



Introduction to Reinforcement Learning

2021. 09. 28

김정재



What is Learning

Learning?

Learn through Experience

= interacting with one's environment

we will explore "learning from interaction"
as the **perspective of artificial intelligence researcher**

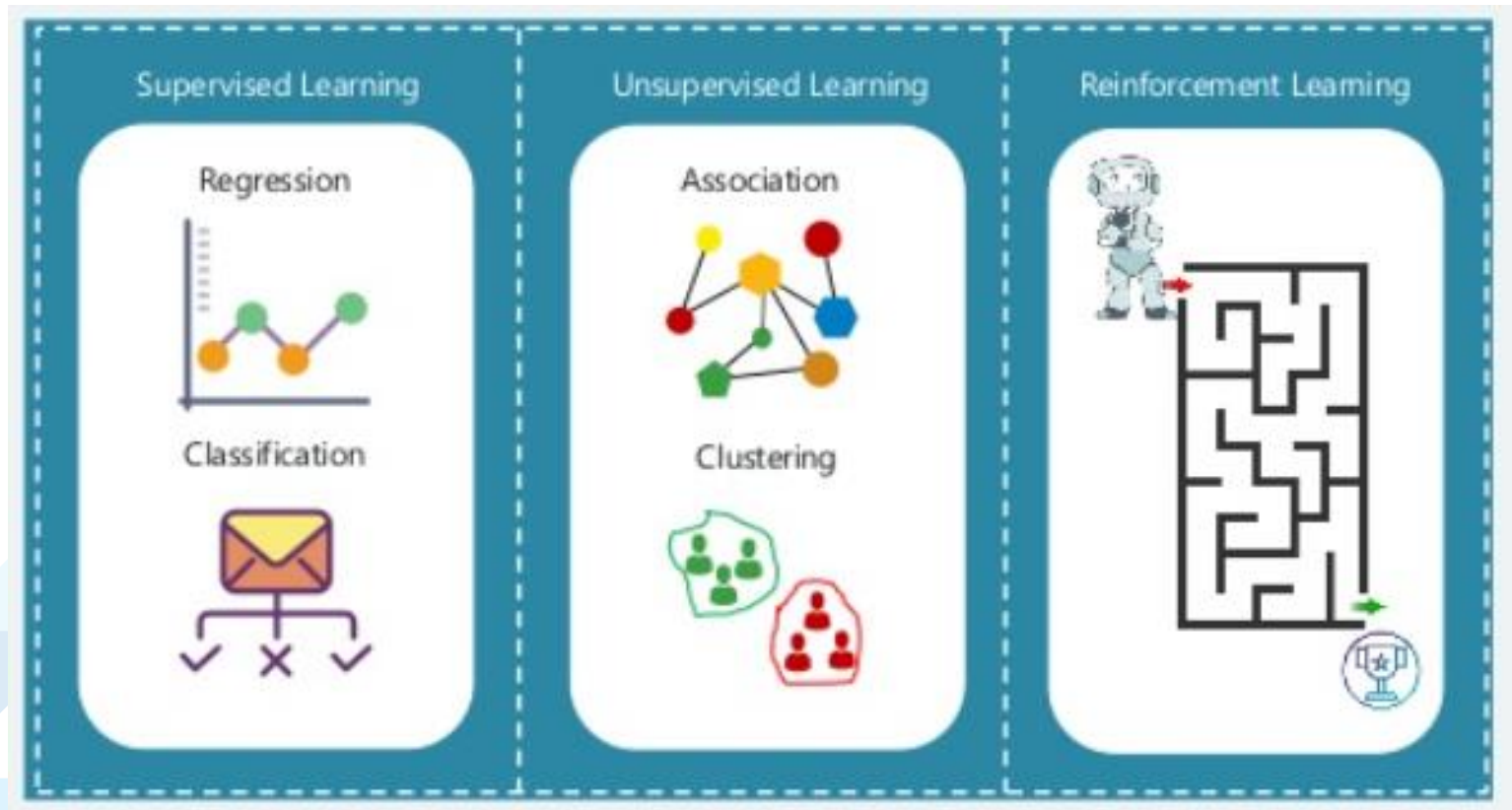
What is Learning



Basics of Reinforcement Learning



머신러닝의 학습방법



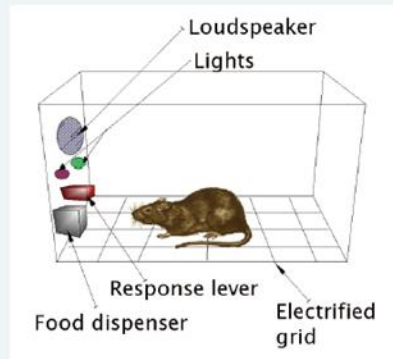
강화학습 기본

■ 환경과 상호작용하며 살아가는 인간과 동물

- 환경의 상태를 보고 자신에게 유리한 행동을 결정하고 실행
- 행동에 따른 결과가 좋으면 기억했다가 반복하고, 결과가 나쁘면 회피
 - 예) 자전거 타기
 - 예) 스키너의 행동심리학 실험
 - 예) 바둑, 비디오 게임, ...
- 행동 → 상태 변화 → 보상의 학습 사이클



(a) 어린이의 자전거 배우기



(b) 스키너 상자(출처: 위키백과)



(c) 학습 사이클

그림 9-1 '강화'를 통한 학습 과정

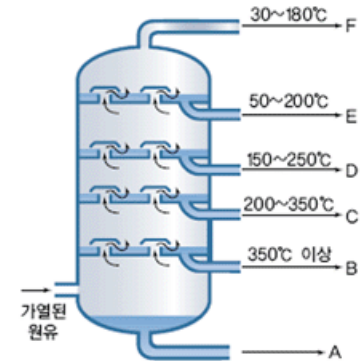
Learning in Real-time World



<chess play>



[정유 공장]



[정유탑의 내부 구조]

<petroleum refinery control>



<baby gazelle's running>



<prepare breakfast>

Real-World Learning in Common

- 모든 상황들이 에이전트와 환경과의 상호작용 (Interaction)을 포함한다.
- 에이전트의 행동이 미래 상태에 영향을 미친다.
- 에이전트는 환경의 불확실성에도 목표를 달성하려 한다.
- 에이전트는 성능을 향상시키기 위하여 이전의 경험을 활용할 수 있다.



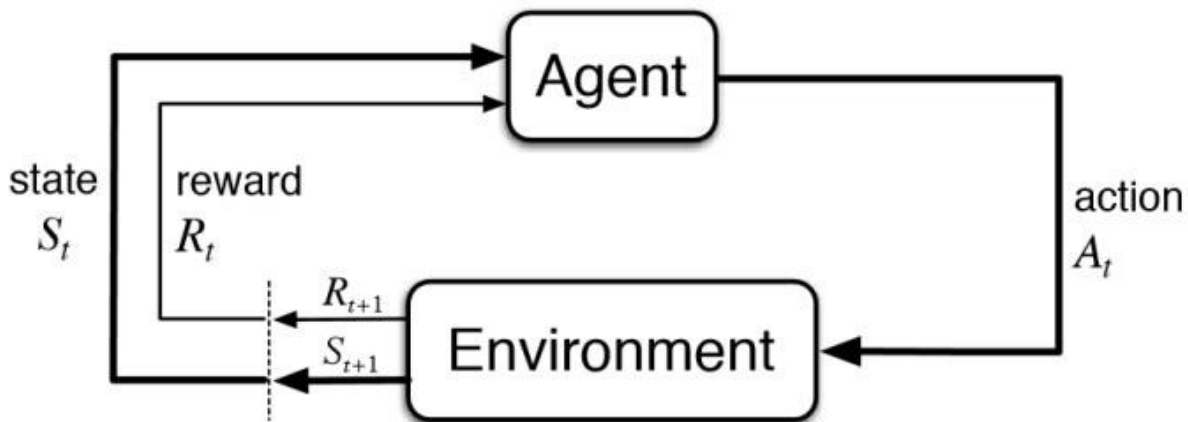
<chess play>



<prepare breakfast>

강화학습 기본

- Agent (에이전트)
 - 행동을 하는 주체
- Environment (환경)
 - 에이전트가 놓여진 곳
- Action (행동)
 - 행동이 환경에 하는 행위
- State (상태)
 - 환경에서 정량적으로 측정된 것
 - 에이전트의 행동에 의해 변화함
- Reward (보상)
 - 에이전트가 행동의 결과로 얻게 되는 것



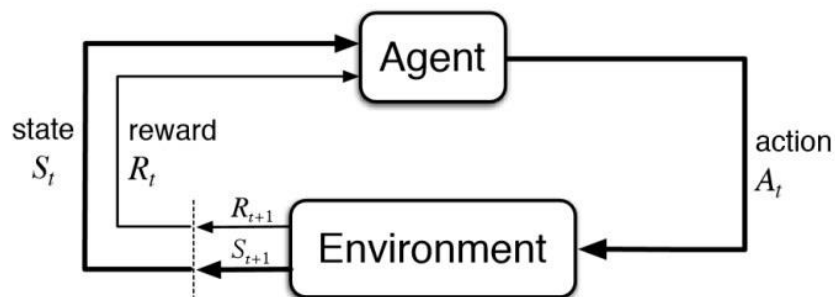
[강화학습 기본 Framework]

- '보상 (Reward)'를 극대화하기 위해 '해야할 일 (What-to-do)'를 학습함
- 행동은 즉각적인 보상을 포함하여 이후의 일련의 모든 보상에 영향을 미친다
- 에이전트는 반드시 상태를 센싱하고 액션을 취한다.
- 에이전트는 상태와 관련한 명확한 목표가 제시되어야 한다

강화학습 설계 예제



[강화학습 기본 Framework]



[강화학습 기본 Framework]

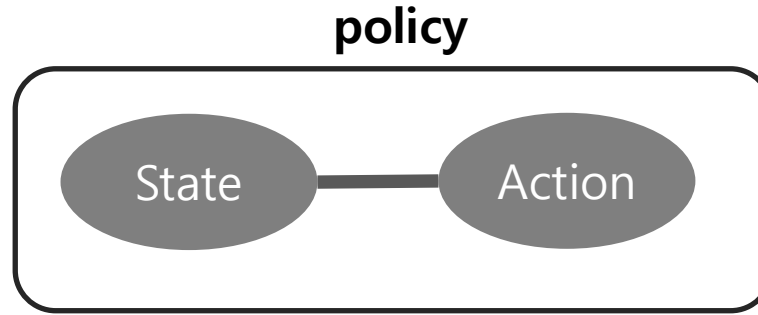
- 주식 거래 강화학습 에이전트 설계
 - Agent (에이전트) : 주식 거래 주체
 - Environment (환경) : 증권거래소
 - Action (행동) : Buy, Hold, Sell
 - State (상태) : 현재가, 전일종가 등등...
 - Reward (보상) : 수익 또는 수익률

Elements of Reinforcement Learning



Elements of Reinforcement Learning

1) Policy

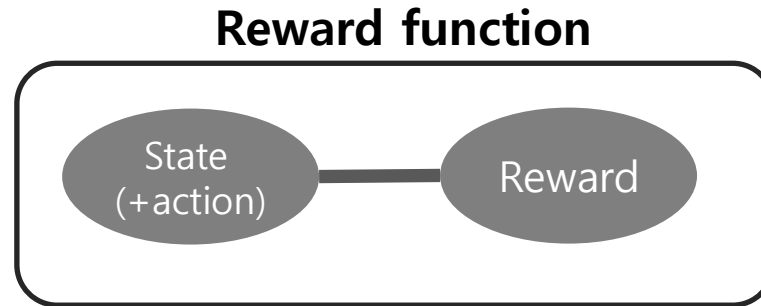


$$Policy = P(a|s)$$

- 에이전트의 행동 방식을 정의함
- 각각의 상태가 주어졌을 때, 에이전트가 취해야할 가장 좋은 행동에 매핑
- 강화학습의 핵심
- Policy는 확률적으로 정의할 수도 있음

Elements of Reinforcement Learning

2) Reward Function



- 에이전트의 목적을 정의함
- 일반적으로 싱글 넘버 형태로 표현
- 직관적으로 행동의 좋은 정도를 판단할 수 있음
- 전체 보상을 최대화하는 것이 에이전트의 목적

Elements of Reinforcement Learning

3) Value Function

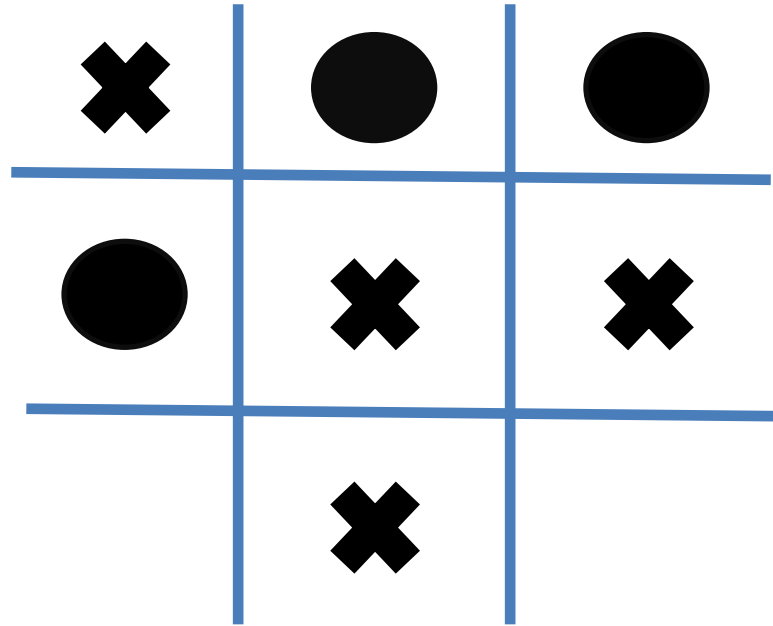
$$\text{value function } v(s) = \max_a (\text{Reward}_s + \text{Reward}_{s+1} + \dots + \text{Reward}_{goal})$$

- 장기적으로 좋은 행동을 선택할 수 있게 함
- 에이전트가 현재 상태에서 시작하여 앞으로 얻을 수 있을 것으로 기대되는 누적 보상
- 의사결정과 플래닝에 있어 중요함
최고의 가치를 가지는 행동을 추구하기 때문

An Extended Example : Tic-Tac-Toe



What is Tic-Tac-Toe?



Assume that

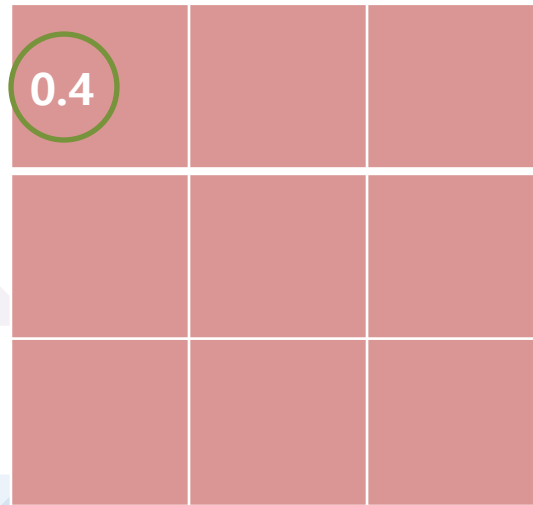
- We are playing against an imperfect player.
- Draws and losses to be equally bad for us.

What is Tic-Tac-Toe?

How do we construct a learning agent?

An reinforcement learning method

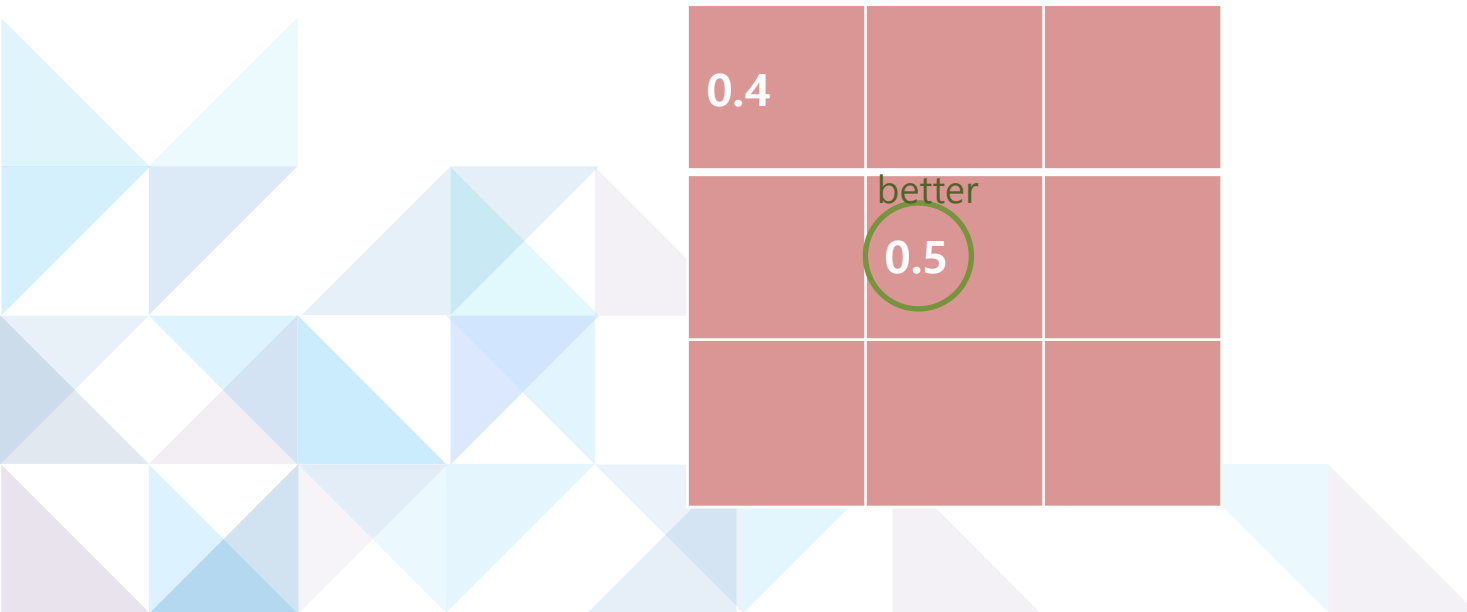
Probability of win from this state = value



What is Tic-Tac-Toe?

How do we construct a learning agent?

An reinforcement learning method




What is Tic-Tac-Toe?


How do we construct a learning agent?

An reinforcement learning method

Initialize all to 0.5



0.5	0.5	0.5
0.5	0.5	0.5
0.5	0.5	0.5

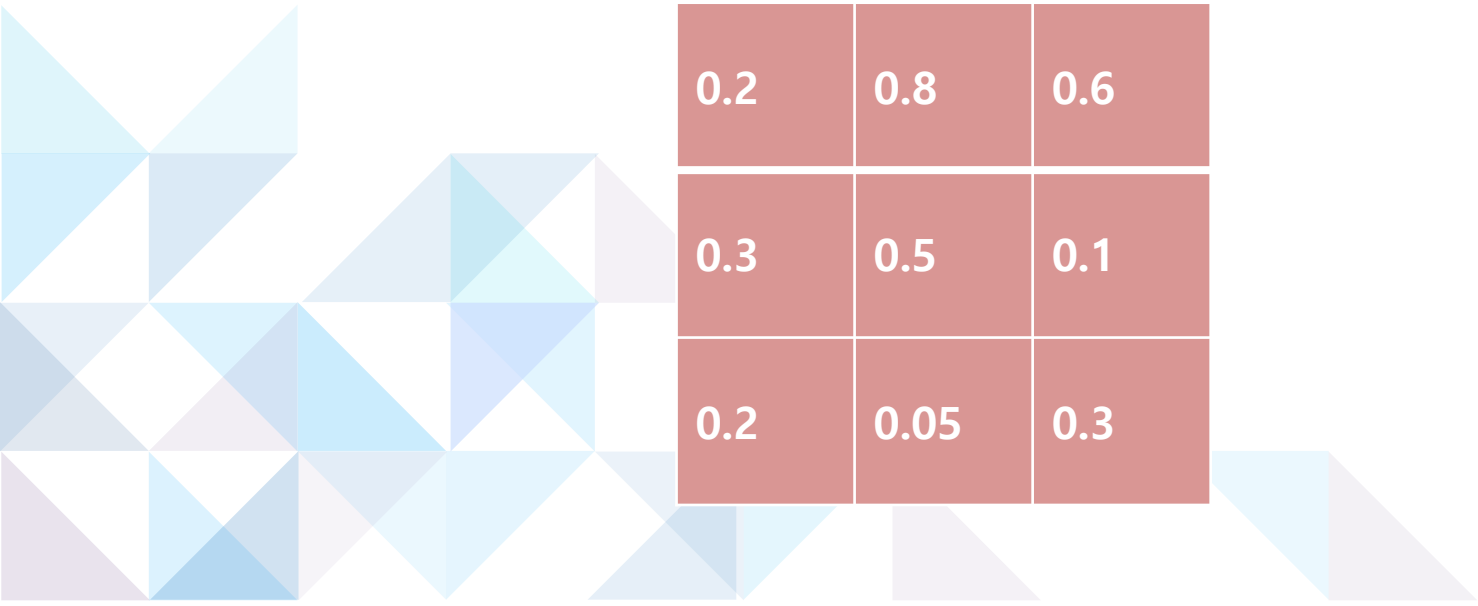


What is Tic-Tac-Toe?

How do we construct a learning agent?

An reinforcement learning method

After play many games against the opponent



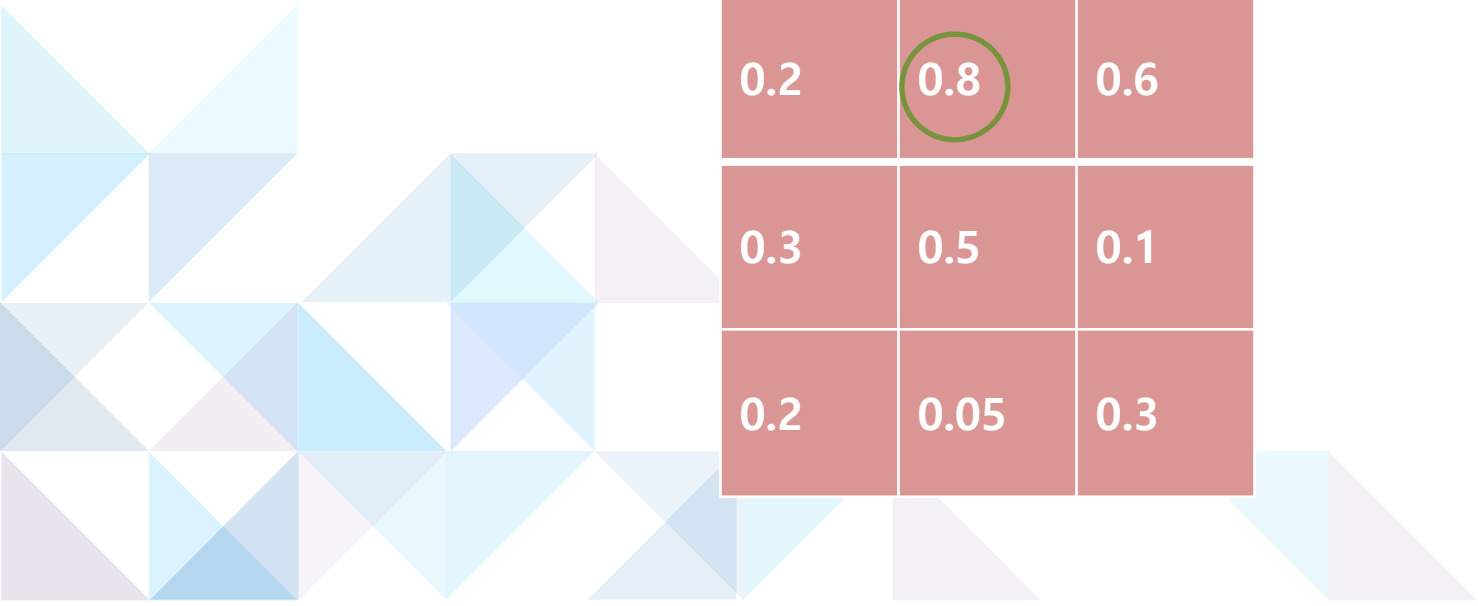
0.2	0.8	0.6
0.3	0.5	0.1
0.2	0.05	0.3

What is Tic-Tac-Toe?

How do we construct a learning agent?

An reinforcement learning method

Most of the time we move greedily



0.2	0.8	0.6
0.3	0.5	0.1
0.2	0.05	0.3

Deepmind DQN



- DeepMind Breakthrough 에이전트
 - Environment (환경) :
 - Action (행동) :
 - State (상태) :
 - Reward (보상) :

An n-Armed Bandit Problem



N-Armed Bandit Machine



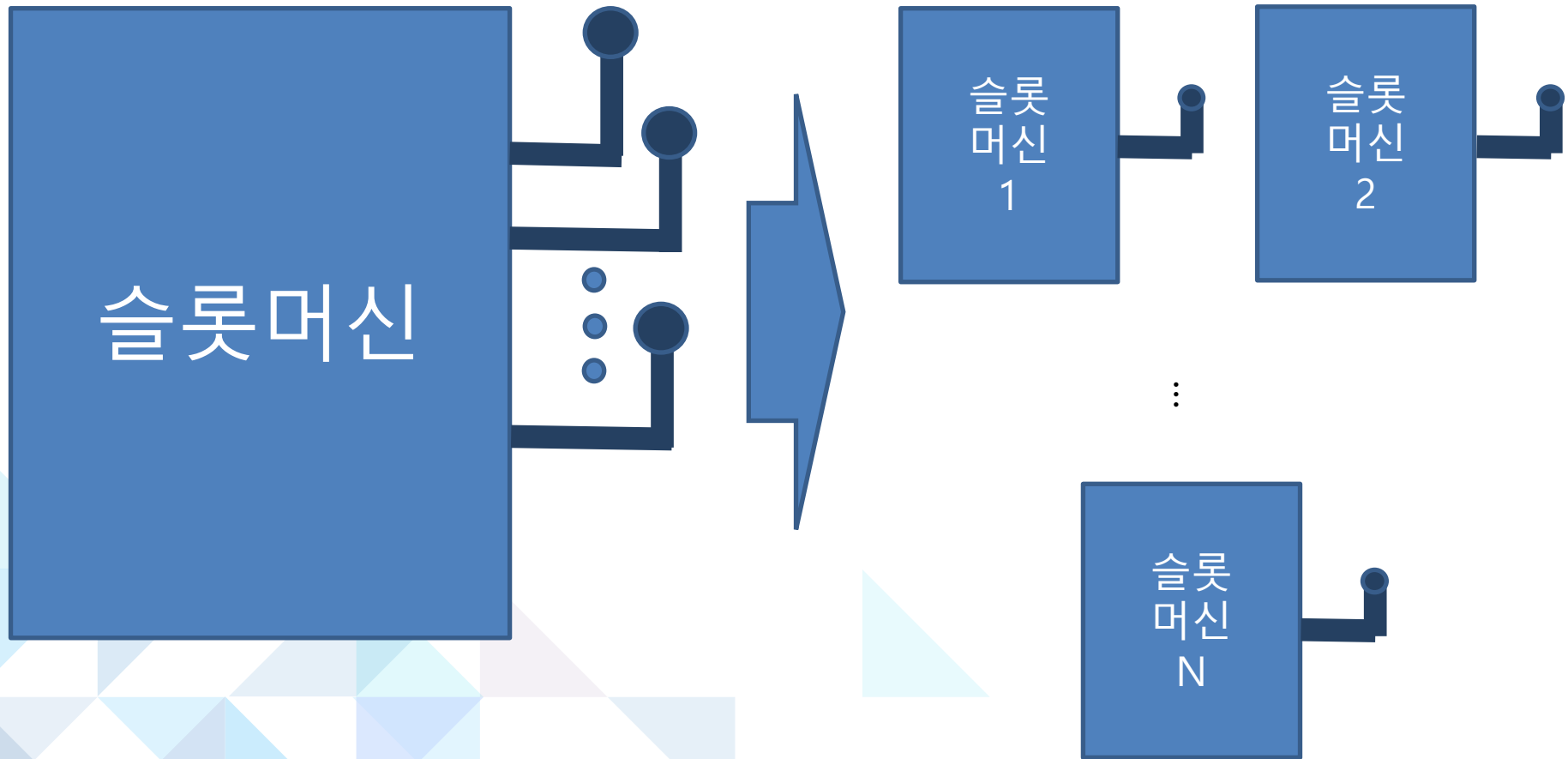
- N 개의 선택지(레버)중 매회 마다 하나의 선택지를 선택 할 수 있다.
- 선택지별로 다른 보상(코인)을 얻을 수 있고 이 보상은 선택지별 정해진 확률 분포를 따른다.
- 우리는 이 머신을 사용하여 최대의 보상을 얻고 싶다.

N-Armed Bandit Machine



- 레버를 한번 당길 때(Action) 얻을 것이라 추정되는 Reward의 평균값, 즉 레버를 당겼을 때 기대값을 **Value**라 한다.
- 각 레버의 true value값은 당연히 모른다고 가정한다.

N-Armed Bandit Machine



N-Armed Bandit Machine



- 각 Action들의 value값을 추정 하였을 때
 - 가장 Value값이 높은 레버를 선택하는 것(Greedy action) => Exploitation(활용)
 - Non-greedy action을 선택하는 것 : Exploration (탐험)

Why do Exploration?

Because of Uncertainty

우리가 추정한 Value가 정확히 맞다는 확신을 할 수가 없기 때문.



Exploration 과 Exploitation 간의 trade-off를 잘 조절 해야 한다.

But, 이 책에서는 최적의 trade-off까지는 고려하지 않는다.

문제에 대한 몇가지 Solution들을 알아본다.



Action-Value Methods

용어 정의

- 각 action a 의 true value 값을 $q_*(a)$ 라고 정의한다.
- Time step t 마다 추정되는 각 action a 의 value 값을 $Q_t(a)$ 라고 정의한다.
- t 번째 time step에서 action a 가 선택된 횟수를 K_a 라고 정의하고 선택 될 때 마다 얻은 reward를 R_1, R_2, \dots, R_{K_a} 라고 정의한다.
- 그때 $Q_t(a)$ 값은 다음과 같이 정의된다.

$$Q_t(a) = \frac{R_1 + R_2 + \dots + R_{K_a}}{K_a}$$

Action-Value Methods

가장 간단한 방법

모험을 하지 말자!

$$Q_t(A_t^*) = \max_a Q_t(a)$$

단, $A_t^* = \text{greedy action at time step } t$

Action-Value Methods

가장 간단한 방법

Time step 1



$$q_*(a) = 5, Q_t(a) = 0$$

$$q_*(a) = 1, Q_t(a) = 0$$

\vdots

$$q_*(a) = 2, Q_t(a) = 0$$

Action-Value Methods

가장 간단한 방법

Time step 2



$$q_*(a) = 5, Q_t(a) = 0$$

$$q_*(a) = 1, Q_t(a) = 1$$

$$\vdots$$

$$q_*(a) = 2, Q_t(a) = 0$$

Action-Value Methods

가장 간단한 방법

Time step ∞



$$q_*(a) = 5, Q_t(a) = 0$$

$$q_*(a) = 1, Q_t(a) = 1$$

\vdots

$$q_*(a) = 2, Q_t(a) = 0$$



모든 레버를 검사 할 수 없다.

Action-Value Methods

가끔은 Exploration 해야하지 않을까?

가끔 무작위로 선택하자!



ϵ - *greedy* Method

: ϵ 의 확률로 무작위 action을 선택

Action-Value Methods

가끔은 Exploration 해야하지 않을까?

Time step 1

where $\varepsilon = 0.1$



$$q_*(a) = 5, Q_t(a) = 0$$

$$q_*(a) = 1, Q_t(a) = 0$$

\vdots

$$q_*(a) = 2, Q_t(a) = 0$$

Action-Value Methods

가끔은 Exploration 해야하지 않을까?

Time step 2

where $\varepsilon = 0.1$



$$q_*(a) = 5, Q_t(a) = 0$$

$$q_*(a) = 1, Q_t(a) = 1$$

\vdots

$$q_*(a) = 2, Q_t(a) = 0$$

Action-Value Methods

가끔은 Exploration 해야하지 않을까?

Time step 10

where $\varepsilon = 0.1$



$$q_*(a) = 5, Q_t(a) = 5$$

$$q_*(a) = 1, Q_t(a) = 1$$

$$\vdots$$

$$q_*(a) = 2, Q_t(a) = 0$$

Action-Value Methods

가끔은 Exploration 해야하지 않을까?

Time step ∞

where $\varepsilon = 0.1$



$$q_*(a) = 5, Q_t(a) = 4.5$$

$$q_*(a) = 1, Q_t(a) = 1$$

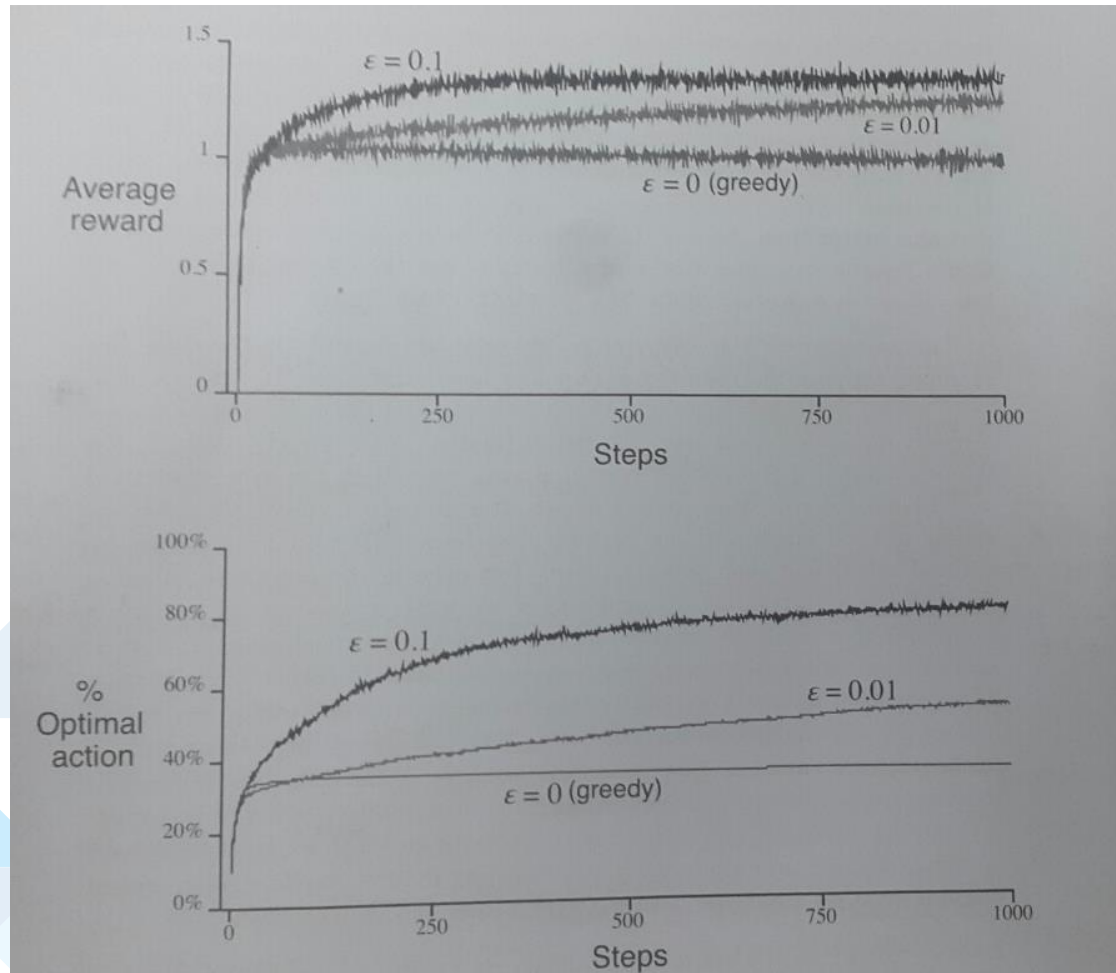
\vdots

$$q_*(a) = 2, Q_t(a) = 2$$

모든 레버의 q값을 알아낼 수 있다.

Action-Value Methods

가끔은 Exploration 해야하지 않을까? (실험결과)



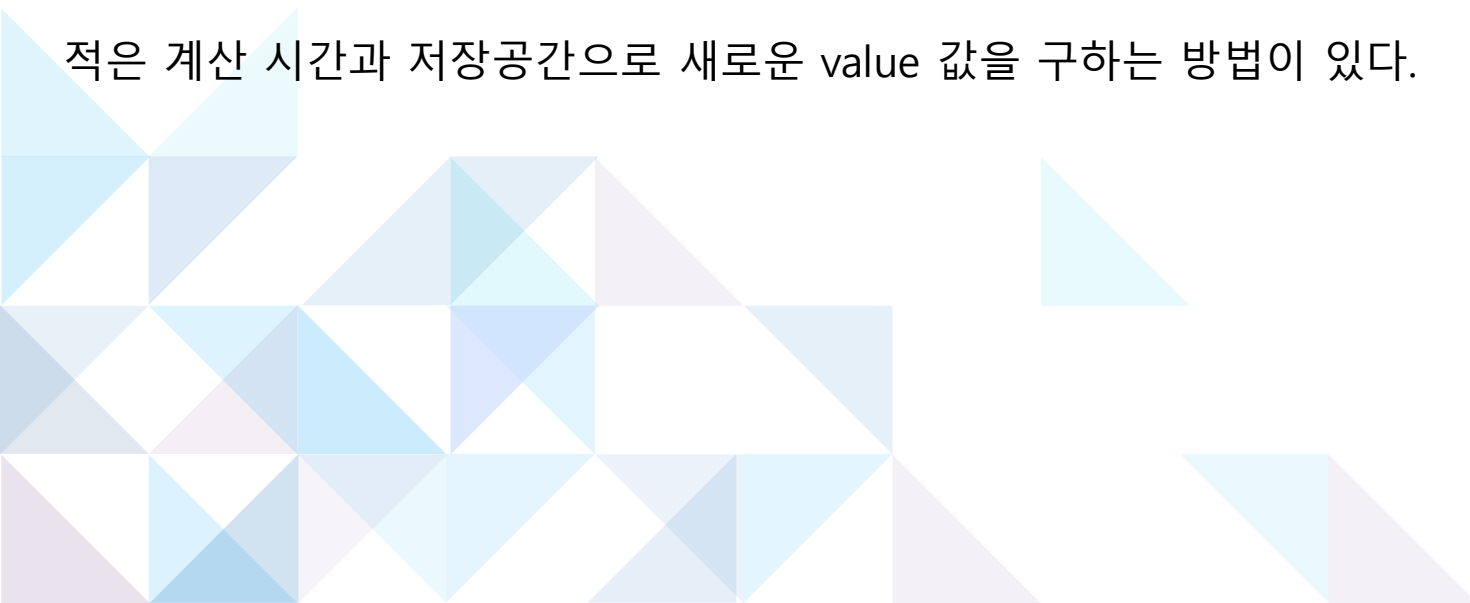
Incremental Implementation

문제점 : 점점 늘어나는 시간 및 공간복잡도 요구량

Tiem step이 늘어날 수록 아래의 식을 계산하는데 걸리는 시간과 저장공간이 점점 늘어난다.

$$Q_t(a) = \frac{R_1 + R_2 + \dots + R_{K_a}}{K_a}$$

적은 계산 시간과 저장공간으로 새로운 value 값을 구하는 방법이 있다.



Incremental Implementation

문제점 : 점점 늘어나는 시간 및 공간복잡도 요구량

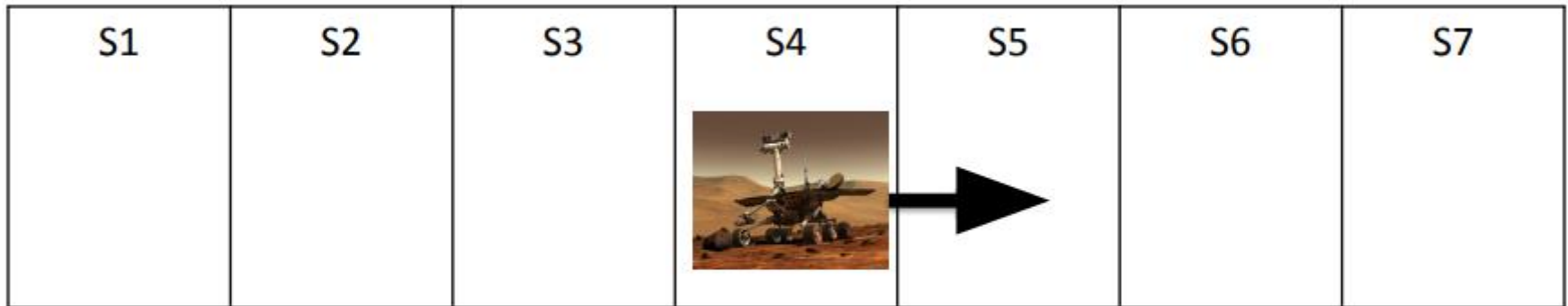
k 번째 value값을 Q_k 라고 하면

$$\begin{aligned}Q_{k+1} &= \frac{1}{k} \sum_{i=1}^k R_i \\&= \frac{1}{k} \left(R_k + \sum_{i=1}^k R_i \right) \\&= \frac{1}{k} (R_k + (k-1)Q_k) \\&= \frac{1}{k} (R_k + kQ_k + Q_k) \\&= Q_k + \frac{1}{k} [R_k - Q_k]\end{aligned}$$

➡ Q_k, k 의 값을 저장 할 기억공간만 있으면 Q_{k+1} 을 구할 수 있다.

Simple Mars Rover

Example: Simple Mars Rover



- 7 discrete states (location of rover)
- 2 actions: TryLeft or TryRight

실습

Simple Mars Rover

문제점 : 최종 Goal 까지 행동 횟수가 많음

```
action_cnt : 10  
cur_state : 2  
next_state : 1
```

```
action_cnt : 11  
cur_state : 1  
next_state : 0
```

```
action_cnt : 12  
cur_state : 0  
next_state : -1
```

```
left : [0, 0, 0.0, 0.0, 0.0, 0.0, 8.3222784]  
right : [3.3616, 0.0, 0.0, 0.0, 0.0, 0, 0]
```



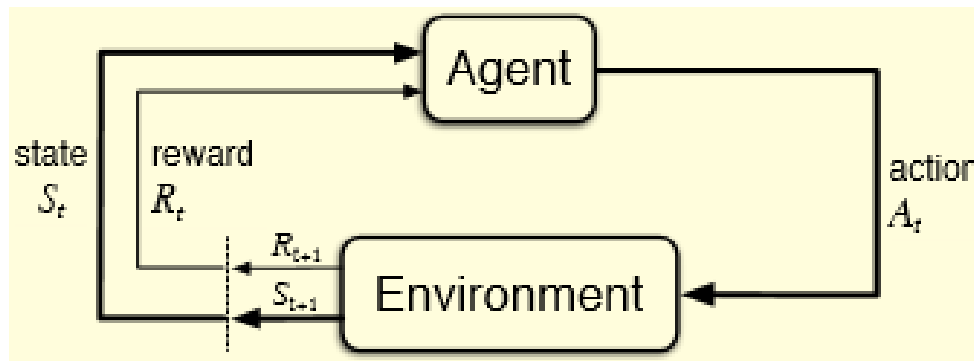
중간 과정의 Value 업데이트가 없음

행동의 순서가 Value에서 고려되지 않음

→ Policy의 변경이 없음

Reinforcement Learning

- 강화학습이란 "상호작용을 통해 목적을 이루는 학습문제"를 간단히 framing 한 것이다.
- 학습자(learner, decision-maker)를 **Agent**라고 부른다.
- Agent와 상호작용할 수 있는 (Agent외의) 모든 것을 **Environment**라 부른다.
- Agent와 Environment는 끊임없이 상호작용한다.
- 좀 더 자세히 말하면, 둘은 time-step t 마다 상호작용을 한다.
매 t 마다 Agent는 Environment의 **state** $S_t \in \mathcal{S}$ (단 \mathcal{S} 는 사용 가능한 state의 집합)를 받고,
그에 따라서 **action** $A_t \in \mathcal{A}(S_t)$ 를 선택한다.
한번의 t 가 끝나고 action에 대한 결과로써 numerical reward $R_{t+1} \in \mathbb{R}$ 를 얻는다.
그리고 새로운 S_{t+1} 를 찾는 것을 반복.



Reinforcement Learning

- 각 t 마다, Agent는 S_t 에서 각 A_t 들이 선택될 확률을 맵핑 시키는데 이를 **Policy** π_t 라 부른다. 단, $\pi_t(a|s)$ 는 $S_t = s, A_t = a$ 일 확률을 나타냄
- 강화학습 method는 Agent가 경험을 하며 Policy를 어떻게 수정해 나갈 것인지 구체적으로 명시한다.
- Agent의 goal은 자신이 얻을 수 있는 **reward의 총량을 최대화** 시키는 것이다.

강화학습의 최대 이슈

Policy를 수정하여 Agent가 가질 수 있는 reward의 총량을 최대화 시키는 것

How?

Policy를 어떻게 수정 할까?

Sequential Returns



Returns

Goal을 어떻게 식으로써 나타낼까?

- time step t 이후로 얻는 reward들을 $R_{t+1}, R_{t+2}, R_{t+3}, \dots$ 이라고 둔다.
- t 이후 얻을 것이라 기대되는 결과값(return)을 G_t 라 둔다. G_t 는 reward집합에 대한 함수
- 이때 G_t 를 Maximize하는 것이 Goal을 정의하는 것이다.

Then

가장 간단한 형태의 G_t 는 다음과 같다.

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

(where T : final time step)

-Terminal state가 존재

-모든 Nonterminal states들의 집합은 \mathcal{S} , terminal state까지 포함한 집합은 \mathcal{S}^+ 라 부른다.



Episodic task

Returns

Goal을 어떻게 식으로써 나타낼까?

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

(where T : final time step)

- 문제점 : Agent가 첫 행동에서 받는 보상과 끝 행동을 통해 받는 보상을 구분하기 어려움
- 문제점 : 해결책 Discounting Factor (γ) 도입

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(where $0 \leq \gamma < 1$)

0 ← γ → 1

myopic

farsighted

Markov Decision Process



Markov Property

State라는 signal이 필요한 이유에 대해 알아본다.

- Agent가 결정을 내릴 때 쓰이는 environment쪽의 signal을 **state**라 한다.
- state를 구성하고 수정하고 학습하는 것 등에 대해서는 다루지 않는다.
어떤 state에서도 작동하는 함수를 통해 action을 결정하는 것에 중점을 둔다.
- state는 일시적인 sensing으로 구성될 수도 있다. 반대로 state가 과거의 모든 유용한 정보들을 알고있을 필요는 없다.
- 하지만 과거의 **모든 유용한 정보들을 알 수 있으면 좋다.**
- 과거의 모든 유용한 정보들을 포함한 채 계승 되는 State signal을 **'Markov하다'** 또는 **Markov Property**를 가진다고 칭한다.

Markov Property

Markov Property를 형식화 한다.

- 강화학습 문제에서 일반적으로, time step t 에서 수행한 action에 대한 응답으로 $t+1$ 에서의 state, reward S_{t+1}, R_{t+1} 가 정해질 때,
이들은 이전의 일어났던 모든 일들 $R_t, S_t, A_t, R_{t-1}, S_{t-1}, A_{t-1}, \dots, R_1, S_0, A_0$ 에 의존한다.

$$\Pr\{S_{t+1} = s', R_{t+1} = r \mid R_t, S_t, A_t, R_{t-1}, S_{t-1}, A_{t-1}, \dots, R_1, S_0, A_0\} \quad \dots\dots(\text{식 1})$$

하지만, state signal이 **Markov Property**를 가진다면

$$\Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t, A_t\} \quad \text{과 같이 표현 될 수 있고} \quad \dots\dots(\text{식 2})$$

이를 **Markov state**라 부른다.

그리고 (식 1)과 (식 2)를 동시에 만족하는 것이 Markov state의 **필요충분조건**이다.

Markov state을 통해 최고의 action을 선택하는 것은
지금까지의 history를 통해 최고의 action을 선택하는 것과 같다.

$$\pi(S_t) == \pi(R_t, S_t, R_{t-1}, S_{t-1}, A_{t-1}, \dots, R_1, S_0, A_0)$$

Markov Property

Markov State로 근사화 한다.

- Non-Markov한 State라도 Markov한 상태라고 근사화 시켜 사용한다.

Ex) 자동차는 엔진의 힘에 의해서만 위치가 바뀐다

Why?

Markov State는 Reward 예측, Action 선택에 언제나 좋은 **기준**이 되어 주기 때문

Markov Property

Exercise

● exercise 3.6 : Broken Vision System

자신이 vision system이라 생각해보자.

처음 눈을 뜨고 하루 동안 보이는 이미지들을 camera에 담는다. 단 벽 뒤에 가려져있는 것 등은 못본다.

첫번째 scene을 본 후 당신은 environment의 Markov state를 구할 수 있는가? ...(1)

그 날 이후 camera가 부서져서 앞으로 image들을 얻지못하게 되었다고 가정하자.

그때는 Markov state를 구할 수 있는가? ...(2)



Markov Decision Process

정의

- Markov Property를 만족하는 강화 학습 task를 **Markov decision process** 또는 **MDP**라 부른다. 특히, state, action의 개수가 유한하면 **Finite MDP**라 부른다.
- Finite MDP는 현재 state s 와 action a , 다음 state s' , 그리고 한 step의 dynamics에 의해 정의된다.
- s, a 가 주어졌을 때 s' 로의 전이 확률 $p(s'|s, a)$ 은

$$p(s'|s, a) = \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\}$$

s, a, s' 가 주어졌을 때 reward $r(s, a, s')$ 은

$$r(s, a, s') = \mathbb{E}\{R_{t+1} \mid S_t = s, A_t = a, S_t = s'\} \quad \text{이고,}$$

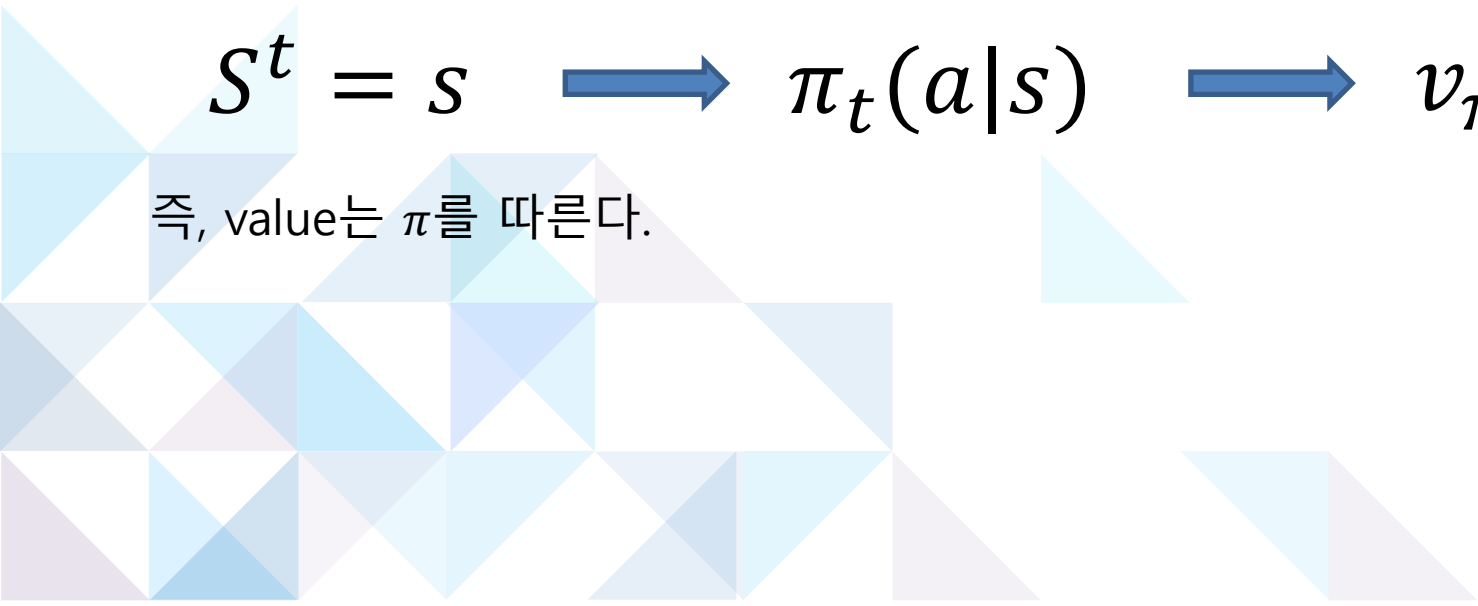
이 두가지 수치값은 Finite MDP의 dynamics를 아주 잘 표현할 수 있다.

Bellman Equation



Value Functions

Value는 얻을 수 있는 Reward의 총량을 예측한 것


$$S^t = s \quad \longrightarrow \quad \pi_t(a|s) \quad \longrightarrow \quad v_{\pi_t}(s)$$

즉, value는 π 를 따른다.

Value Functions

Value는 얻을 수 있는 Reward의 총량을 예측한 것

Then

State-Value Function

$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+K+1} \mid S_t = s \right]$$

Action-Value Function

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+K+1} \mid S_t = s, A_t = a \right]$$

$v_{\pi}(s)$ 와 $q_{\pi}(s, a)$ 는 여러 번의 experience을 통해 얻어진 Return들의 평균값을 각각 자신의 값으로 삼는다.
이렇게 실험을 반복하여 값을 추정하는 방법을 Monte Carlo methods라 한다.

Value Functions

Value Function은 재귀적인 성질을 가진다.

$$\begin{aligned}v_{\pi}(s) &= \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+K+1} \mid S_t = s\right] \\&= \mathbb{E}_{\pi}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+K+2} \mid S_t = s\right] \\&= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+K+2} \mid S_{t+1} = s'\right] \right] \\&= \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v_{\pi}(s')]\end{aligned}$$

where $(s \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R})$



Bellman equation for v_{π}

Value Functions

Value는 얻을 수 있는 Reward의 총량을 예측한 것

Then

State-Value Function

$$v_{\pi}(s) = \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+K+1} \mid S_t = s \right]$$

$$v_{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v_{\pi}(s')]$$

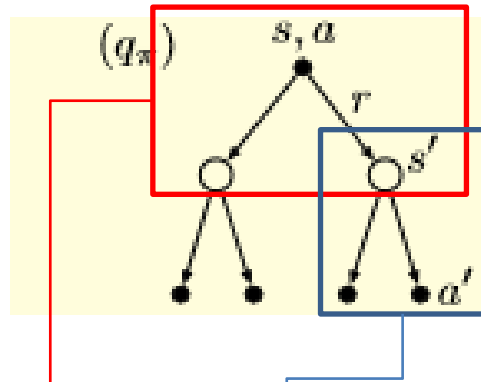
Action-Value Function

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+K+1} \mid S_t = s, A_t = a \right]$$

$$q_{\pi}(s, a) = \sum_{s'} p(s'|s, a) \sum_{a'} \pi(a'|s) [r(s, a, s') + \gamma q_{\pi}(s', a')]$$

Value Functions

- exercise 3.8 : action values의 Bellman equation은 무엇인가?



$$\begin{aligned} q_{\pi}(s, a) &= \sum_{s'} p(s'|s, a) \sum_{a'} \pi(a'|s) [r(s, a, s') + \gamma q_{\pi}(s', a')] \\ &= \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v_{\pi}(s')] \end{aligned}$$

Value Functions

- Optimal Policy π_* 를 찾는 것은 Optimal value function을 찾는 것과 같다.

$$\pi < \pi' , \text{ if and only if } v_\pi(s) < v_{\pi'}(s) \text{ and } q_\pi(s, a) < q_{\pi'}(s, a)$$

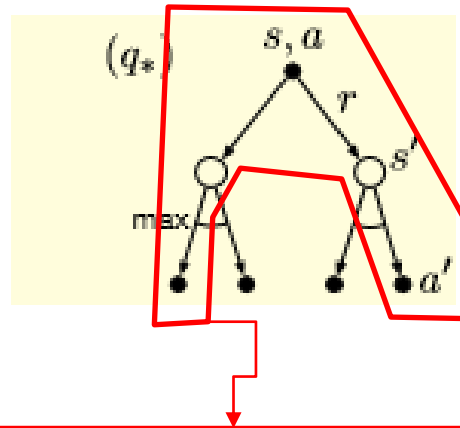
$$\text{So. } v_*(s) = \max_{\pi} v_\pi(s)$$

$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Optimal Value Functions

Bellman optimality equation

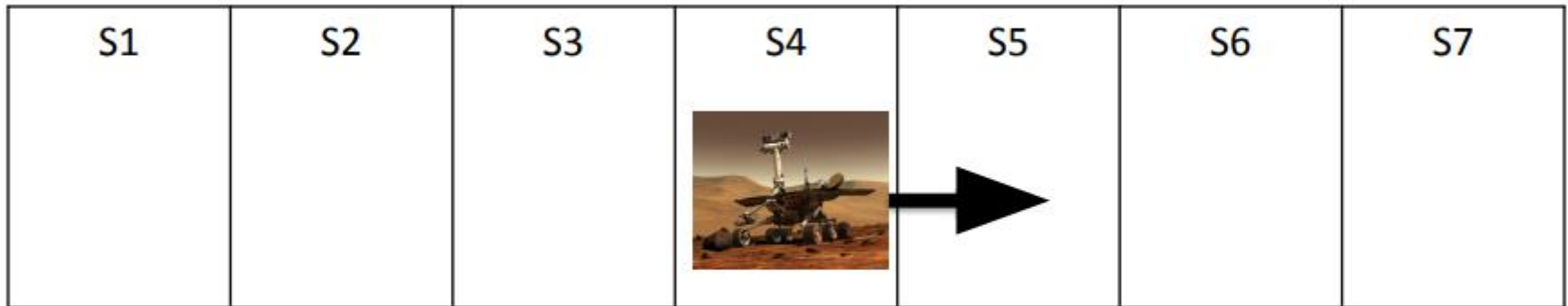


$$q_*(s, a) = \sum_{s'} p(s'|s, a) \left[r(s', a, s) + \gamma \max_{a'} q_*(s', a') \right]$$

- 한차례(immediate, local)의 결과만 보고 greedy하게 선택을 하는 모습이지만, 실제로는 장기적으로(long-term) 최적의 선택이 된다.
→ $q_*(s, a)$ 가 앞으로 일어날 모든 행동들의 reward 이기 때문.

Simple Mars Rover_MDP

Example: Simple Mars Rover



- 7 discrete states (location of rover)
- 2 actions: TryLeft or TryRight

실습

Simple Mars Rover

문제점 : Transition Probability를 알아야함

$$q_*(s, a) = \sum_{s'} \underline{p(s'|s, a)} \left[r(s', a, s) + \gamma \max_{a'} q_*(s', a') \right]$$

```
if (action - cur_state) == 1:  
    right_action_val[action] = 1 * (action_reward + (gamma * right_action_val[action + 1]))  
else:  
    left_action_val[action] = 1 * (action_reward + (gamma * left_action_val[action - 1]))
```

➡ State Transition Probability가 없으면 계산이 불가능
환경의 Dynamics를 이미 알고 있음
→ 불가능 (강화학습 할 필요가 없음)

Summary

- 강화학습은 **interaction**을 통해 goal을 성취하는 방법을 배우는 학습법이다.
- 강화학습 **Agent와 Environment**가 매 time-step마다 interaction한다.
- **action**은 agent가 선택하는 것, **state**는 선택을 하는 기준, **reward**는 선택을 평가하는 기준이다.
- **policy**는 agent가 현재 state에 따라 사용할 수 있는 action중 하나를 선택하는 확률 함수다.
- Agent의 목적은 **최종으로 얻을 reward의 총량을 최대화** 하는 것이다.
- **return**은 agent가 최대화 하고자 하는 reward의 총량에 대한 기대값 함수이다.
- reward가 점차 discount되는, 끝나지 않는 task를 **continuing task**라 한다.
반대로 episode단위로 끊기는, 끝이 있는 task를 **episodic task**라 한다.



감사합니다

강의 내용 관련 질문 : kjj6929@gmail.com

