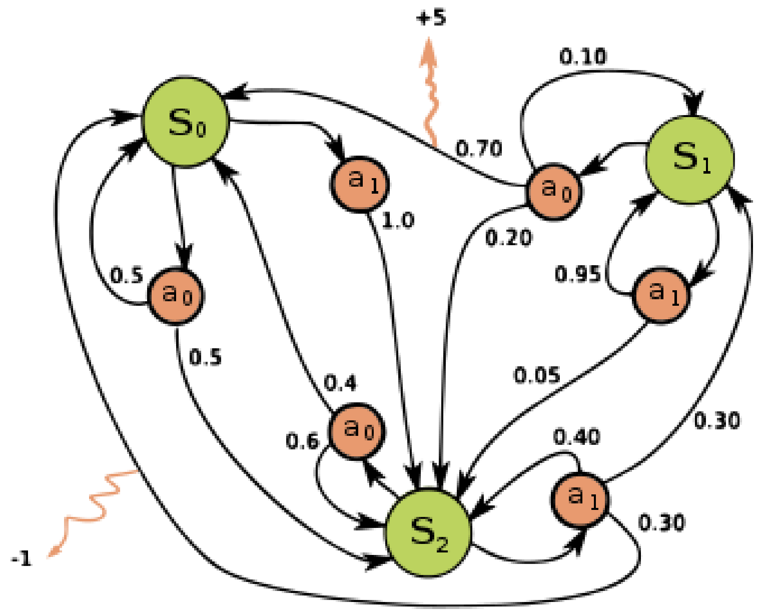
강화학습 개요

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. 문제에 대한 수학적 정의 : Markov Decision Process
3. MDP를 계산으로 푸는 방법 : Dynamic Programming
4. MDP를 학습으로 푸는 방법 : Reinforcement Learning
5. 상태공간이 크고 차원이 높을 때 쓰는 방법 : Function Approximation
6. 바둑과 같은 복잡하고 어려운 문제를 푸는 방법 : Deep Reinforcement Learning

Markov Decision Process

- 시간에 따라 변하는 “상태”가 있으며 상태 공간 안에서 움직이는 “에이전트”가 있다
  - 에이전트는 행동을 선택할 수 있다 → 확률적
  - 에이전트의 행동에 따라 다음 상태와 보상이 결정된다 → 확률적

→ 확률적 모델링 : Markov Decision Process



https://en.wikipedia.org/wiki/Markov_decision_process

Markov Decision Process

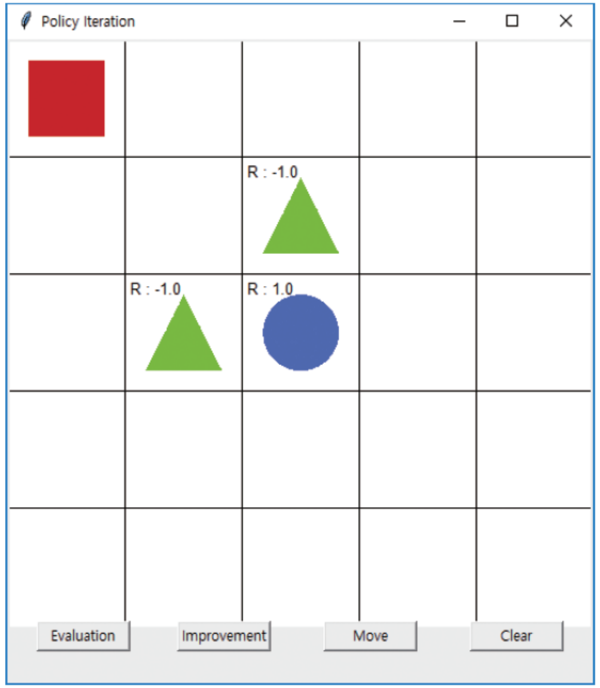
1. MDP = $\{S,\ A,\ R,\ P_{ss'}^{a},\ \gamma\}$ 로 정의되는 tuple

2. MDP의 구성요소

- S : 상태(state)
- A : 행동(action)
- R : 보상(reward)
- $P_{ss'}^{a}$: 상태변환확률(state transition probability)
- γ : 할인율(discount factor)

Grid World 예제

1. 격자를 기반으로 한 예제 : 5 X 5 = 25개의 격자를 가짐
2. 고전 강화학습의 가장 기본적인 예제 : 에이전트가 학습하는 과정을 눈으로 보기 쉬움
3. 목표 : 세모를 피해서 파란색 동그라미로 가기



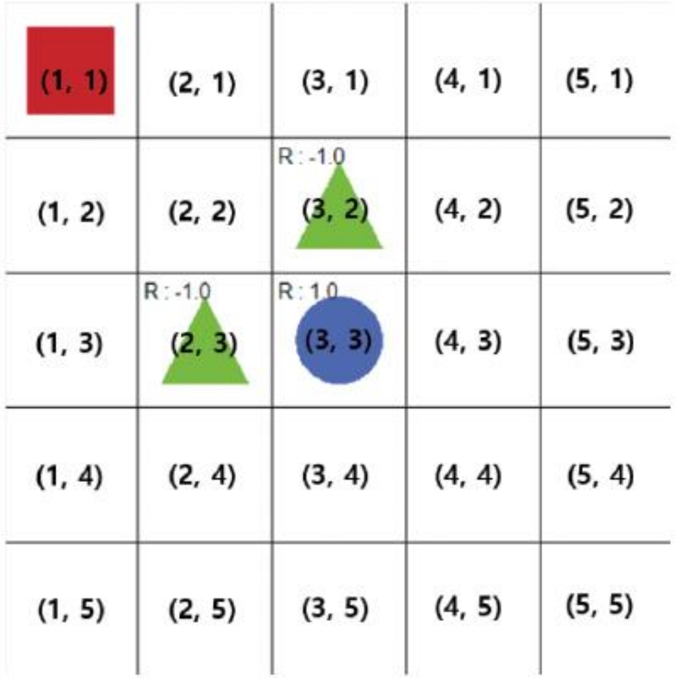
MDP 1 : 상태

- 상태 : 현재 상황을 나타내는 정보 (observation과의 관계)

에이전트가 탁구를 치려면 탁구공의 위치, 속도, 가속도와 같은 정보가 필요

MDP 1 : 상태

- 상태 : 에이전트가 관찰 가능한 상태의 집합
- 그리드월드의 상태 : $\boldsymbol{S} = \{(1,1),(2,1),(1,2), \cdots, (5,5)\}$



MDP 1 : 상태

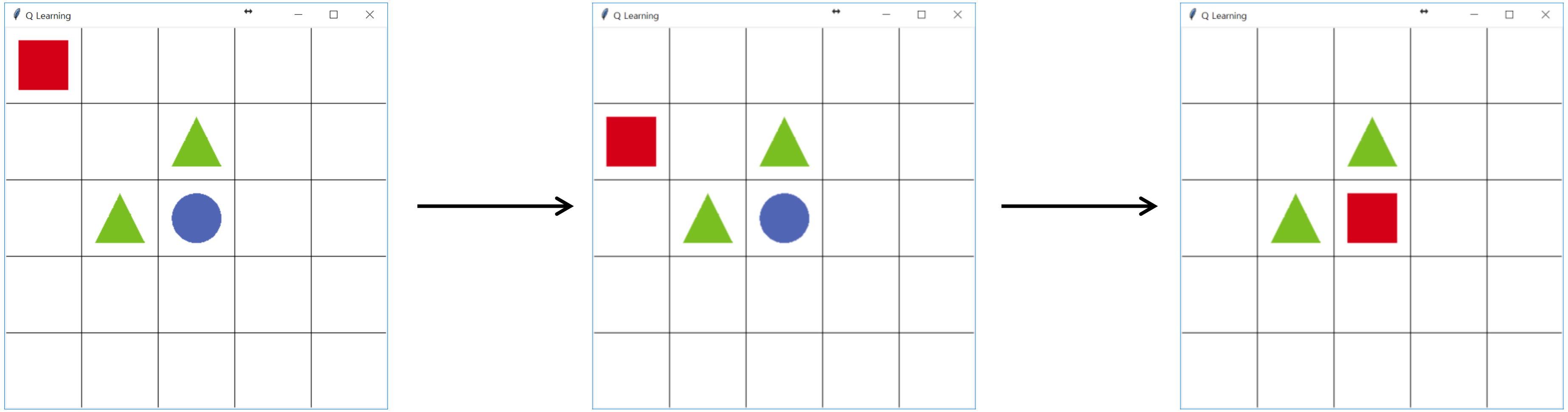
1. 에이전트는 시간에 따라 환경을 탐험 → 상태도 시간에 따라 변한다

2. 시간 t일 때 상태 : $S_t = s \quad or \quad S_t = (1,3)$
   - 확률변수(random variable)은 대문자, 특정 상태는 소문자

3. Episode : 처음 상태부터 마지막 상태까지

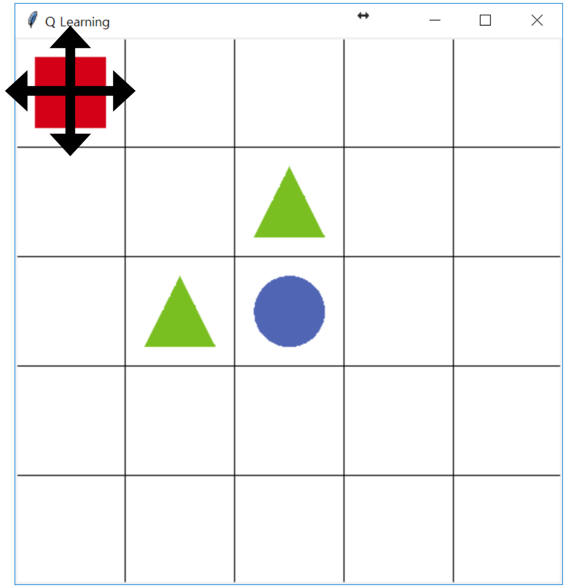
$$\tau = s_0, s_1, s_2, \cdots, s_{T-1}, s_T$$

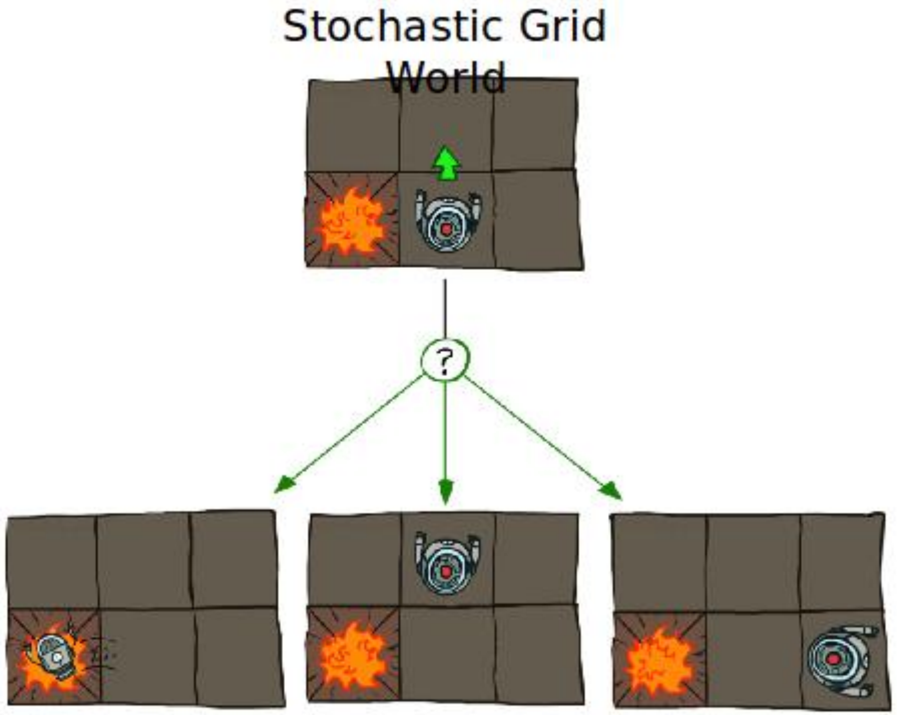
s_0 s_t s_T



MDP 2 : 행동

- 에이전트가 할 수 있는 행동의 집합 : $\boldsymbol{A} = \{위, 아래, 좌, 우\}$
- 시간 t에 취한 행동 $A_t = a$
- 만약 $A_t =$ 우 라면 항상 (3, 1)에서 (4, 1)로 갈까?
  - 상태 변환 확률에 따라 다르다

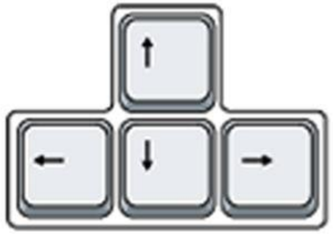


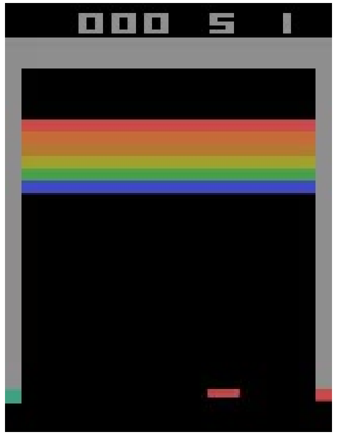


MDP 2 : 행동

- 행동 → (1) discrete action (2) continuous action

discrete action





continuous action

MDP 3 : 보상

- 에이전트가 한 행동에 대한 환경의 피드백 : 보상(+ or −)
- 시간이 t이고 상태 $S_t = s$ 에서 $A_t = a$를 선택했을 때 받는 보상

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

- 보상은 R_s^a으로 표현되거나 $R_{ss'}^a$으로 표현된다
- 보상은 현재 시간 t가 아닌 $t + 1$에 환경으로부터 받는다
- 같은 상태 s에서 같은 행동 a를 했더라도 그때 그때마다 보상이 다를 수 있음

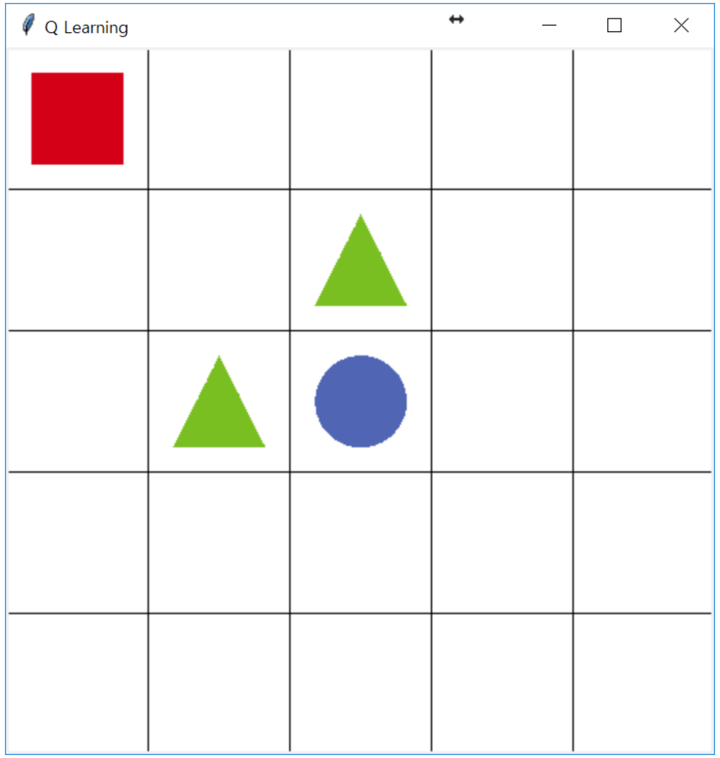
  → 기댓값(expectation)으로 표현

MDP 3 : 보상

1. 보상은 에이전트의 목표에 대한 정보를 담고 있어야 함

2. 그리드월드의 보상 : 초록색 세모 (-1), 파란색 동그라미 (+1)

→ 초록색 세모를 피해 파란색 동그라미로 가라!



MDP 3 : 보상

- 기댓값이란 : 확률을 포함한 평균
- 주사위를 던졌을 때 나올 숫자에 대한 기댓값은?

$$기댓값 = 1 \times \frac{1}{6} + 2 \times \frac{1}{6} + 3 \times \frac{1}{6} + 4 \times \frac{1}{6} + 5 \times \frac{1}{6} + 6 \times \frac{1}{6} = \frac{21}{6}$$

MDP 4 : 상태변환확률

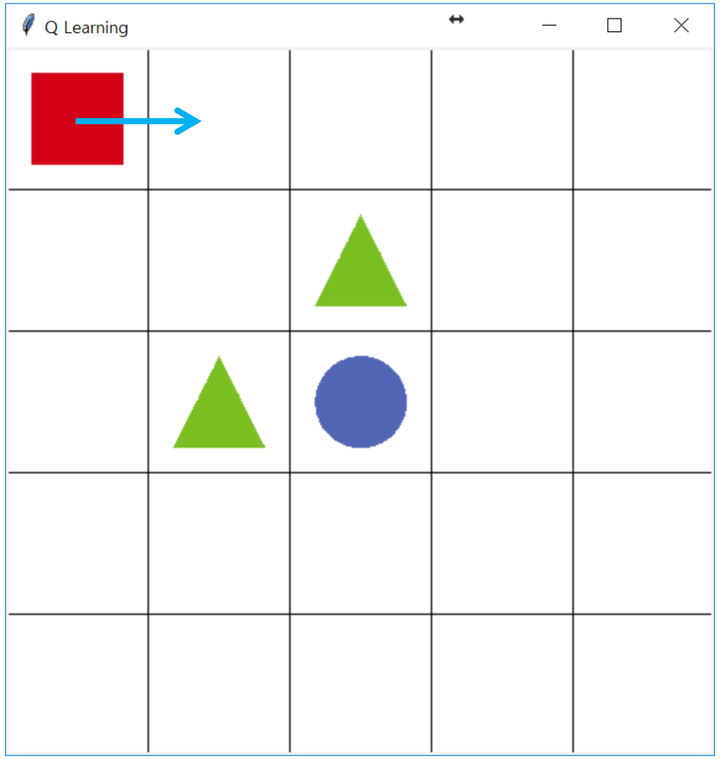
- 상태변환확률 : 상태 s에서 행동 a를 했을 때 상태 s'으로 갈 확률

$$P_{ss'}^{a} = \boldsymbol{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- model or dynamics of environment
- 상태변환확률을 안다면 : model-based
  - Dynamic Programming
- 상태변환확률을 모른다면 : model-free
  - Reinforcement Learning
- 상태변환확률을 학습한다면 : model-based RL
  - Dyna-Q

MDP 4 : 상태변환확률

$$P_{ss'}^{a} = \boldsymbol{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

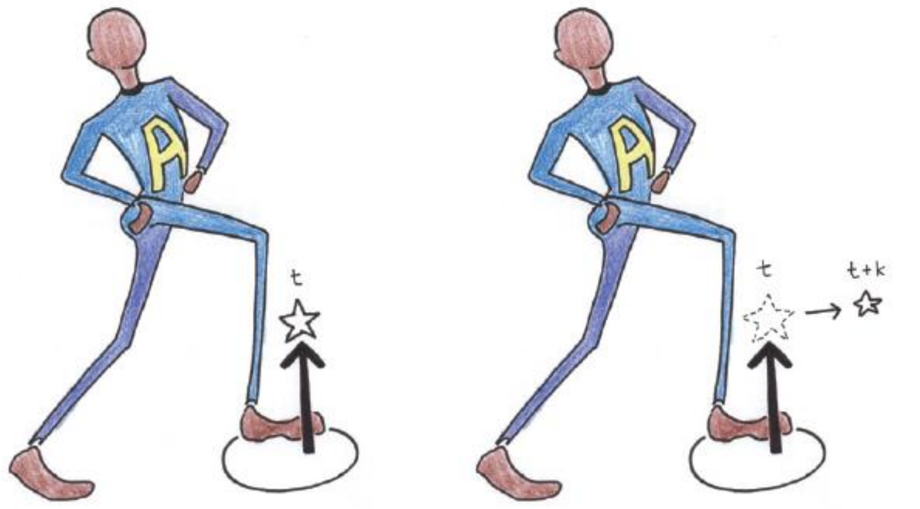


상태 (1, 1)에서 행동 “우”를 했을 경우

1. 상태 (2, 1)에 갈 확률은 0.8
2. 상태 (1, 2)에 갈 확률은 0.2

MDP 5 : 할인율

- 할인율 : 미래에 받은 보상을 현재의 시점에서 고려할 때 할인하는 비율
- 만약 복권에 당첨되었다면 당첨금 1억원을 당장 받을지 10년 뒤에 받을지?
  - 가까운 보상이 미래의 보상보다 더 가치가 있다 → 할인
- 보상에서 시간의 개념을 포함하는 방법



MDP 5 : 할인율

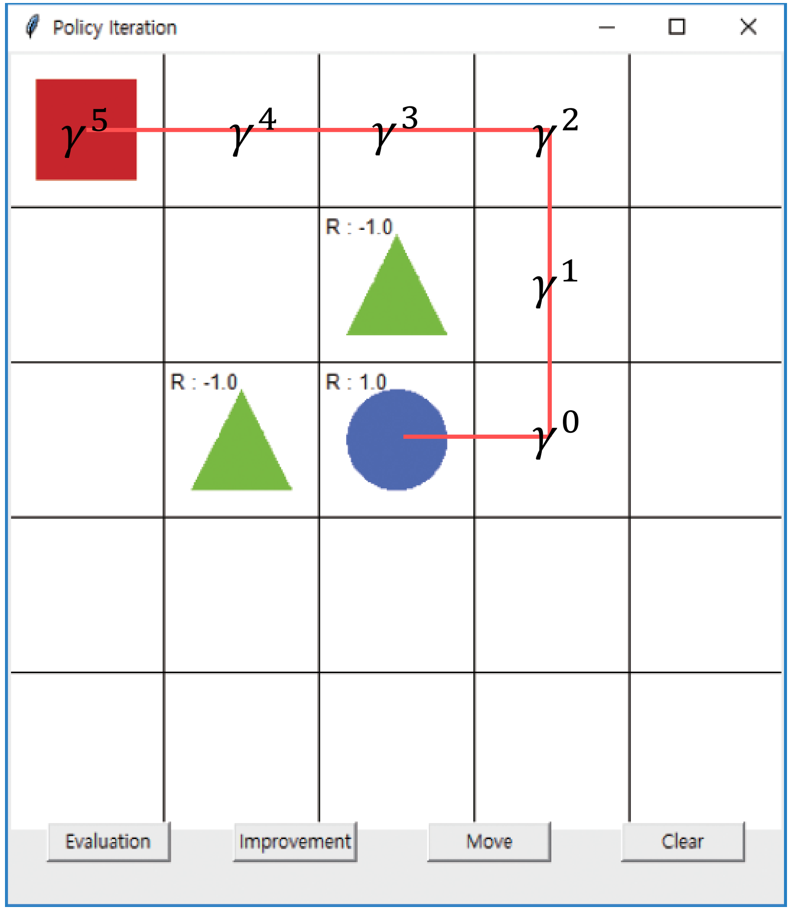
- 할인율은 0에서 1 사이의 값

$$\gamma \in [0, 1]$$

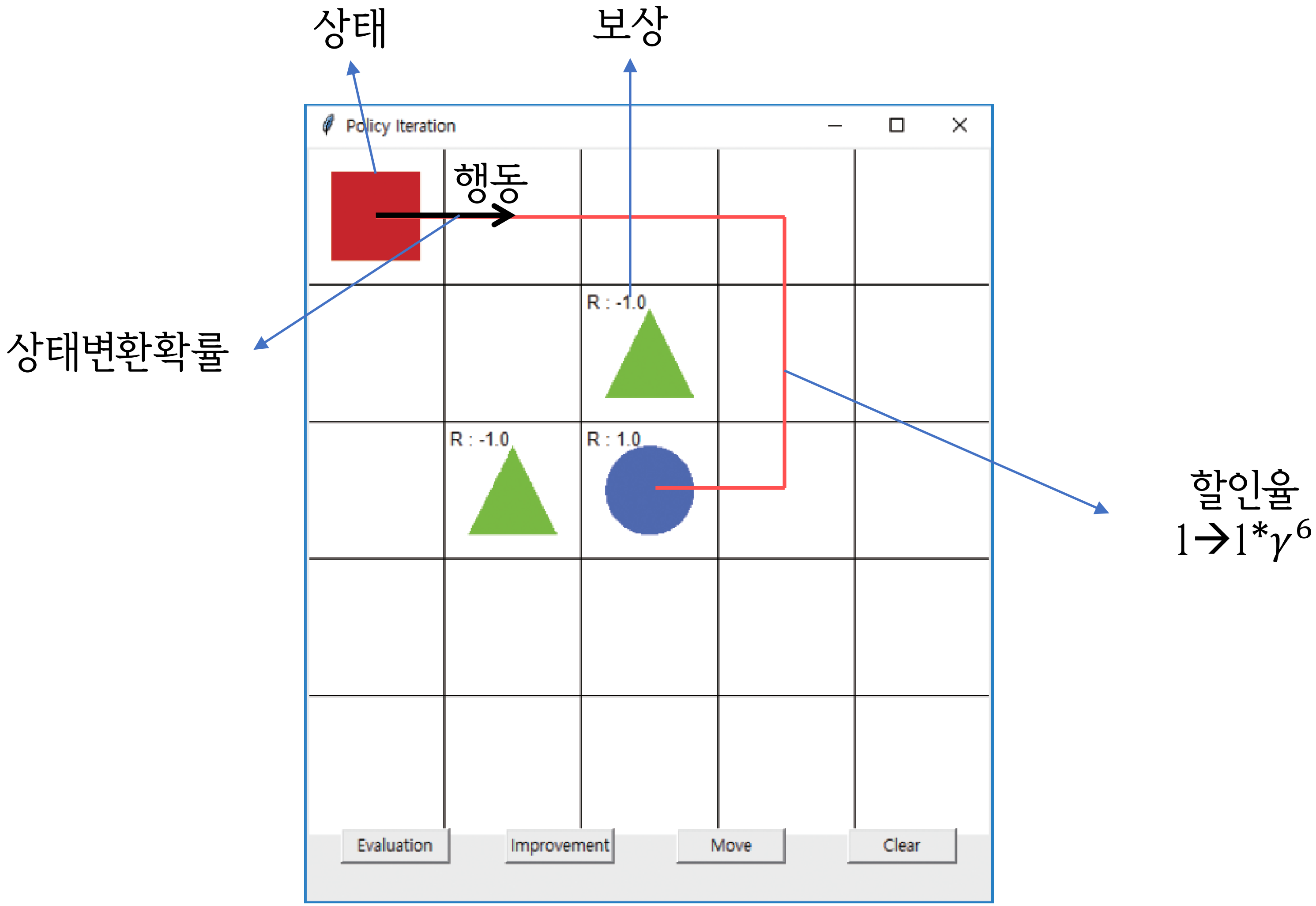
- 현재의 시간 t로부터 k만큼 지난 후 받은 보상의 현재 가치

$$\gamma^{k-1} R_{t+k}$$

- 할인율을 통해 보상을 얻는 최적의 경로를 찾을 수 있다



정리



그리드월드 문제에서의 MDP

정책

1. 에이전트는 각 상태마다 행동을 선택

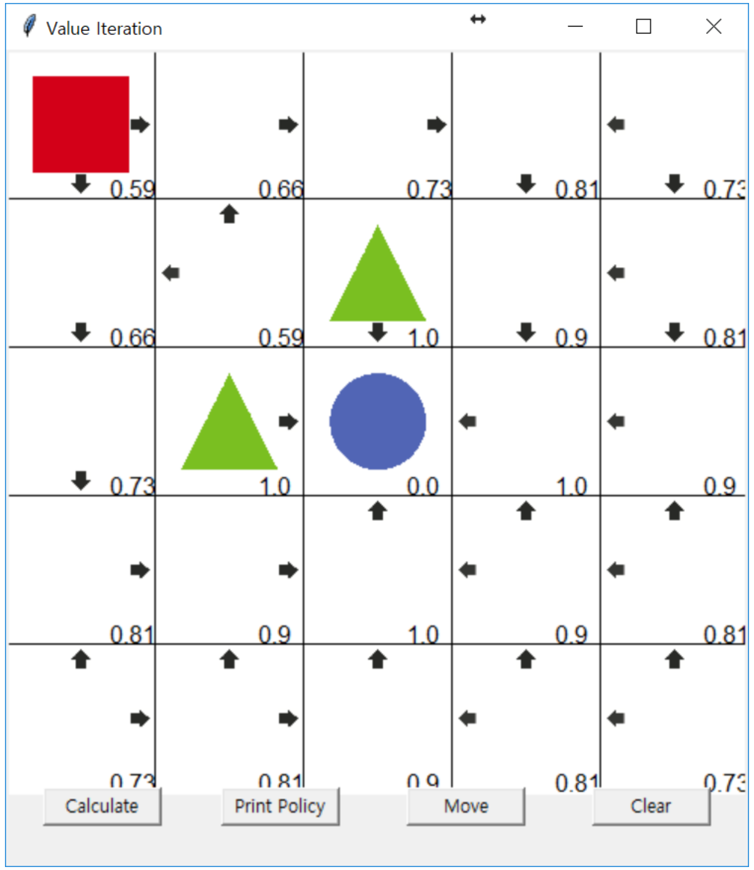
2. 각 상태에서 어떻게 행동할지에 대한 정보 : 정책(Policy)

   - 상태 s에서 행동 a를 선택할 확률

$$\pi(a|s)$$

3. 두 가지 형태의 정책

   - 행동 = 정책(상태) → 명시적(explicit) 정책
   - 행동 = 선택(가치함수(상태)) → 내재적(implicit) 정책



정리

1. 강화학습이 풀고자 하는 문제
    - Sequential Decision Problem
2. Sequential Decision Problem의 수학적 정의
    - MDP
3. MDP의 구성요소
    - 상태, 행동, 보상, 상태변환확률, 할인율
4. 각 상태에서 에이전트가 행동을 선택할 확률
    - 정책

2-3. Bellman Equation

MDP 문제 풀이 방법

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. MDP : Sequential Decision Problem의 수학적 정의
3. MDP의 목표는 최대의 보상을 받는 것

   → 매 타임스텝마다 행동 선택의 기준 : 보상

   → 더 큰 보상을 얻을 수 있는 행동을 선택

4. MDP 문제 풀이 방법
   1) Dynamic Programming : 환경에 대한 모든 정보를 알고 가장 좋은 정책을 "계산"
   2) Reinforcement Learning : 환경과의 상호작용을 통해 가장 좋은 정책을 "학습"

MDP 에이전트의 행동 선택

1. 에이전트와 환경의 상호작용(Value-based)

   (1) 에이전트가 상태를 관찰

   (2) 어떠한 기준에 따라 행동을 선택

   (3) 환경으로부터 보상을 받음

   * 어떠한 기준 : 가치함수, 행동 선택 : greedy action selection

2. 에이전트 행동 선택의 기준

   (1) 에이전트는 매 타임스텝마다 보상을 더 많이 받으려 함

   (2) 단기적 보상만 고려한다면 최적의 정책에 도달할 수 있을까?

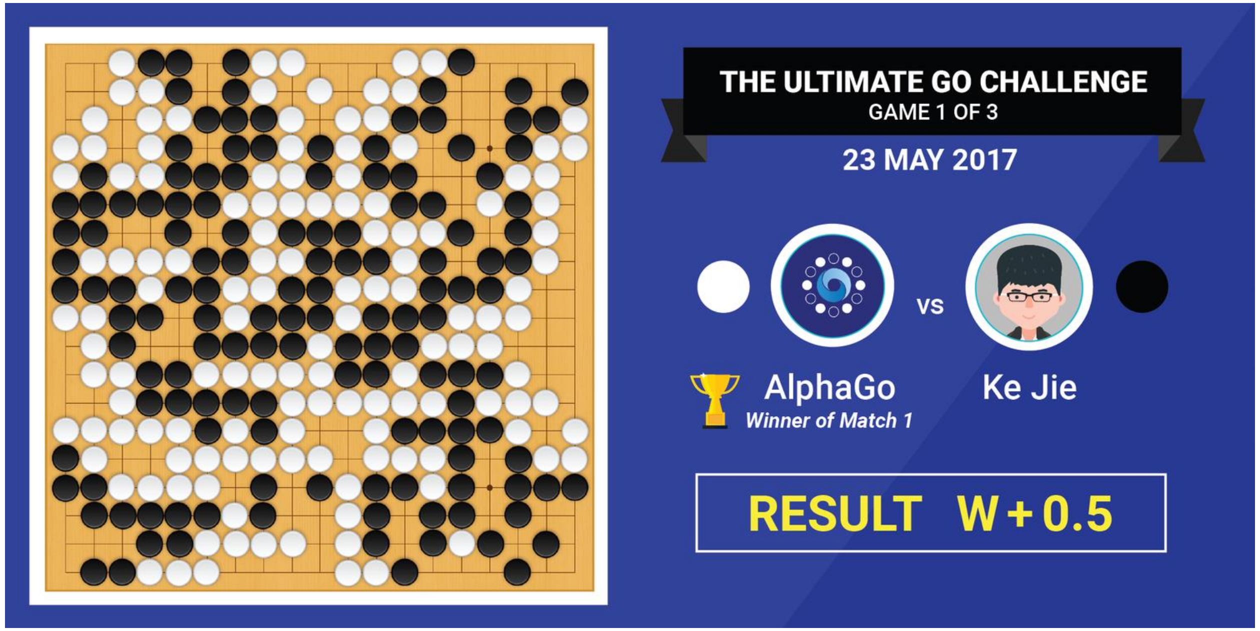
   - Problem 1: sparse reward
   - Problem 2: delayed reward

Sparse Reward

- 매 타임스텝마다 보상이 나오지 않는다면?

  Ex) 바둑

- 대부분의 경우 보상이 sparse 하게 주어짐



http://www.ibtimes.co.uk/alphago-defeats-worlds-best-go-player-ke-jie-humans-prove-no-match-ai-again-1622960

Delayed Reward

- 보통 선택한 행동에 대한 보상은 delay되어서 에이전트에게 주어진다

→즉각적 보상만 고려해서 선택하면 어떤 행동이 좋은 행동이었는지 판단 어려움

→“credit assignment problem”

이 행동만이 좋은 행동이고 나머지는 아니다?

장기적 보상

- 단기적 보상만 고려했을 때의 문제

  (1) Sparse reward (2) delayed reward

- 단기적 보상이 아닌 지금 선택한 행동에 대한 장기적인 결과(보상)를 보자!
- 장기적 보상을 어떻게 알아낼 수 있을까?

  (1) 반환값(Return)

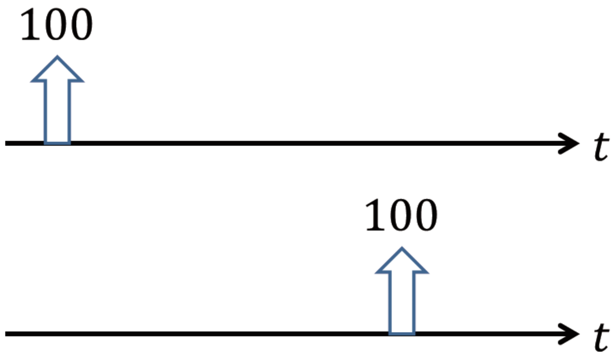
  (2) 가치함수(Value function)

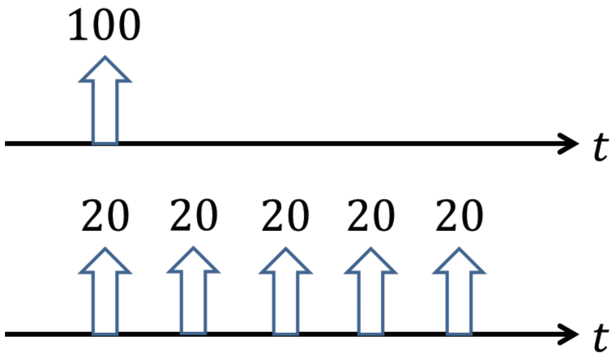
장기적 보상 1 : 단순합

- 단기적 보상이 아닌 장기적 보상 → 앞으로 받을 보상을 고려
- 현재 시간 t로부터 앞으로 받을 보상을 다 더한다

$$R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

- 현재 시간 t로부터 앞으로 받을 보상을 다 더한다

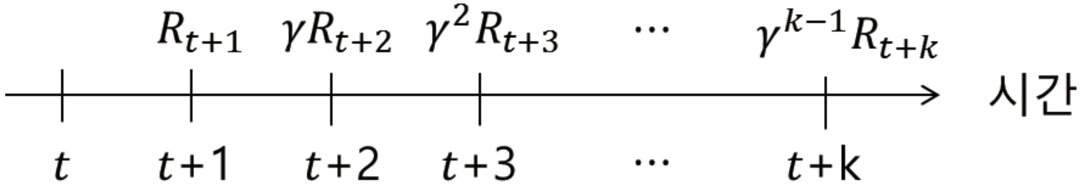




$$0.1 + 0.1 + \cdots = \infty$$
$$1 + 1 + \cdots = \infty$$

장기적 보상 2 : 반환값

- 현재 시간 t로부터 에피소드 끝까지 받은 보상을 할인해서 현재 가치로



- 반환값(Return) : 현재 가치로 변환한 보상들을 다 더한 값

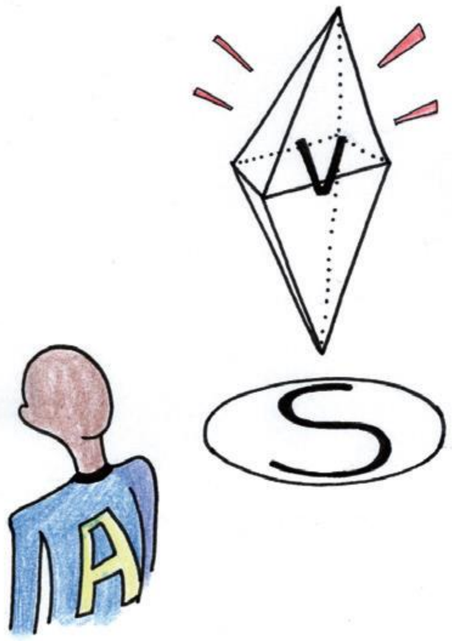
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$$

- 실제 에이전트와 환경의 상호작용을 통해 받은 보상을 이용
  - Unbiased estimator : 실제 환경으로부터 받은 값
  - High variance : 시간 t 이후에 행동을 어떻게 하는지에 따라 값이 크게 달라짐

장기적 보상 3 : 가치함수

- 가치함수(Value function) : 반환값에 대한 기댓값
  - 어떠한 상태 s에 갈 경우 그 이후로 받을 것이라 예상되는 보상에 대한 기대
  - 반환값은 에이전트의 정책에 영향을 받음

$$v_\pi(s) = \boldsymbol{E}_{\boldsymbol{\pi}}[G_t | S_t = s]$$



- 반환값은 상태 s에서 어떤 행동을 선택하는지에 따라 다름
- 가치함수는 상태 s로만 정해지는 값 → 가능한 반환값들의 평균
- 기댓값을 계산하기 위해서는 환경의 모델을 알아야함
  - DP는 가치함수를 계산
  - 강화학습은 가치함수를 계산하지 않고 sampling을 통한 approximation

가치함수 식의 변형

- 벨만 기대 방정식(Bellman expectation equation)의 유도

$$v_\pi(s) = \boldsymbol{E}_{\boldsymbol{\pi}}[G_t | S_t = s]$$

$$v_\pi(s) = \boldsymbol{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s]$$

$$v_\pi(s) = \boldsymbol{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma(R_{t+2} + \cdots) | S_t = s]$$

$$v_\pi(s) = \boldsymbol{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma \boldsymbol{E}_{\boldsymbol{\pi}}(R_{t+2} + \cdots) | S_t = s]$$

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

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

큐함수에 대한 정의

- 큐함수(Q-function)

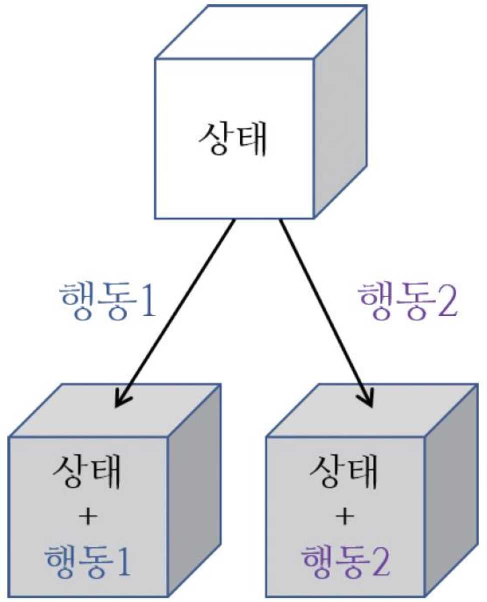
상태 s에서 행동 a를 했을 경우 받을 것이라 예상되는 반환값에 대한 기댓값

$$q_\pi(s,a) = \boldsymbol{E}_{\boldsymbol{\pi}}[G_t | S_t = s, A_t = a]$$

- 가치함수는 큐함수에 대한 기댓값

$$v_\pi(s) = \boldsymbol{E}_{a\sim\pi}[q_\pi(s,a) | S_t = s]$$

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



정책을 고려한 벨만 기대 방정식

- 정책 π에 따라 행동을 선택할 때의 벨만 기대 방정식

1. 가치함수에 대한 벨만 기대 방정식

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

2. 큐함수에 대한 벨만 기대 방정식

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

벨만 방정식은 현재 상태 s와 다음 상태 S_{t+1}의 가치함수(큐함수) 사이의 관계식

벨만 기대 방정식과 최적의 정책

- 벨만 기대 방정식 → 정책 π를 따라갔을 때의 가치함수

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

- 에이전트가 알고자 하는 것 : π^*
  - 가장 높은 보상을 얻게 하는 최적의 정책 π^*

- 최적의 정책 π^*는 deterministic policy
  - 상태 s에서 가장 큰 큐함수를 가지는 행동 a를 반환
  - 이 때, 큐함수 또한 최적의 큐함수

벨만 기대 방정식과 최적의 정책

- 최적의 큐함수 → 정책이 최적일 때 그에 따르는 큐함수도 최적

$$q^*(s,a) = \max_{\pi}[q_\pi(s)]$$

- 최적의 정책 (optimal policy)

$$\pi^*(s) = \underbrace{argmax_{a\in A}}_{}[q^*(s,a)]$$

Greedy policy : 가능한 행동 중에서 최고의 큐함수를 가지는 행동을 선택하는 정책

벨만 최적 방정식

- 일반 정책일 때 가치함수와 큐함수 사이의 관계

$$v_\pi(s) = \ \boldsymbol{E}_\pi[q_\pi(s, A_t)|S_t = s]$$

- 최적의 정책일 때 가치함수와 큐함수 사이의 관계

$$v^*(s) = \max_a[q^*(s, a)|S_t = s]$$

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

벨만 최적 방정식

- 벨만 최적 방정식 : 행동을 선택할 때는 max, 가치함수 or 큐함수도 최적
- 가치함수에 대한 벨만 최적 방정식

$$v^*(s) = \max_a \boldsymbol{E}[R_{t+1} + \gamma v^*(S_{t+1})|S_t = s, A_t = a]$$

- 큐함수에 대한 벨만 최적 방정식

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

정리

1. 단기적 보상 → 장기적 보상
   - Sparse reward & delayed reward
   - 가치함수 & 큐함수
2. 벨만 기대 방정식 (Bellman Expectation Eqn.)
   - $v_\pi(s) = \boldsymbol{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$
   - $q_\pi(s,a) = \boldsymbol{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$
3. 벨만 최적 방정식 (Bellman Optimality Eqn.)
   - $v^*(s) = \max_a \boldsymbol{E}[R_{t+1} + \gamma v^*(S_{t+1})|S_t = s, A_t = a]$
   - $q^*(s,a) = \boldsymbol{E}[R_{t+1} + \gamma\, max_{a'}\, q^*(S_{t+1}, a')\,|S_t = s, A_t = a]$

2-4. Value Iteration

강화학습 개요

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. 문제에 대한 수학적 정의 : Markov Decision Process
3. MDP를 계산으로 푸는 방법 : Dynamic Programming
4. MDP를 학습으로 푸는 방법 : Reinforcement Learning
5. 상태공간이 크고 차원이 높을 때 쓰는 방법 : Function Approximation
6. 바둑과 같은 복잡하고 어려운 문제를 푸는 방법 : Deep Reinforcement Learning

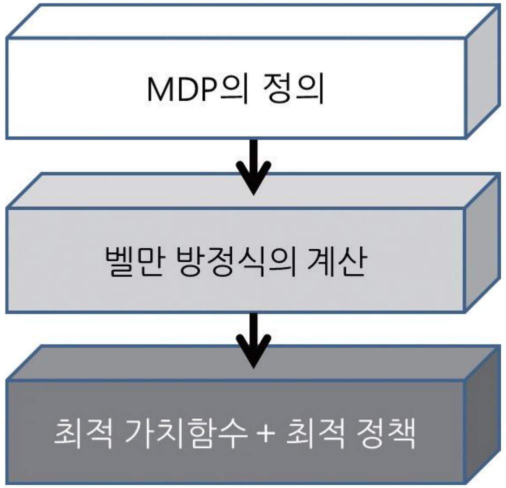
Dynamic Programming

- 알고리즘 강의에서 접하는 DP(Dynamic Programming)
  - 분할정복 기법 : 큰 문제를 해결하기 위해 작은 여러 문제로 나누기
  - 작은 문제들이 사실은 같은 문제를 푸는 것
  - 매번 재 계산하지 않고 값을 저장하고 재사용하는 것

- MDP에서의 DP
  - 큰 문제는 무엇이고 작은 문제는 무엇인가?
  - 반복되는 작은 문제는 어떻게 푸는지?
  - 저장되는 값이 무엇인지?

Dynamic Programming

1. MDP의 목표는 보상을 최대로 받는 정책을 구하기 : π^*
2. 큰 문제 : $\pi \rightarrow \pi^*$ (처음 π로부터 시작해서 π^*을 구하기)
   - 내재적 π : Value Iteration
   - 명시적 π : Policy Iteration
3. 작은 문제 : $\pi_k \rightarrow \pi_{k+1}$ (1 Iteration)
4. 반복되는 작은 문제를 푸는 방법 : Bellman Eq. (기대 or 최적)
5. 저장되는 값 : 가치함수



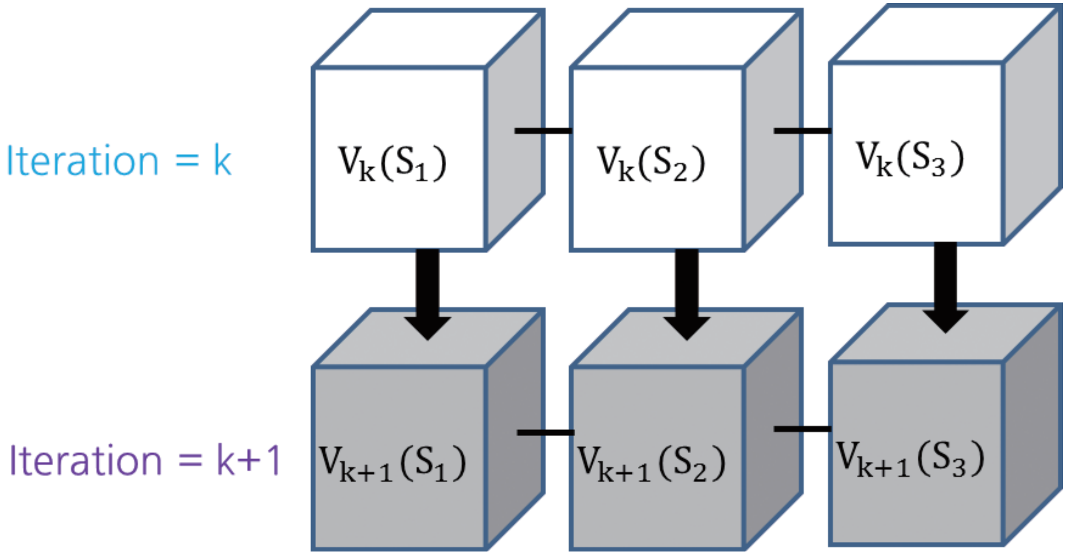
MDP의 해결 방법

Dynamic Programming

1. 저장되는 값이 가치함수이므로 결국 가치함수의 업데이트

2. 가치함수 업데이트의 반복적 계산

$$v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v^*$$



Value Iteration

1. MDP를 풀었다면 π^*와 v^*를 구한 것 → 벨만 최적 방정식

$$v^*(s) = \max_a \boldsymbol{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a]$$

2. 처음부터 벨만 최적 방정식을 만족한다고 가정 (iteration k)

$$v_k(s) \neq \max_a \boldsymbol{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]$$

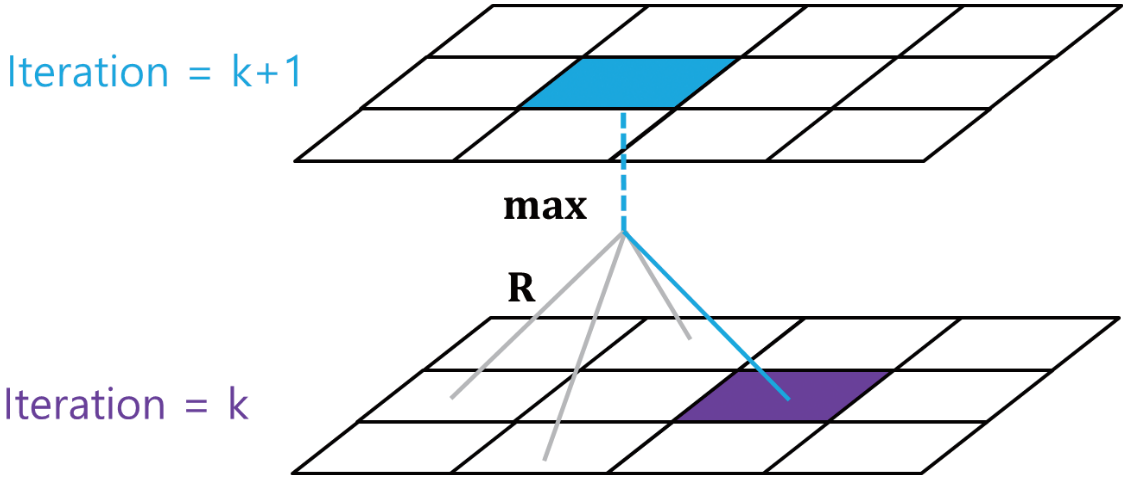
3. 벨만 최적 방정식을 통해 가치함수를 업데이트 → 가치함수가 수렴하면 greedy policy 로 선택

$$v_{k+1}(s) \leftarrow \max_a \left[R_s^a + \gamma \sum_{a \epsilon A} P_{ss'}^a \, v_k(s') \right]$$

Value Iteration

- Iteration 1번 : 모든 상태에 대해서 벨만 최적 방정식을 통한 업데이트 1번

$$\text{for } s \in \boldsymbol{S},\ v_{k+1}(s) \leftarrow \max_a \left[R_s^a + \gamma \sum_{a \epsilon A} P_{ss'}^a\, v_k(s') \right]$$



Value Iteration

- 벨만 최적 방정식을 통해 가치함수를 업데이트하려면 R_s^a와 $P_{ss'}^a$를 알아야함

→ DP가 model-based 인 이유 : learning이 아닌 Planning

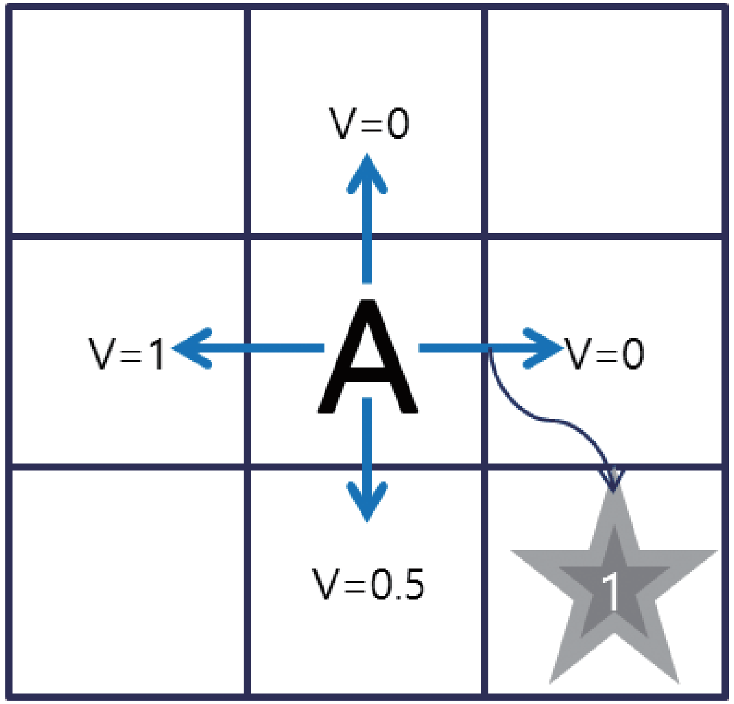
$$v_{k+1}(s) \leftarrow \max_a \left[R_s^a + \gamma \sum_{a \epsilon A} P_{ss'}^a v_k(s') \right]$$

- $P_{ss'}^a$를 그 행동이 가려는 상태에 대해서 1, 나머지 0이라고 가정
  - s'은 a가 right면 현재 상태에서 오른쪽에 있는 상태

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

Grid world에서 Value Iteration

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

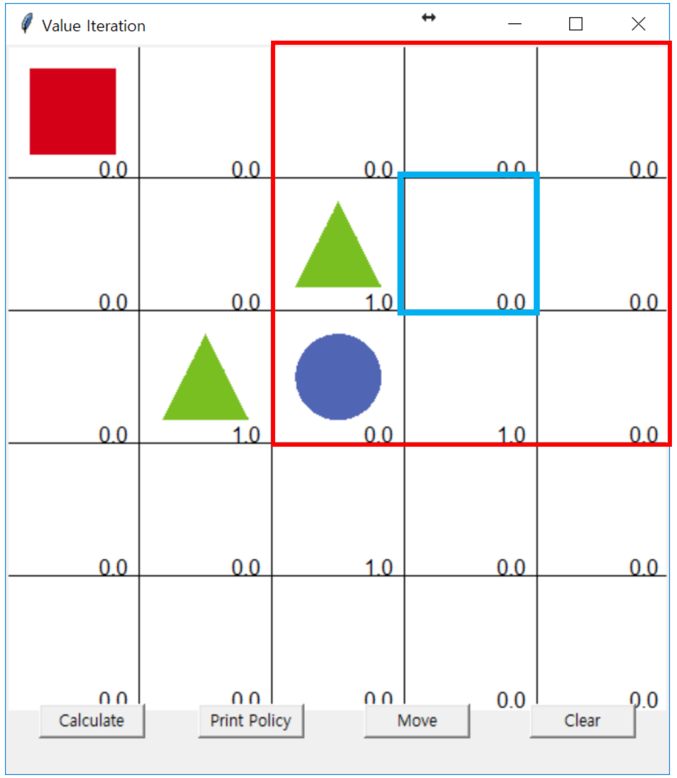


- ‘상’ : $0 + 0.9 \times 0 = 0$
- ‘하’ : $0 + 0.9 \times 0.5 = 0.45$
- ‘좌’ : $0 + 0.9 \times 1 = 0.9$
- ‘우’ : $1 + 0.9 \times 0 = 1$

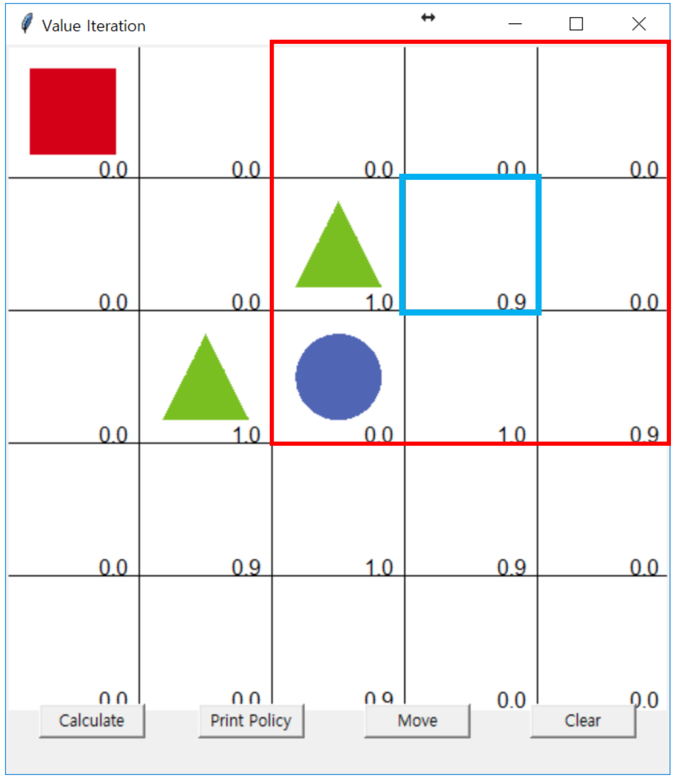
$$\begin{aligned} v_{k+1}(s) &= \max[0, 0.45, 0.9, 1] \\ &= 1 \end{aligned}$$

Grid world에서 Value Iteration

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$



K=1



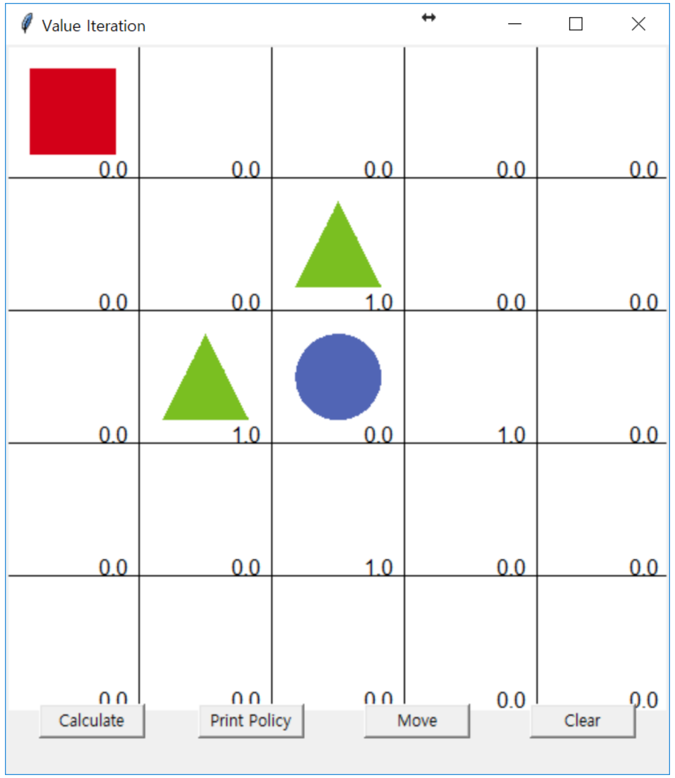
K=2

- '상' : $0 + 0.9 \times 0 = 0$
- '하' : $0 + 0.9 \times 1.0 = 0.9$
- '좌' : $-1 + 0.9 \times 1.0 = -0.1$
- '우' : $0 + 0.9 \times 0 = 0$

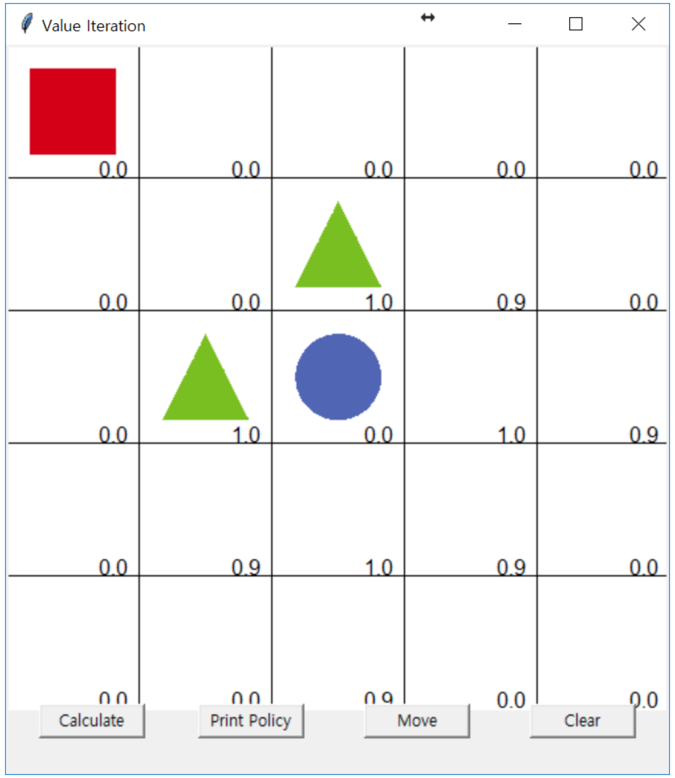
$$v_{k+1}(s) = \max[0, 0.9, -0.1, 0] = 0.9$$

Grid world에서 Value Iteration

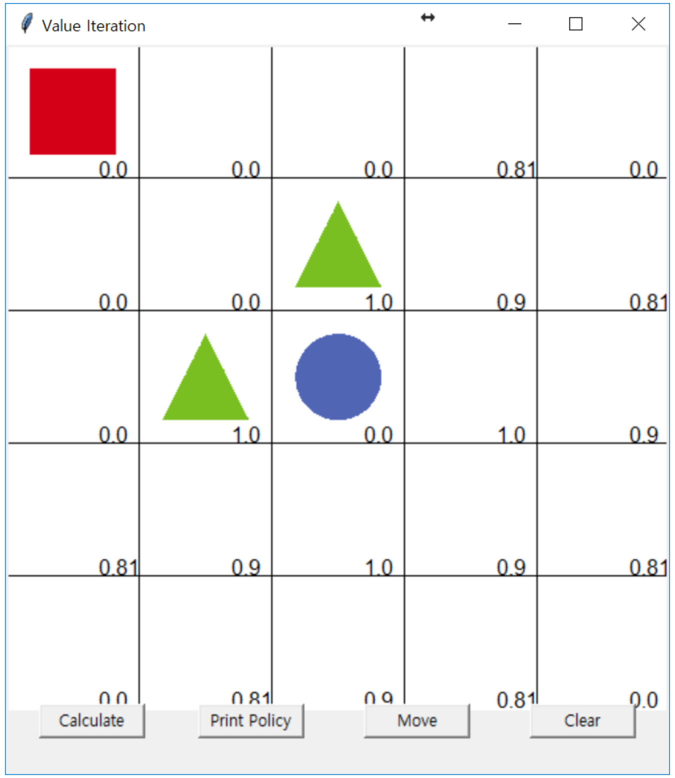
$$v_{k+1}(s) \leftarrow \max_{a}[R_s^a + \gamma v_k(s')]$$



K=1



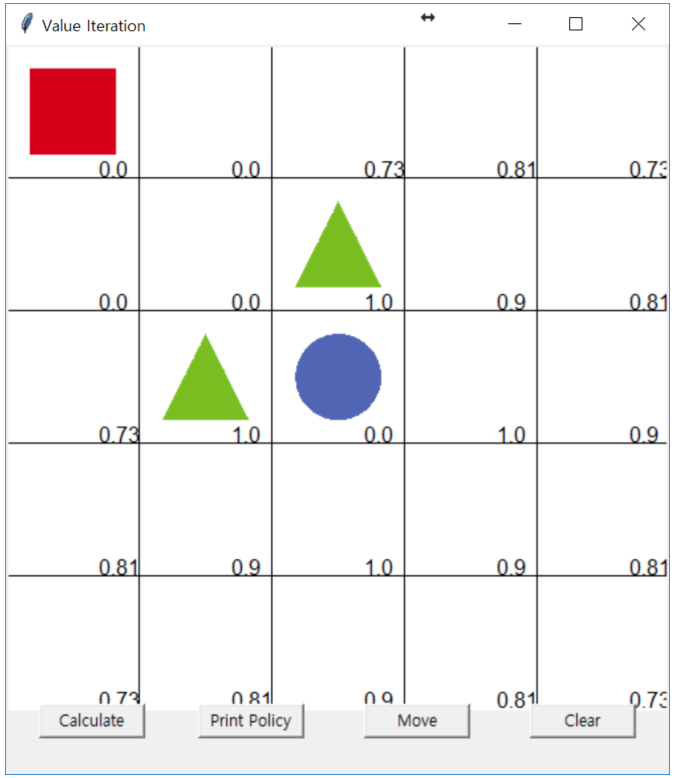
K=2



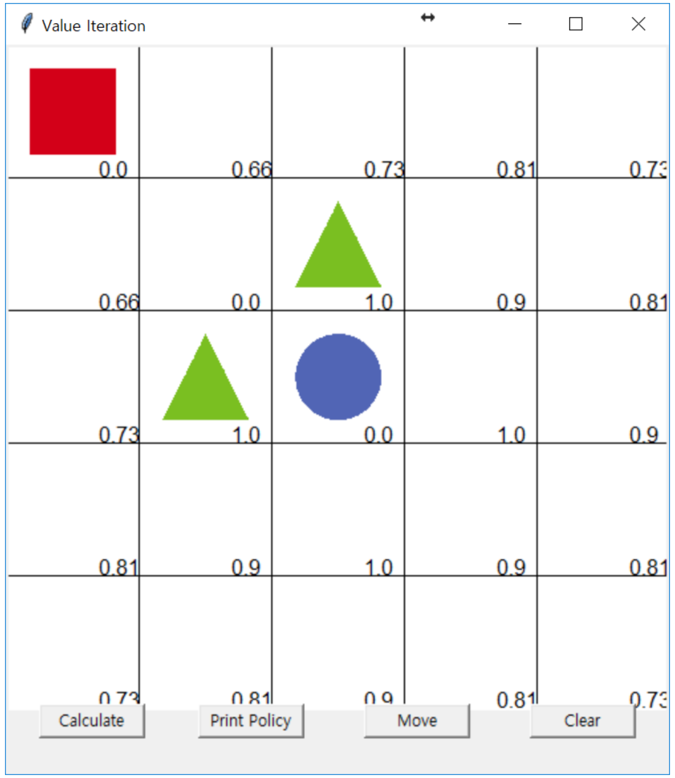
K=3

Grid world에서 Value Iteration

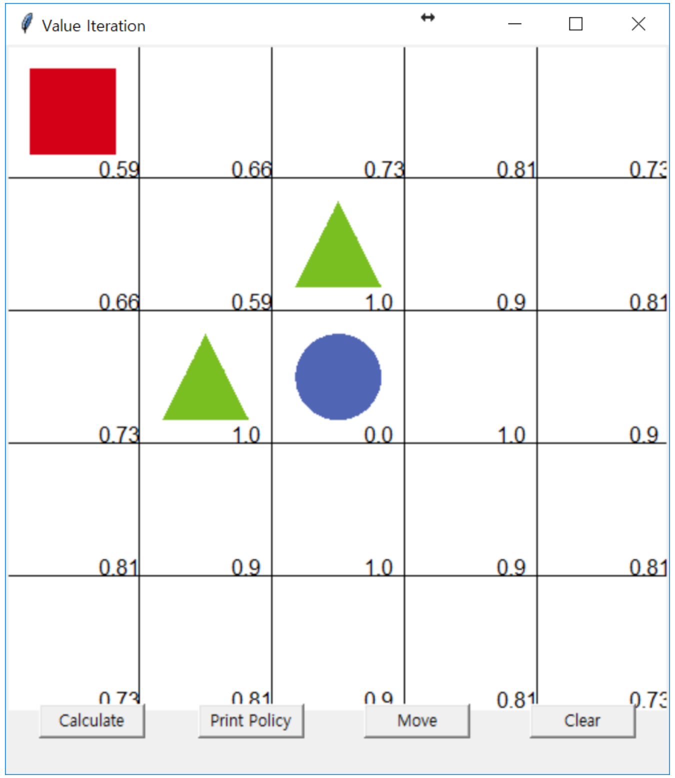
$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$



K=4



K=5

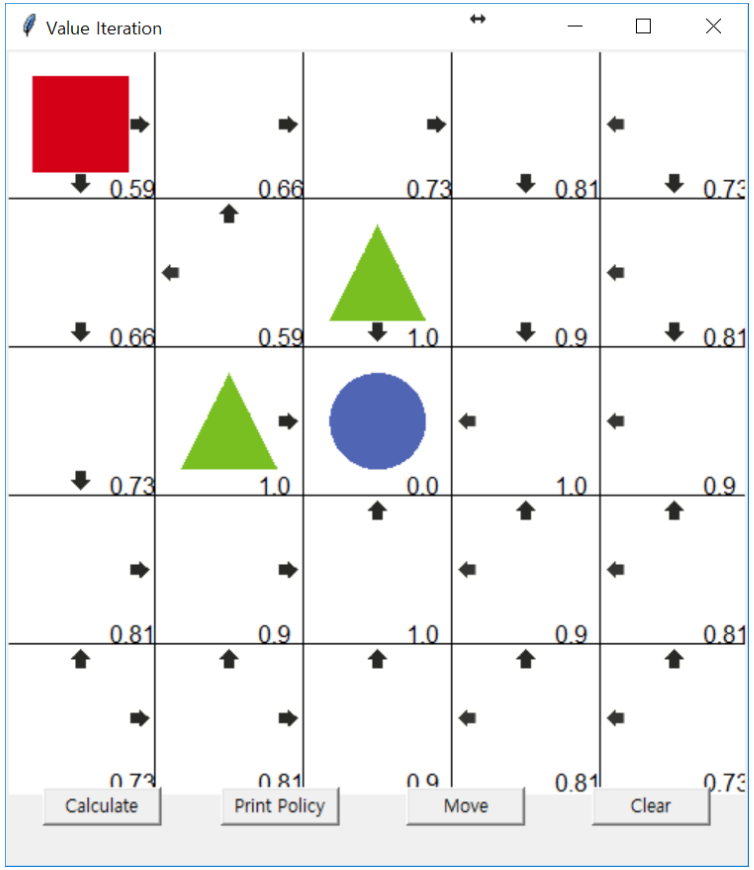


K=6

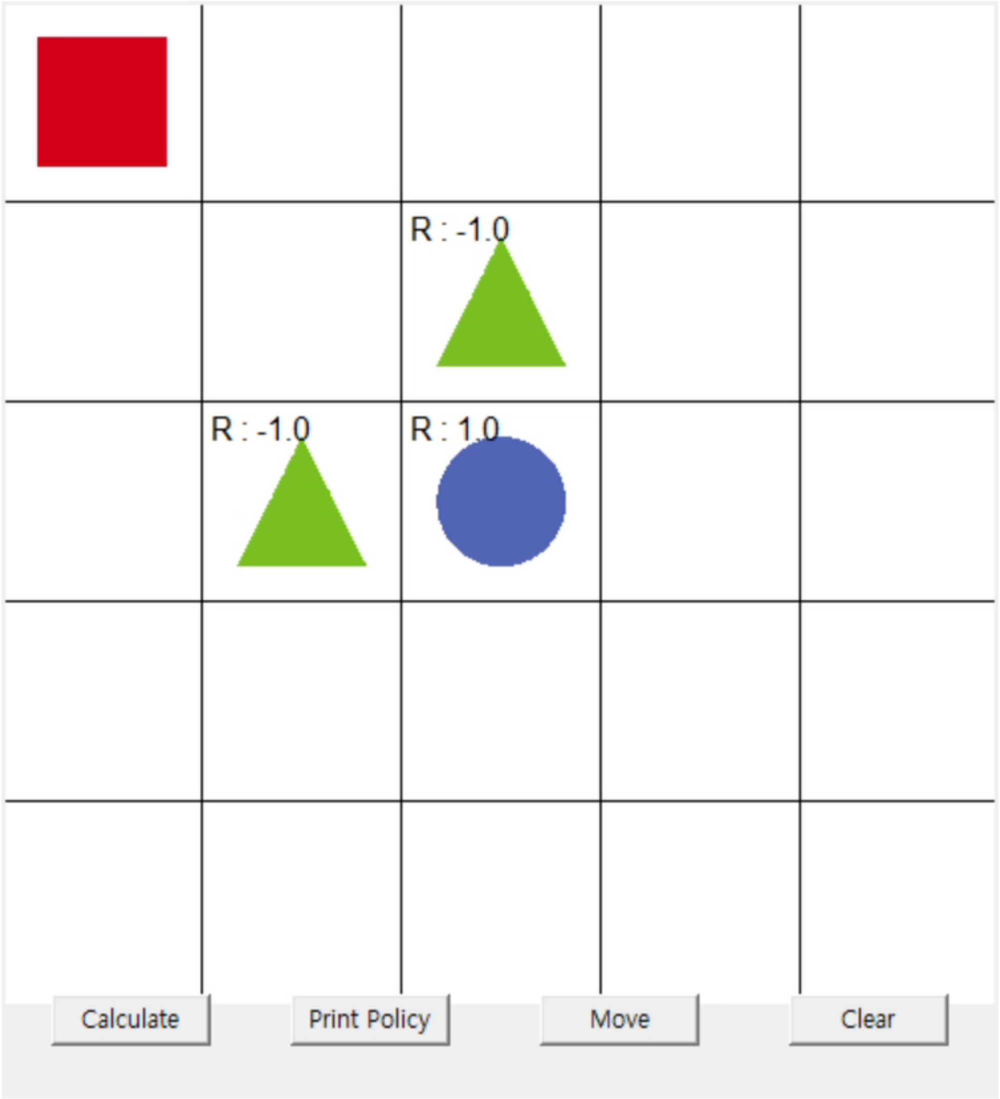
Grid world에서 Value Iteration

최적의 정책과 최적의 가치함수

$$\pi^*(s) = argmax_{a \in A} \boldsymbol{E}[R_s^a + \gamma v^*(s')]$$



Grid world에서 Value Iteration



정리

1. Dynamic Programming
   - 큰 문제를 작은 문제로, 반복되는 문제를 값을 저장하면서 해결
   - 큰 문제 : 최적 가치함수 계산 $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v^*$
   - 작은 문제 : 현재의 가치함수를 더 좋은 가치함수로 업데이트 $v_k \rightarrow v_{k+1}$
   - Bellman Eq.를 이용해서 1-step 계산으로 optimal을 계산하기

2. Value Iteration
   - 가치함수가 최적이라고 가정하고 그 사이의 관계식인 벨만 최적 방정식 이용

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

   - 수렴한 가치함수에 대해 greedy policy
   - Q-Learning으로 연결

2-5. Q-Learning

강화학습 개요

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. 문제에 대한 수학적 정의 : Markov Decision Process
3. MDP를 계산으로 푸는 방법 : Dynamic Programming
4. MDP를 학습으로 푸는 방법 : Reinforcement Learning
5. 상태공간이 크고 차원이 높을 때 쓰는 방법 : Function Approximation
6. 바둑과 같은 복잡하고 어려운 문제를 푸는 방법 : Deep Reinforcement Learning

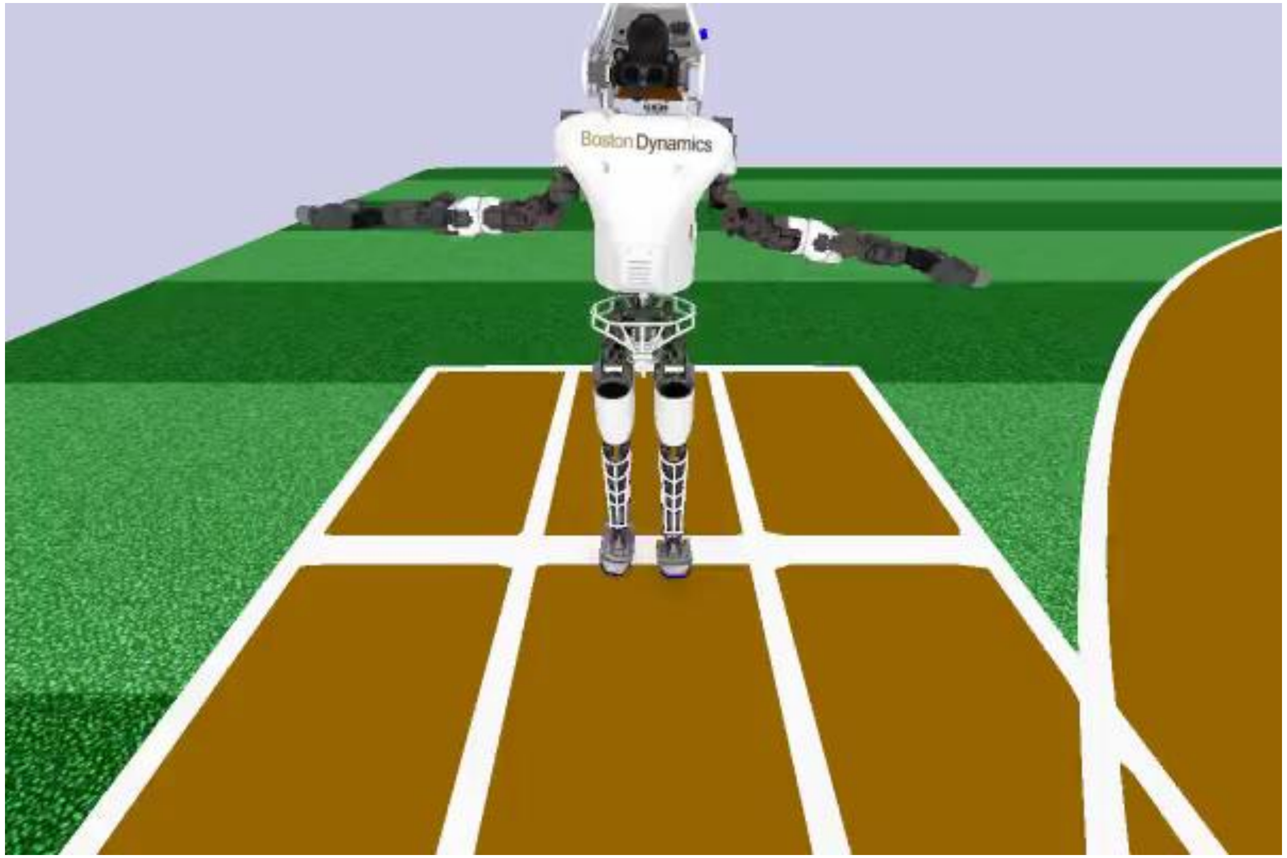
DP와 Reinforcement Learning

- Dynamic Programming
  - 벨만 방정식을 통한 가치함수 및 정책의 업데이트
  - 기댓값을 계산하기 위해 환경의 모델을 알아야함
  - 에이전트라는 개념이 없음

- 환경의 모델을 알아야하기 때문에 전문적인 지식이 필요
- 일정 이상 복잡한 문제에 적용하기 힘듬

Walking, Running

- Dynamics …???? → Reinforcement Learning : 모델없이 배우자!



https://blog.openai.com/openai-baselines-ppo/

Model-free RL

- Reinforcement Learning : 에이전트가 환경과 직접 상호작용

1. 일단 행동을 선택 후 환경에서 진행
2. 선택한 행동을 평가 (보상을 받음)
3. 평가한 대로 자신을 업데이트

- Model-free Learning : sampling을 통해 학습

Sampling

- 에이전트와 환경의 상호작용 : 에피소드

시간의 흐름

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, s_3, a_3, r_4, s_4, \cdots, s_T$$

샘플 1　　샘플 2　　…

- 전체 에피소드를 부분 부분 sampling
- Q-Learning
  - Value Iteration에 sampling을 적용
  - Sample : [s, a, r, s']

Q-Learning

1. Value Iteration

- 가치함수가 최적이라고 가정하고 그 사이의 관계식인 벨만 최적 방정식 이용

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

- 수렴한 가치함수에 대해 greedy policy

2. Q-Learning

- 행동 선택 : $\varepsilon -$ 탐욕정책

$$\pi(s) = \begin{cases} a^* = argmax_{a \in A} q(s,a), & 1-\varepsilon \\ \quad a \neq a^* \quad , & \varepsilon \end{cases}$$

- 큐함수 업데이트 : 벨만 최적 방정식 이용

$$q(s,a) = q(s,a) + \ \alpha \left(r + \gamma \max_{a'} q(s',a') - q(s,a) \right)$$

ε – 탐욕정책

1. 탐욕 정책

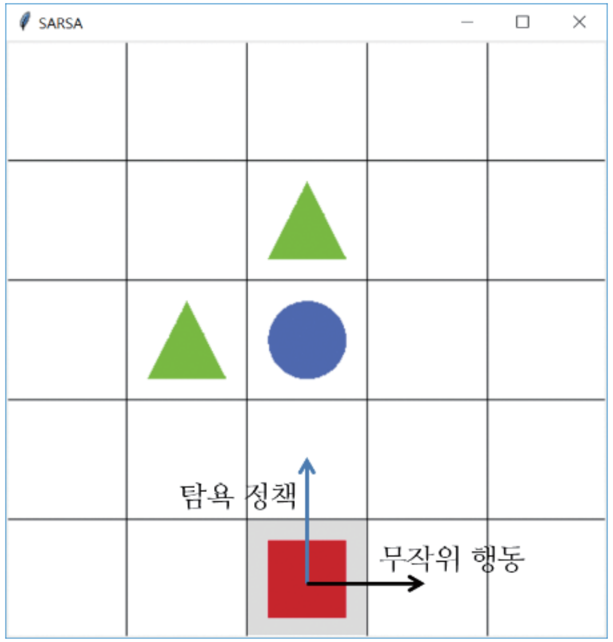
- 가치함수를 사용할 경우 $P_{ss'}^{a}$를 알아야 함

$$\pi(s) = argmax_{a \epsilon A}\left[R_s^a + \gamma \sum_{s' \in S} P_{ss'}^{a} v_{k+1}(s')\right]$$

- 큐함수를 이용한 탐욕 정책 발전 : model−free $\pi(s) \leftarrow argmax_{a\epsilon A}[q(s,a)]$

2. ε – 탐욕정책

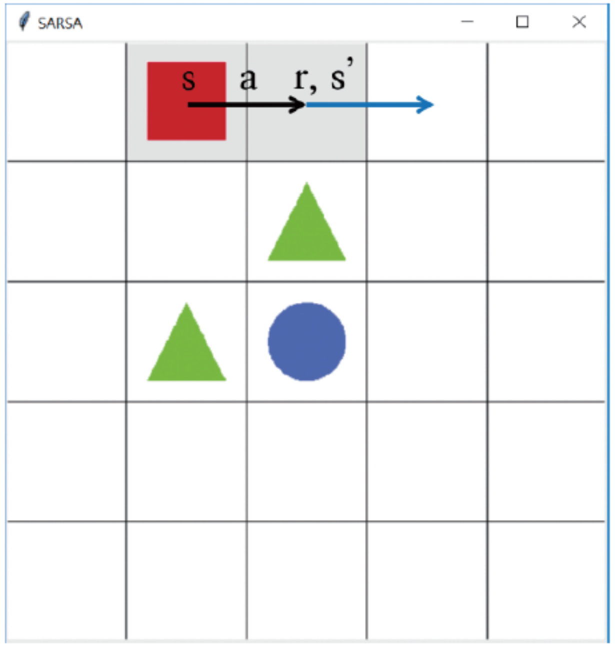
- ε의 확률로 랜덤한 행동을 선택 : $\pi(s) = \begin{cases} a^* = argmax_{a \in A} q(s,a), & 1-\varepsilon \\ random\ action & , \quad \varepsilon \end{cases}$



큐함수 업데이트

- 샘플[s, a, r, s'] 모으기

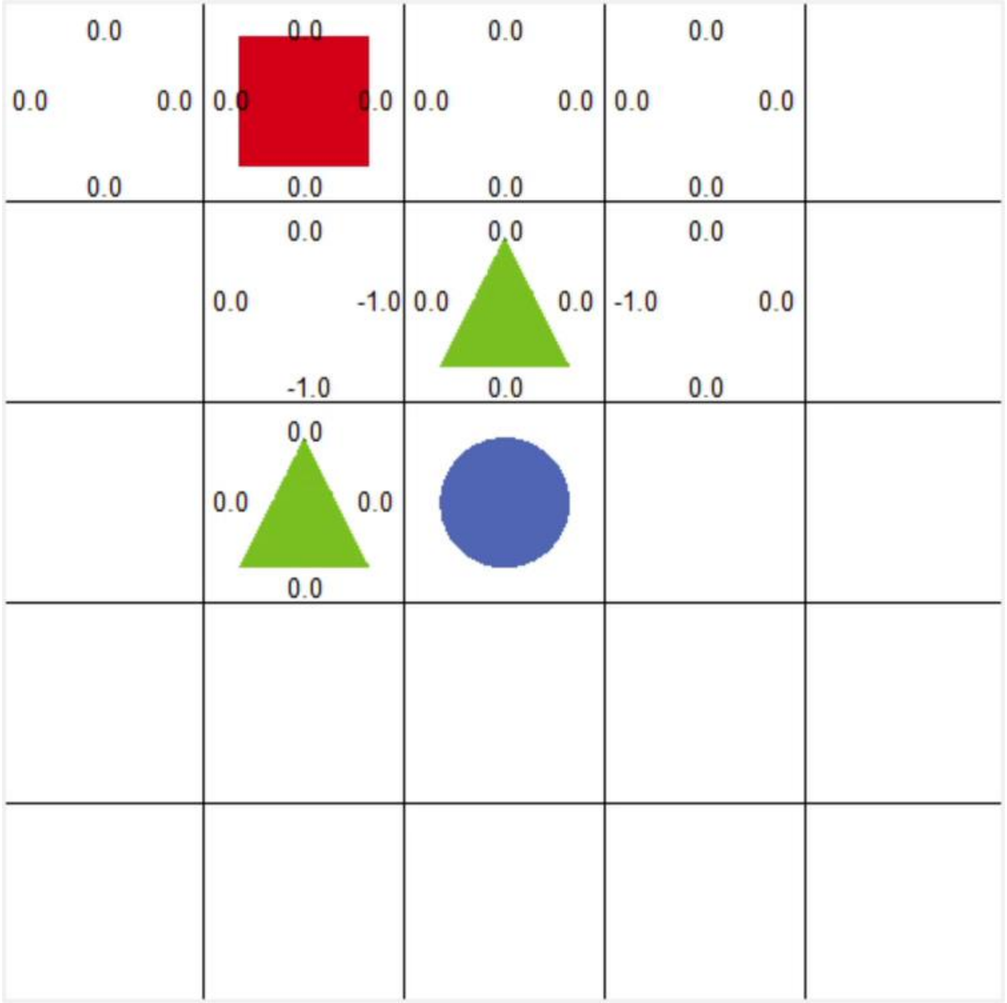
1. 상태 s에서 행동 a는 행동 정책(ε – 탐욕정책)으로 선택
2. 환경으로부터 다음 상태 s' 과 보상 r을 받음



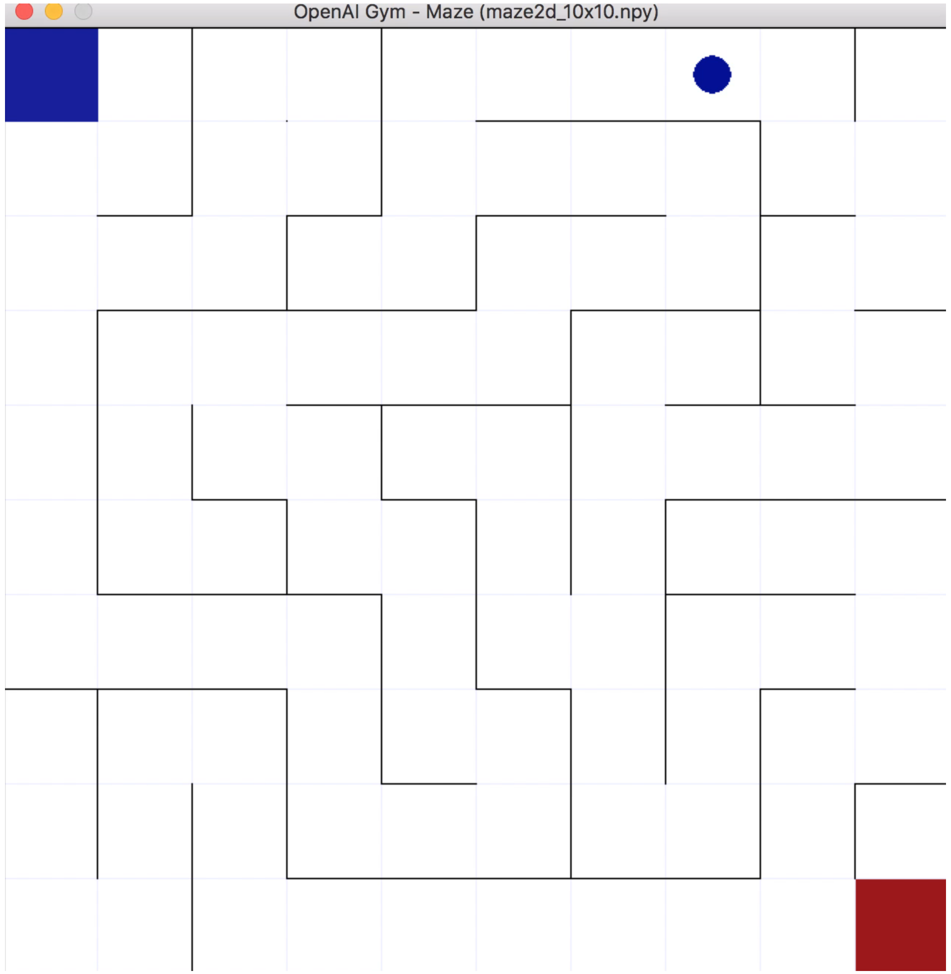
- 샘플로 $q(s,a)$를 업데이트
  - 벨만 최적 방정식을 이용

$$q(s,a) = q(s,a) + \alpha\left(r + \gamma \max_{a'} q(s',a') - q(s,a)\right)$$

Grid world에서의 Q-Learning



Maze에서의 Q-Learning



3. 딥러닝과 강화학습

3-1. Function Approximation

고전 강화학습의 한계

- Tabular Solution Methods
  - 모든 상태의 Q-function을 table의 형태로 저장
  - 모든 상태의 Q-function을 방문할 때마다 하나씩 업데이트
  - 모든 상태를 충분히 방문해야 optimal Q-function에 수렴

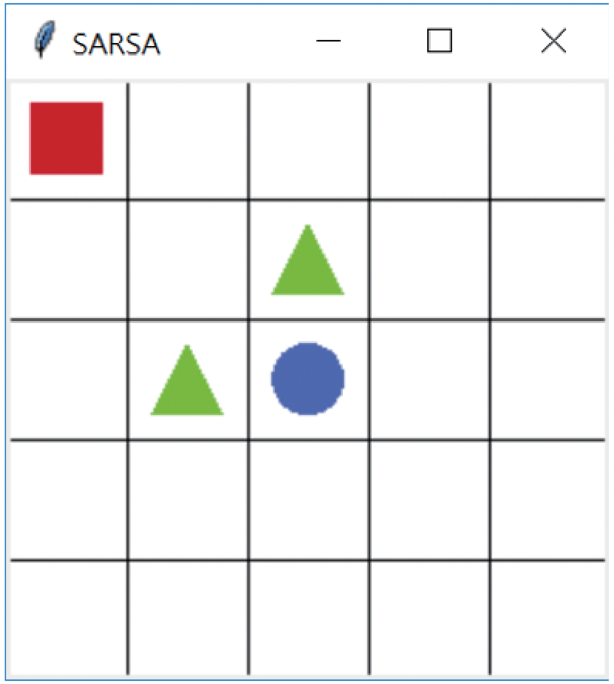
→ 비효율적인 업데이트 방식

→ 적용할 수 있는 문제의 제한

- 환경이 변하지 않는 문제
- 간단한 문제

고전 강화학습 알고리즘의 한계

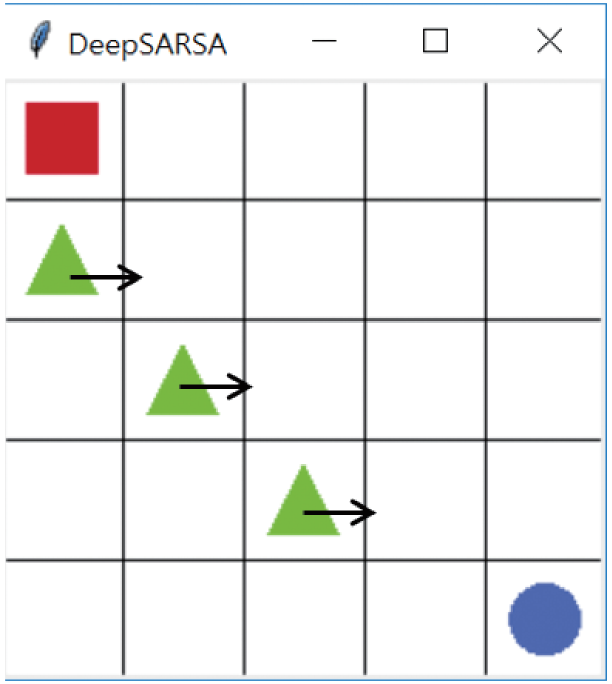
1. 환경이 변하지 않는 Grid world



상태 : 2차원, 25개

에이전트의 위치 x, y

2. 환경이 변하는 Grid world



상태 : 15차원, 18,225,000개

(1) 에이전트에 대한 장애물의 상대 위치 x, y

(2) 장애물의 속도(방향)

(3) 에이전트에 대한 도착지점의 상대 위치 x, y

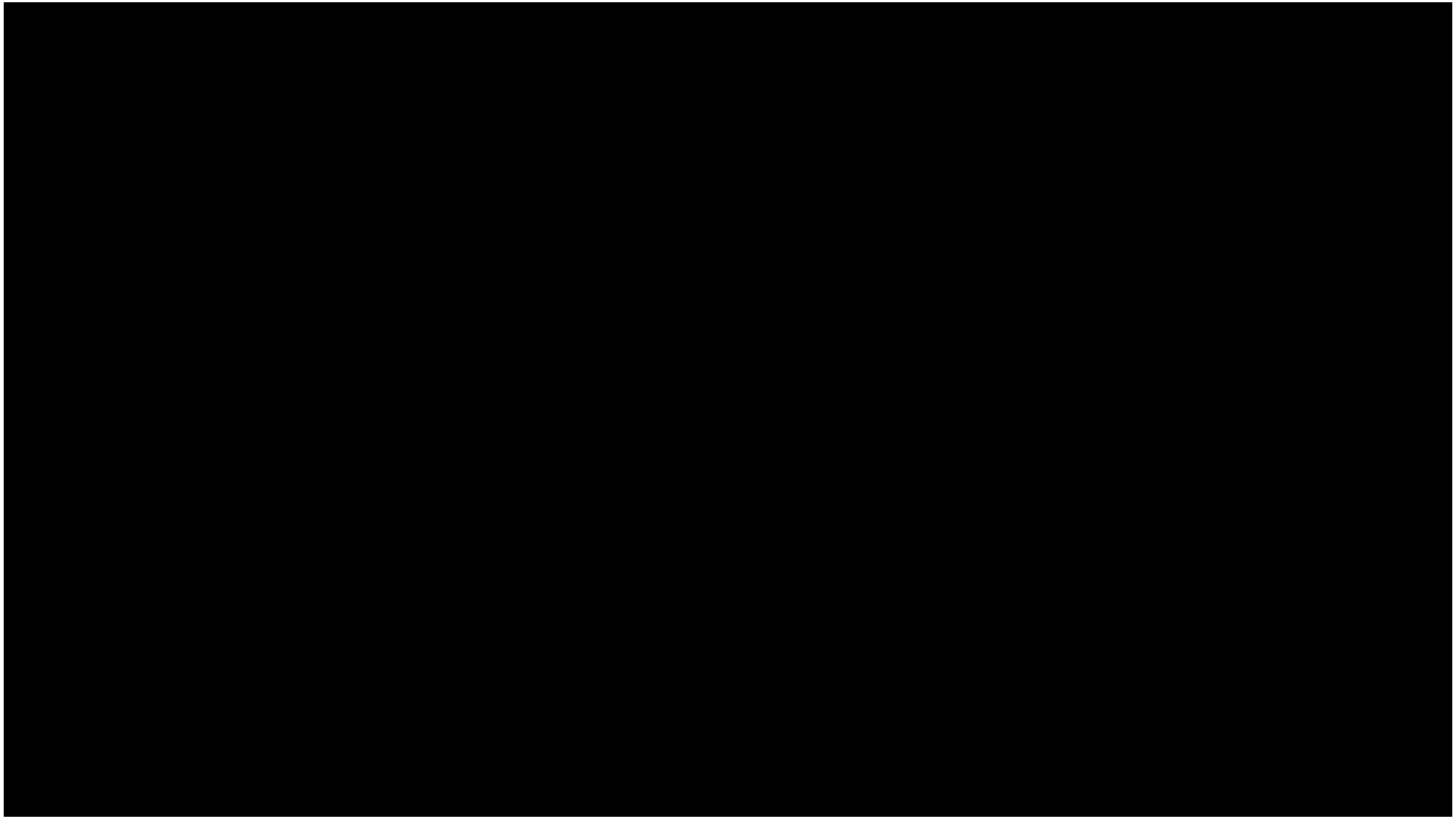
(4) 에이전트, 장애물, 도착점 레이블

고전 강화학습 알고리즘의 한계

1. TD-gammon이 학습했던 Backgammon의 가능한 state 수
   - 10^{20}개
2. AlphaGo가 학습했던 바둑의 가능한 state 수
   - 10^{270}개
3. Possible Camera image?
4. Robot, Drone → Continuous state space

https://www.cyberoro.com/news/news_view.oro?div_no=13&num=521047

우리가 하고 싶은 것



Function approximation을 통한 generalization

1. 대부분 강화학습을 적용하고 싶은 문제 → Large state space
2. Large state space : 많은 양의 메모리, 계산량, 데이터 필요
3. 한정된 양의 자원 → Good approximate solution
4. 비슷한 state는 비슷한 function의 output을 가질 것 → Generalization!!
   - 어떻게 지금까지 방문한 state들에 대한 경험으로 전체 문제에 대해 generalize 할 수 있을까?
5. Generalization을 하기 위해 Supervised Learning의 기법을 가져다 쓰자!
   - Function Approximation : Target function 이 있고 그 target function을 approximate 하는 function을 찾기

강화학습과 Function approximation

- Q-Learning에서

1. Target function : 큐함수
2. 방문한 상태들에 대한 경험 → 다른 상태들의 큐함수를 generalize
3. Approximate하는 함수의 parameter : θ

$$q_\theta(s,a) \sim q_\pi(s,a)$$

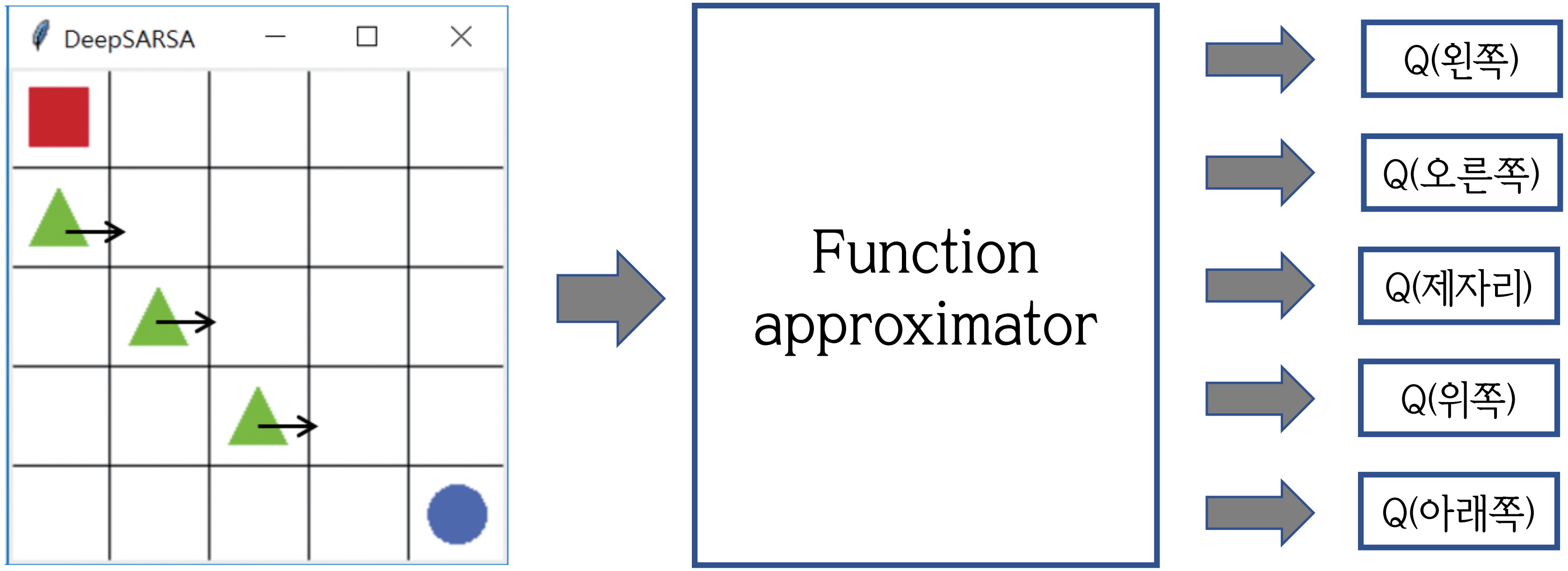
강화학습과 Function approximation

1. Table 형태의 큐함수

A \ S	s_0	s_1	s_2	s_3	s_4	$\cdots$
a_0	$q(s_0, a_0)$	$q(s_1, a_0)$	$q(s_2, a_0)$	$q(s_3, a_0)$	$q(s_4, a_0)$	$\cdots$
a_1	$q(s_0, a_1)$	$q(s_1, a_1)$	$q(s_2, a_1)$	$q(s_3, a_1)$	$q(s_4, a_1)$	$\cdots$
a_2	$q(s_0, a_2)$	$q(s_1, a_2)$	$q(s_2, a_2)$	$q(s_3, a_2)$	$q(s_4, a_2)$	$\cdots$
a_3	$q(s_0, a_3)$	$q(s_1, a_3)$	$q(s_2, a_3)$	$q(s_3, a_3)$	$q(s_4, a_3)$	$\cdots$

강화학습과 Function approximation

2. 함수 형태로 근사한 큐함수



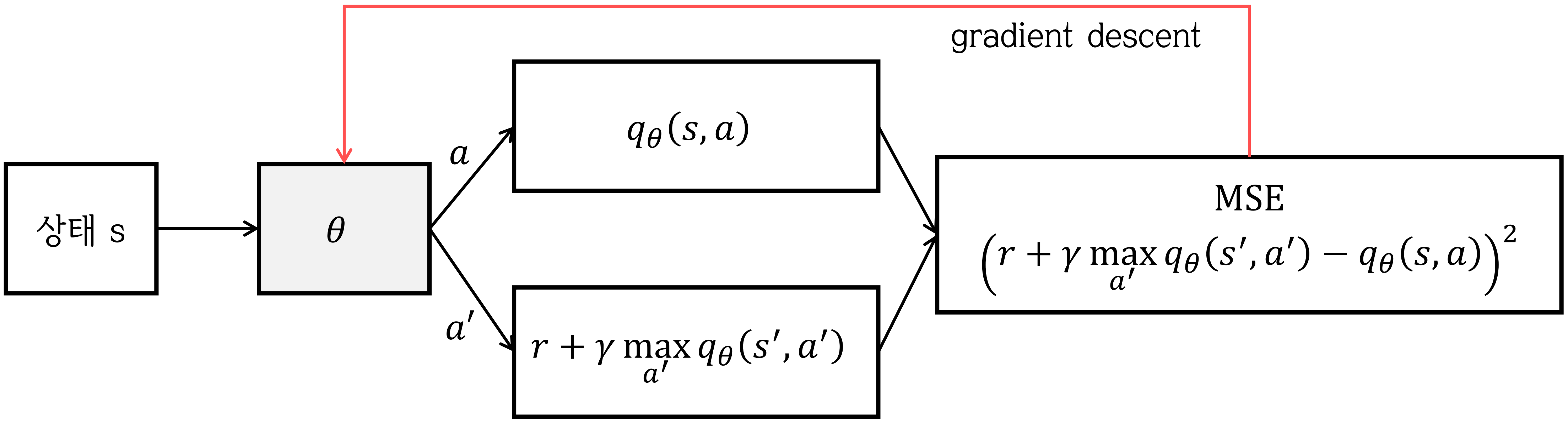
상태 s → $q_\theta(s,a)$ → 각 행동 별 큐함수 값

Mean square error

1. 업데이트 대상 : 각 큐함수의 값 → 큐함수의 parameter 값
   - $q_\theta(s, a)$의 θ
2. Function approximation을 사용하므로 지도학습의 기법들을 가져올 수 있음
3. 지도학습 : 정답과 예측의 오차를 최소화하도록 모델을 학습
   - 정답과 예측의 오차 정의 : Mean square error
   - 오차를 최소화할 방법을 결정 : gradient descent

Mean square error

1. 큐함수의 값은 continuous → regression
2. 큐함수의 값은 0에서 1 사이가 아님 → MSE(Mean square error)
3. 큐함수를 업데이트 loss function → Bellman Equation에서의 TD error를 이용

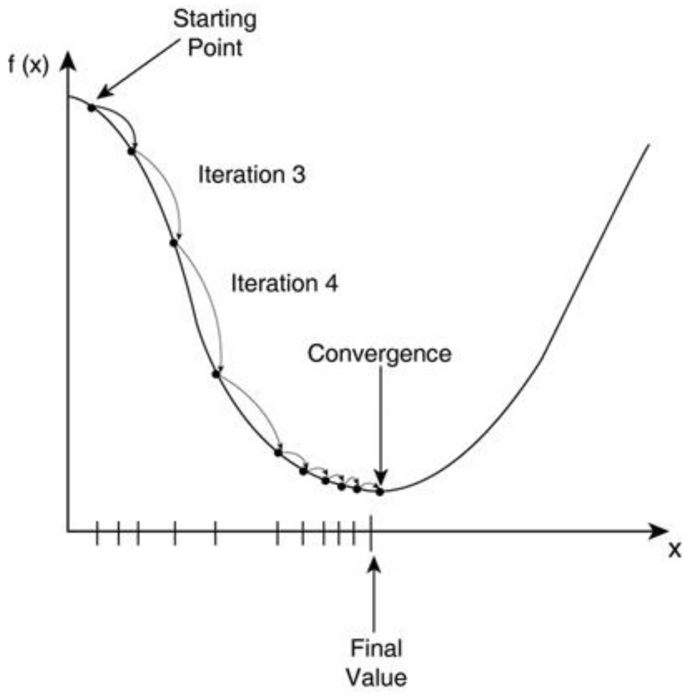


Q-Learning with function approximation

Q-Learning with function approximator 학습하는 과정

1. 상태 관찰, 행동 선택
2. 행동하고 다음 상태와 보상을 받음
3. TD error의 gradient를 따라 큐함수의 parameter를 업데이트

$$\left(r + \gamma \max_{a'} q_\theta(s', a') - q_\theta(s, a)\right)^2$$



강화학습과 Neural Network

1. Nonlinear function approximation → Neural Network
   - Neural Network의 activation function이 nonlinear
2. Neural Network를 이용한 큐함수 근사
   - $q_\theta(s, a)$의 θ가 Neural Network의 parameter

3. 큐함수의 업데이트
   - MSE error에 대한 gradient :

$$\nabla_\theta \left(r + \gamma \max_{a'} q_\theta(s', a') - q_\theta(s, a)\right)^2$$

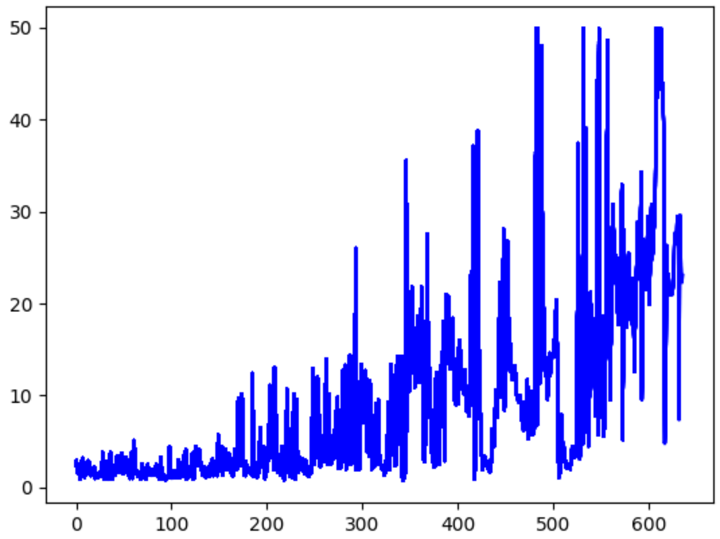
   - 계산한 gradient를 backpropagation

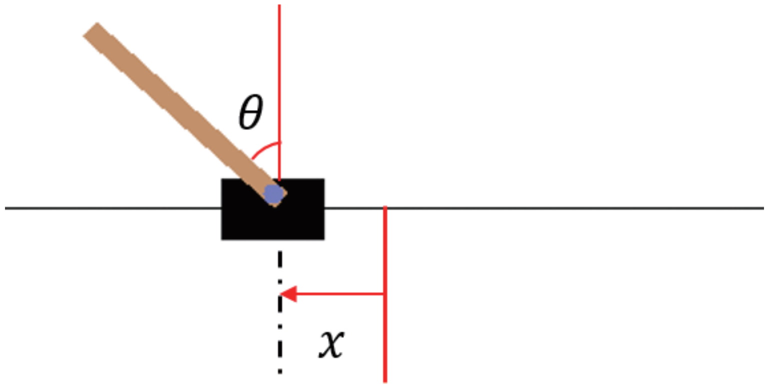
Cartpole과 Neural Network

1. Neural network 구조

```
def build_model(self):
    model = Sequential()
    model.add(Dense(20, input_dim=self.state_size, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
    return model
```

2. 학습 결과

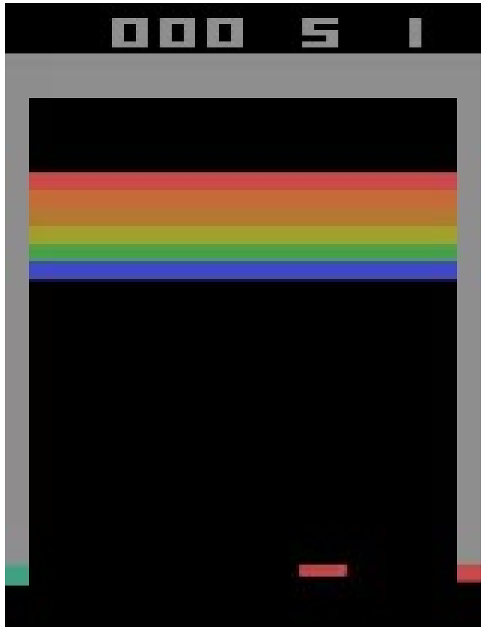




3-2. Deep Q-Network

Deep Q-Learning

1. Deep Reinforcement Learning = Reinforcement Learning + Deep Learning
2. DQN(2013년)
    - "Playing Atari with Deep Reinforcement Learning"–Mnih, 2013
    - DeepMind의 초창기 논문
    - Atari game을 화면으로부터 학습
3. 화면은 high-dimension → Deep Learning을 사용



DQN의 네 가지 Key point

- Function approximator로 neural network를 사용할 경우 에이전트가 수렴하지 못하고 발산
- 이 문제를 해결하기 위해 지도학습의 방법을 가져옴 → DQN
- DQN의 네 가지 특징

1. CNN
2. Experience replay
3. Online learning with Stochastic gradient descent
4. Target Q-network

DQN의 네 가지 Key point

1. CNN
   - 화면으로부터 direct로 학습할 수 있음
2. Experience replay
   - Sample 들의 상관관계를 깸 → Neural Network의 안정적인 학습
   - 일정한 크기를 가지는 memory (FIFO)
3. Online update with stochastic gradient descent
   - 매 스텝마다 replay memory에서 추출한 mini-batch로 Q-function 업데이트
   - 점진적으로 변하는 Q-function에 대해 $\varepsilon - greedy\ policy$로 행동 선택

DQN의 네 가지 Key point

- Q-Learning update

$$q(s,a) = q(s,a) + \ \alpha\left(r + \gamma \max_{a'} q(s',a') - q(s,a)\right)$$

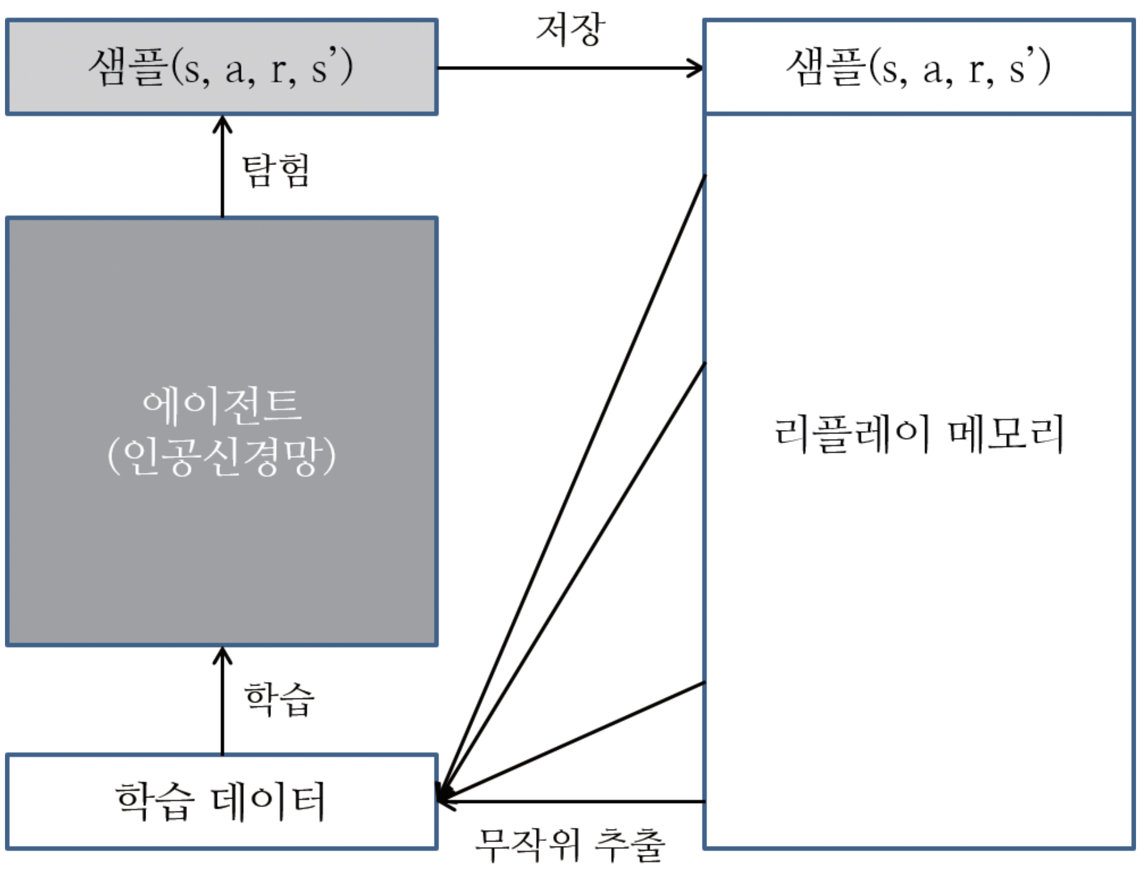
- DQN update : MSE error를 backpropagation

$$MSE\ error :\ \left(\underline{r + \gamma \max_{a'} q_{\theta^-}(s',a')} - q_{\theta}(s,a)\right)^2$$

4. Target Q-network

- Target network θ^-의 사용 : update의 target이 계속 변하는 문제를 개선
- 일정주기마다 현재의 network θ^- 를 θ로 업데이트

DQN의 다이어그램



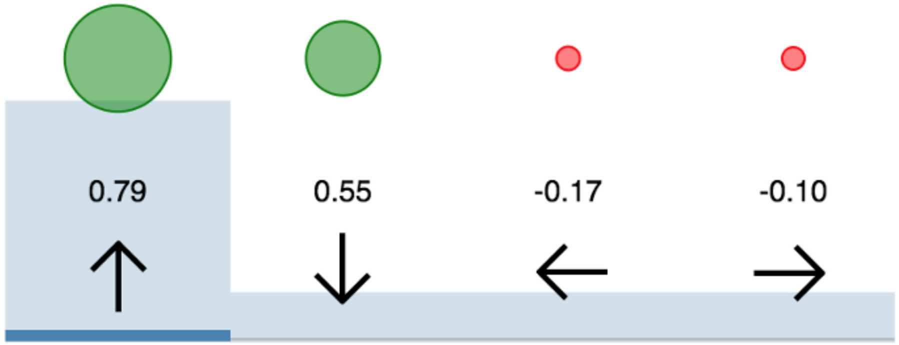
DQN의 학습과정

1. exploration
2. append sample to replay memory
3. random sampling, training
4. target Q-network update

DQN의 학습과정

1. exploration

- 정책은 큐함수에 대한 $\varepsilon - greedy\,policy$
- ε은 time-step에 따라서 decay → 점점 수렴
- ε은 1에서 시작해서 0.1까지 decay, 0.1을 계속 유지 → 지속적 탐험



DQN의 학습과정

2. append sample to replay memory

- 에이전트는 $\varepsilon - greedy\ policy$에 따라 샘플 [s, a, r, s']을 생성
- 샘플을 replay memory에 append
  - Replay_memory.append(sample)

```
def append_sample(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))
```

- Replay memory가 다 차면 오래된 sample부터 하나씩 빼고 새로운 sample을 memory에 넣기

DQN의 학습과정

3. random sampling, training

```
# 메모리에서 배치 크기만큼 무작위로 샘플 추출
mini_batch = random.sample(self.memory, self.batch_size)
```

- Mini-batch (32개) 샘플을 추출
- 샘플로부터 target값과 prediction 값을 구하기 (32개)
  - $MSE\ error : (target\ - prediction)^2$
  - Target : $r + \gamma \max_{a'} q_{\theta^-}(s', a')$
  - Prediction : $q_\theta(s, a)$
- MSE error에 대한 gradient backpropagation

4. target Q-network update : 일정 주기마다 target Q-network를 현재 Q-network로 업데이트

DQN의 학습과정

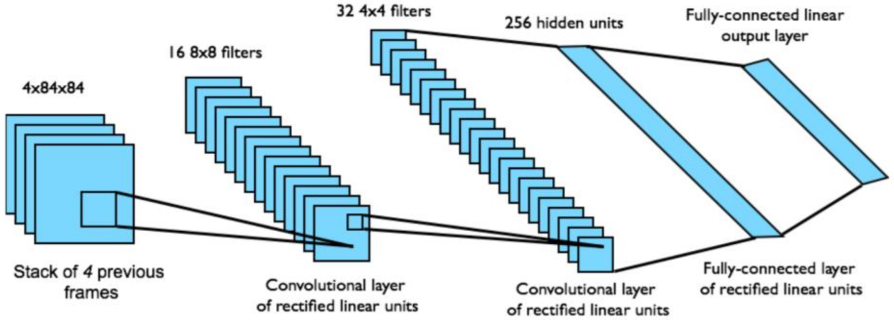
1. 상태에 따른 행동 선택
2. 선택한 행동으로 환경에서 1 time-step을 진행
3. 환경으로부터 다음 상태와 보상을 받음
4. Sampling[s, a, r, s']을 replay memory에 저장
5. Replay memory에서 random sampling → mini-batch update
6. 일정 주기마다 Target network 업데이트

DQN의 세부사항

1. CNN network
2. 4 images 1 history
3. 30 no-op
4. Huber loss

DQN의 세부사항

1. CNN network
   - Image를 input으로 받음
   - 3개의 convolution layer(no pooling layer)



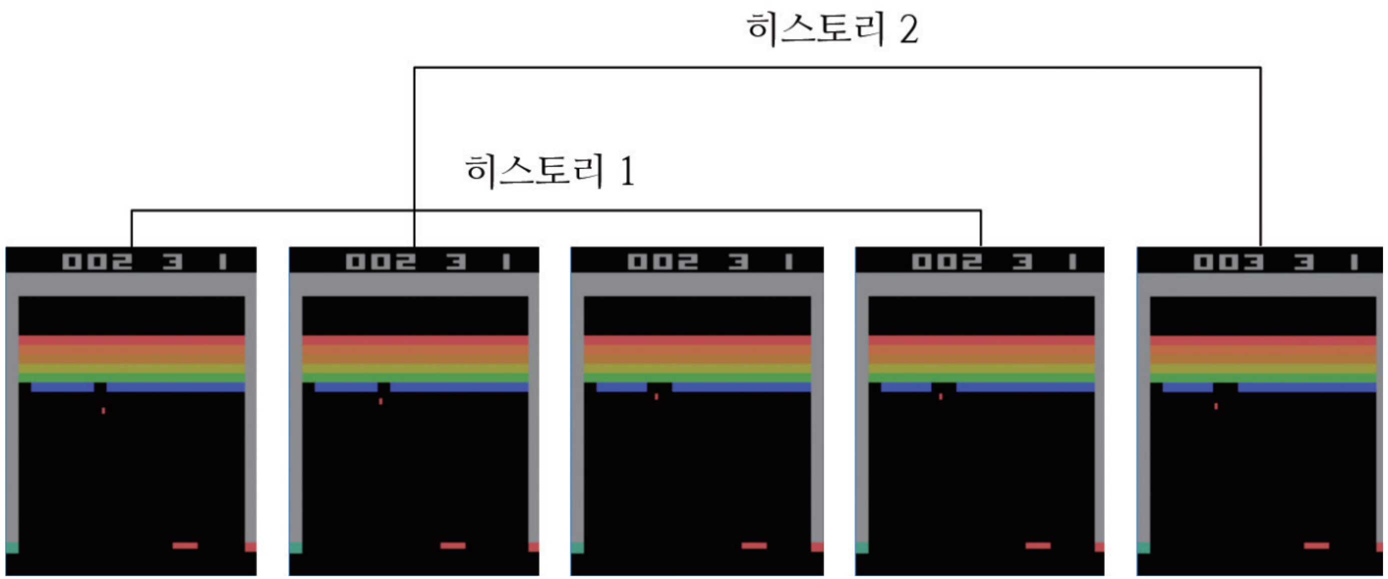
Network architecture and hyperparameters fixed across all games
[Mnih et al.]

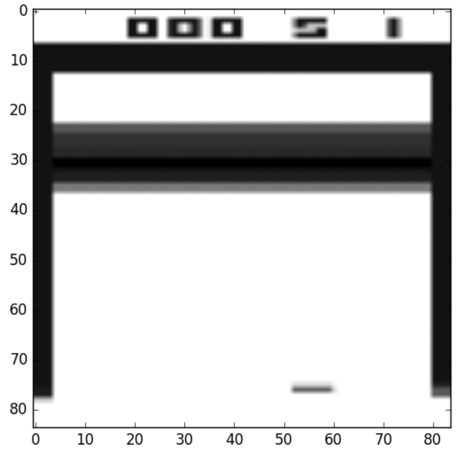
```python
# 상태가 입력, 큐함수가 출력인 인공신경망 생성
def build_model(self):
    model = Sequential()
    model.add(Conv2D(32, (8, 8), strides=(4, 4), activation='relu',
                     input_shape=self.state_size))
    model.add(Conv2D(64, (4, 4), strides=(2, 2), activation='relu'))
    model.add(Conv2D(64, (3, 3), strides=(1, 1), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(self.action_size))
    model.summary()
    return model
```

DQN의 세부사항

2. 4 images 1 history

- Image 한 장은 공의 속도 등의 정보를 표현하지 못함
- 현재로부터 이전까지 4개의 연속된 화면을 input으로

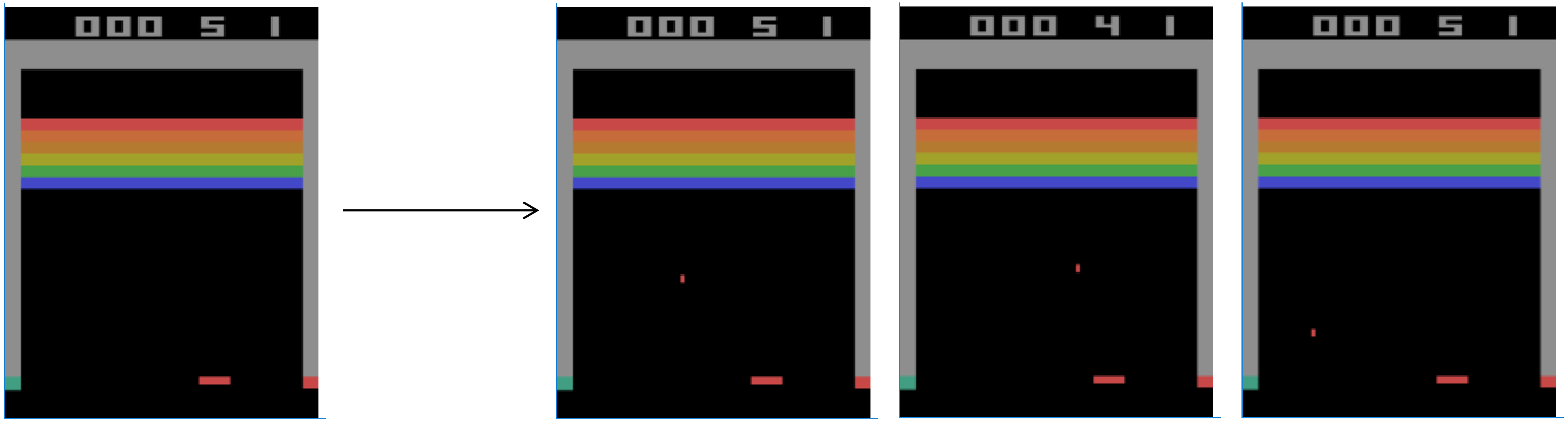




DQN의 세부사항

3. 30 no-op

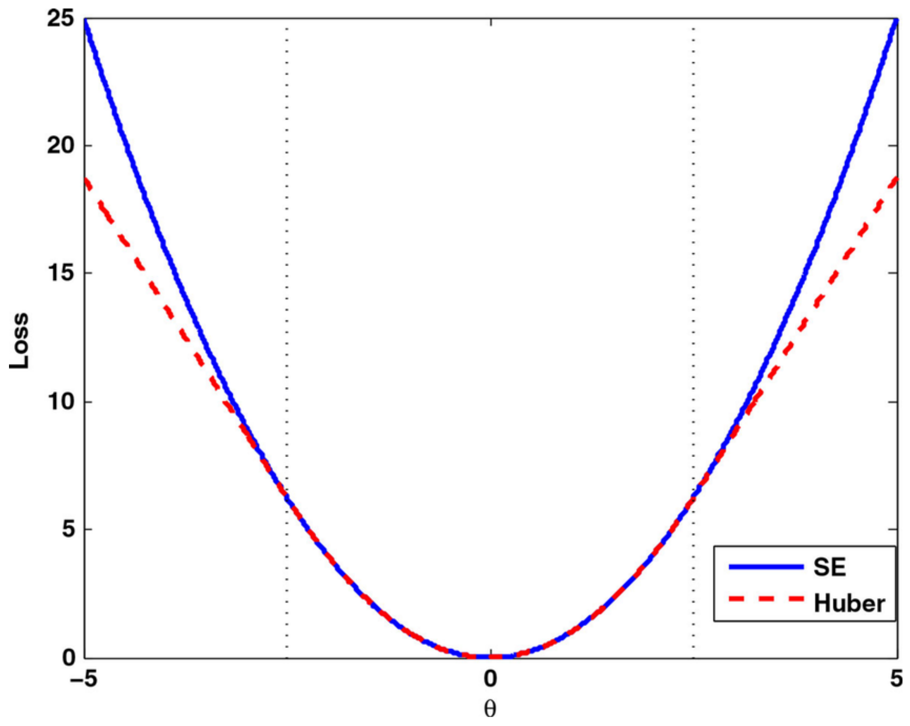
- 항상 같은 상태에서 시작 → 초반에 local optimum으로 수렴할 확률이 높다
- 조금은 다른 상태에서 시작 : 0에서 30 time-step 중에 랜덤으로 선택한 후 그동안 아무것도 안 하기 (no-op)



DQN의 세부사항

4. Huber loss

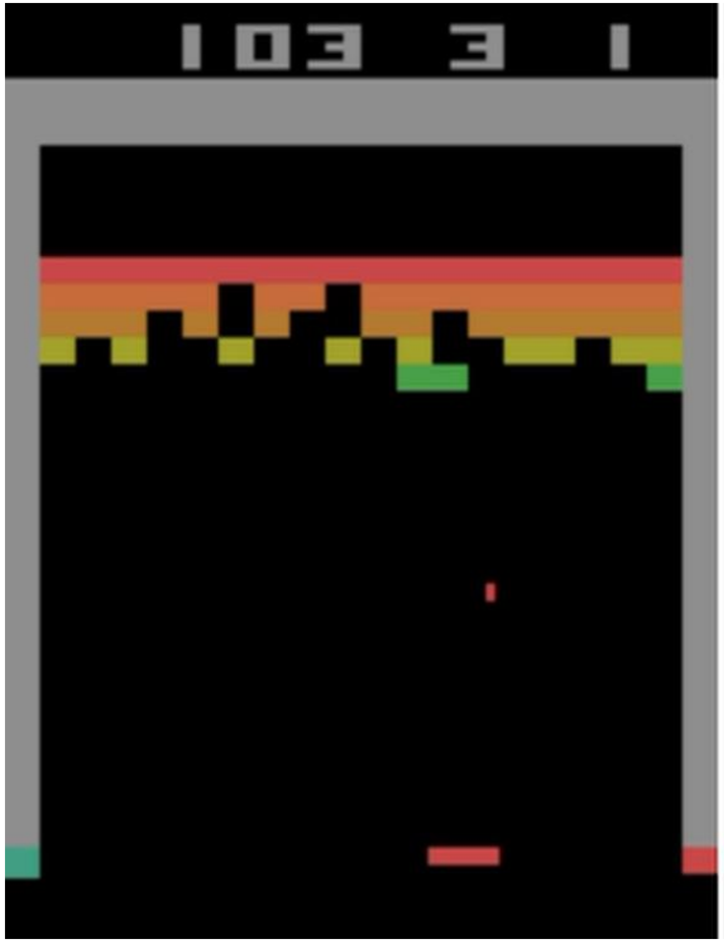
- MSE error가 -1과 1 사이일 때는 quadratic, 다른 곳은 linear

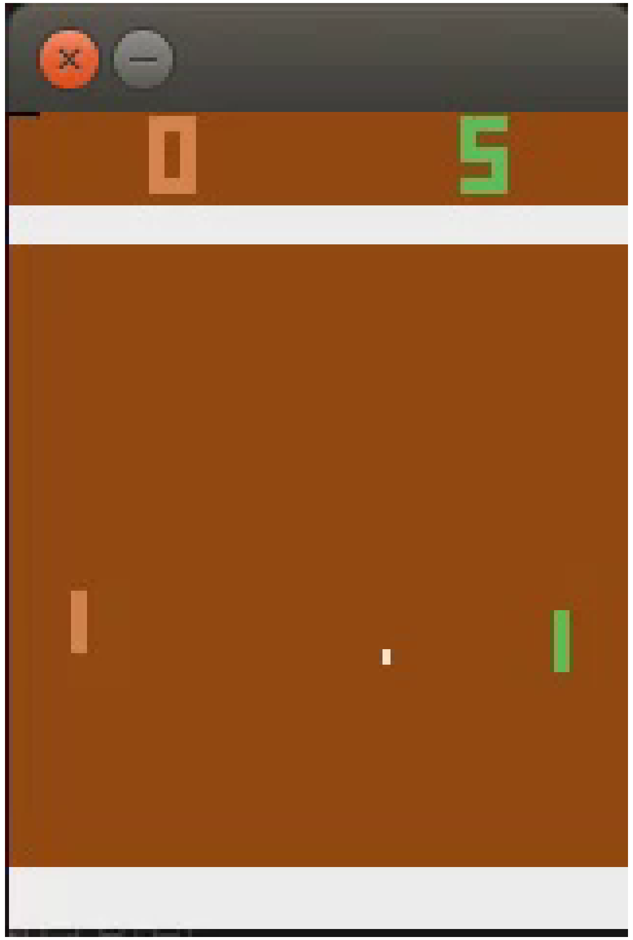


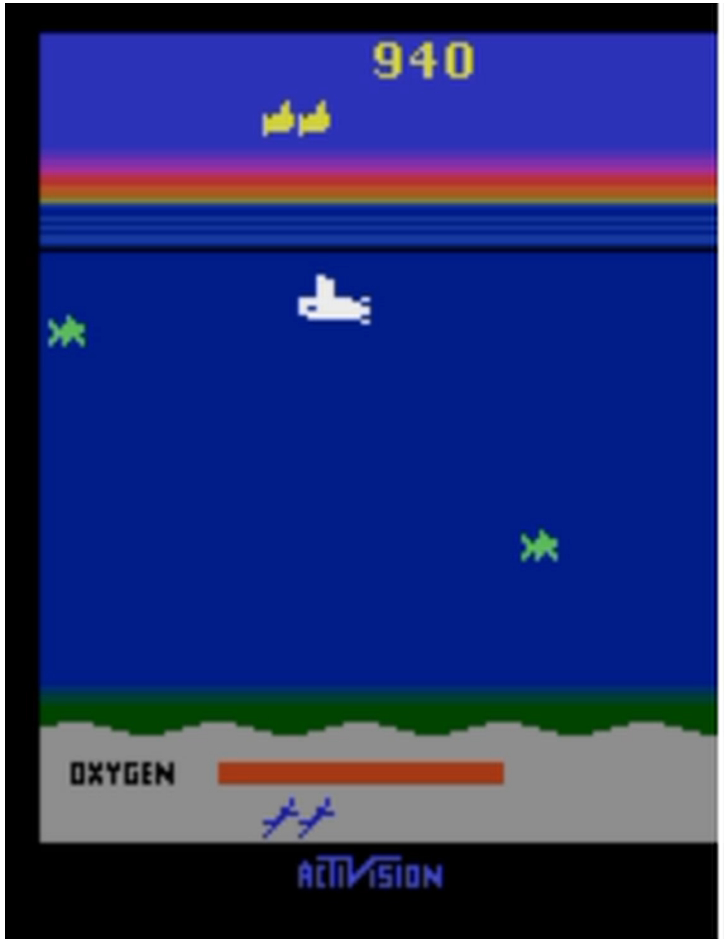
DQN의 세부사항

1. 환경 초기화 및 30 no-op
2. History에 따라 행동을 선택 ($\varepsilon - greedy$), ε값 decay
3. 선택한 행동으로 1 time-step 환경에서 진행, 다음 상태, 보상을 받음
4. 샘플을 형성 (h, a, r, h'), replay memory에 append
5. 50000 스텝 이상일 경우 replay memory에서 mini-batch 추출 후 학습
   - $MSE\ error$: $\left(r + \gamma \max_{a'} q_{\theta^-}(s', a') - q_\theta(s, a)\right)^2$
6. 10000 스텝마다 target network 업데이트

DQN on Atari







Thank you