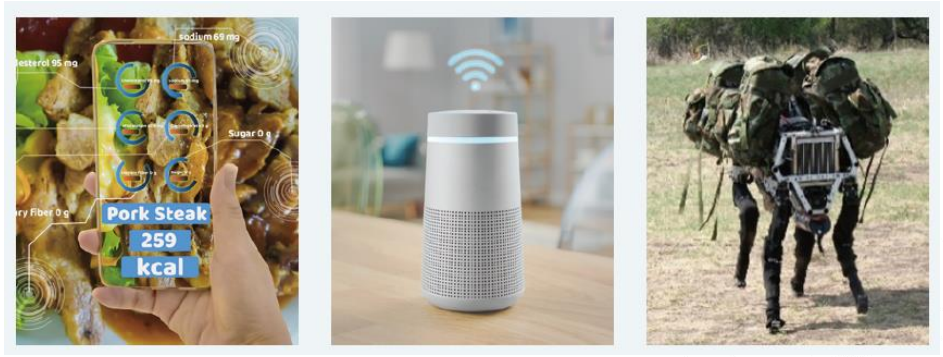
Intelligent Systems with RL

Intelligent Agents

- Cognitive behavior of humans and intelligent agents
  - Humans ***sense*** their surroundings → ***perceive*** → ***interact*** appropriately with the environment
  - AI products must also perform similar processes to be effective

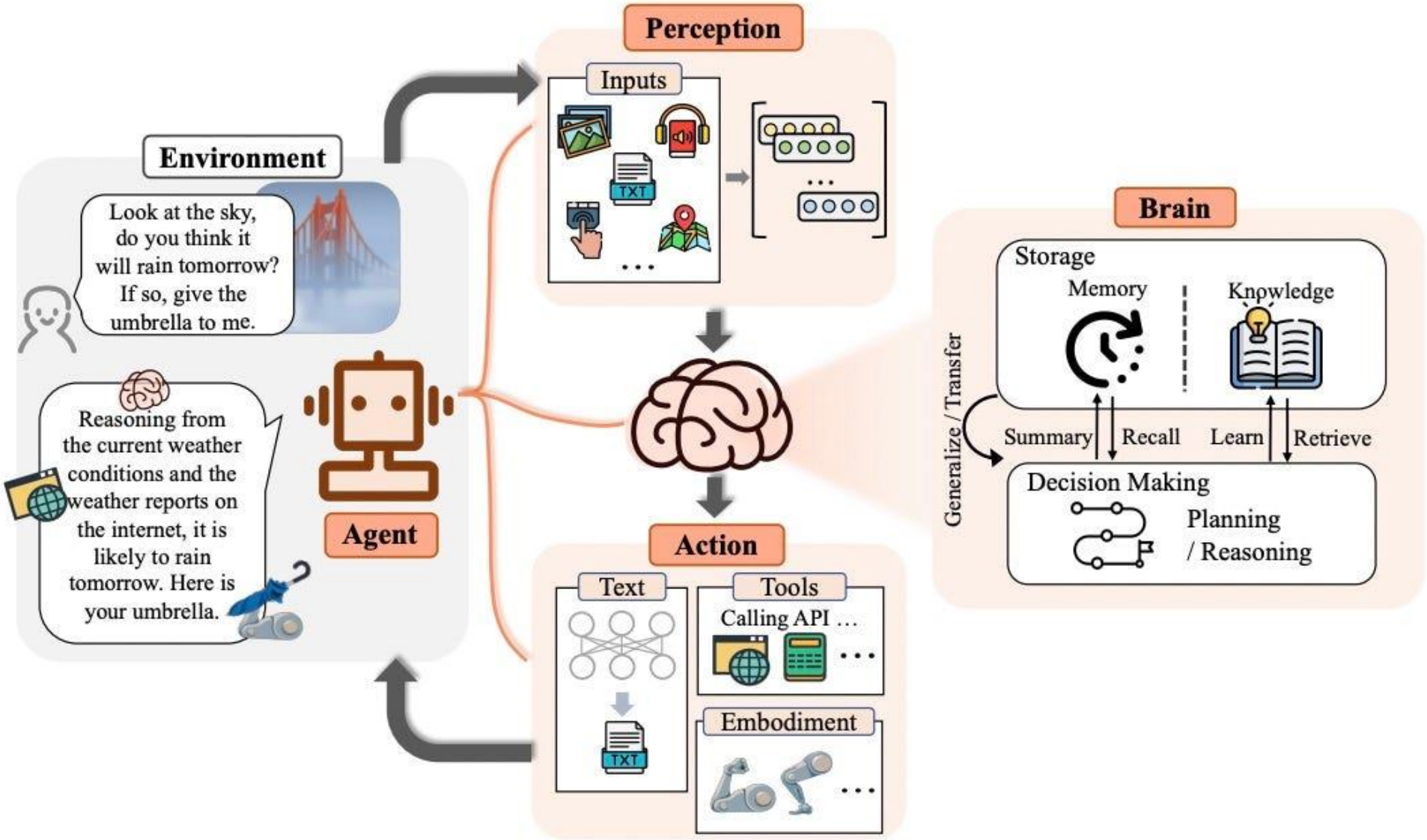


Intelligent Agents

- Agents in daily life
  - People who perform specific tasks in everyday life are referred to as agents
    - e.g., a travel agent organizes itineraries and books transportation and accommodations

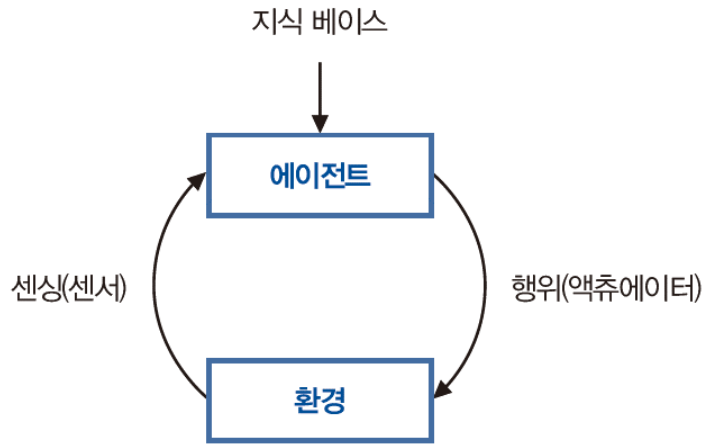
- Agents in computing
  - Software that performs tasks on behalf of people in computing is called an agent
    - Focus on intelligent agents that handle cognitive tasks for humans

Intelligent Agents



Intelligent Agents

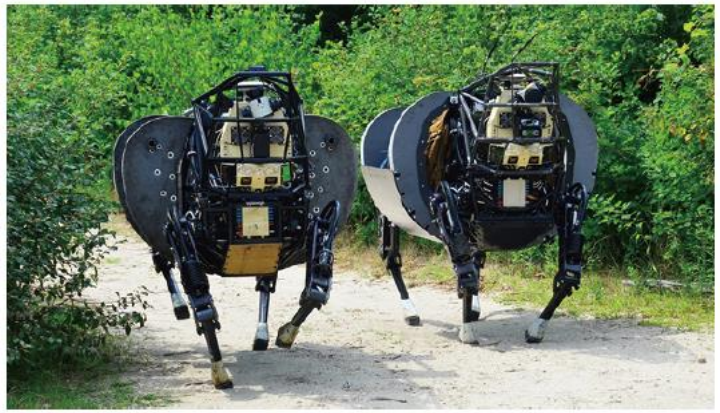
- Intelligent agents
  - Robots act as ***agents***, and the ***environment*** and ***knowledge*** base are clearly defined
    (The robot body and software are the agent, cameras and other sensors are the sensors, and robotic arms are actuators)
  - Voice recognition applications simulate this structure to some extent
    (Smartphones and apps are the ***agents***, cameras are the ***sensors***, and screens and speakers are the ***actuators***)
- Knowledge base
  - Frameworks, semantics, ontologies, and rule-based methods for knowledge representation



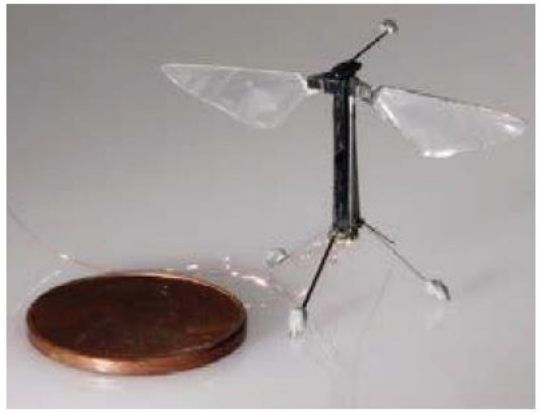
AI Robots

(a) Manipulator

(b) Cleaning robot



(c) BigDog



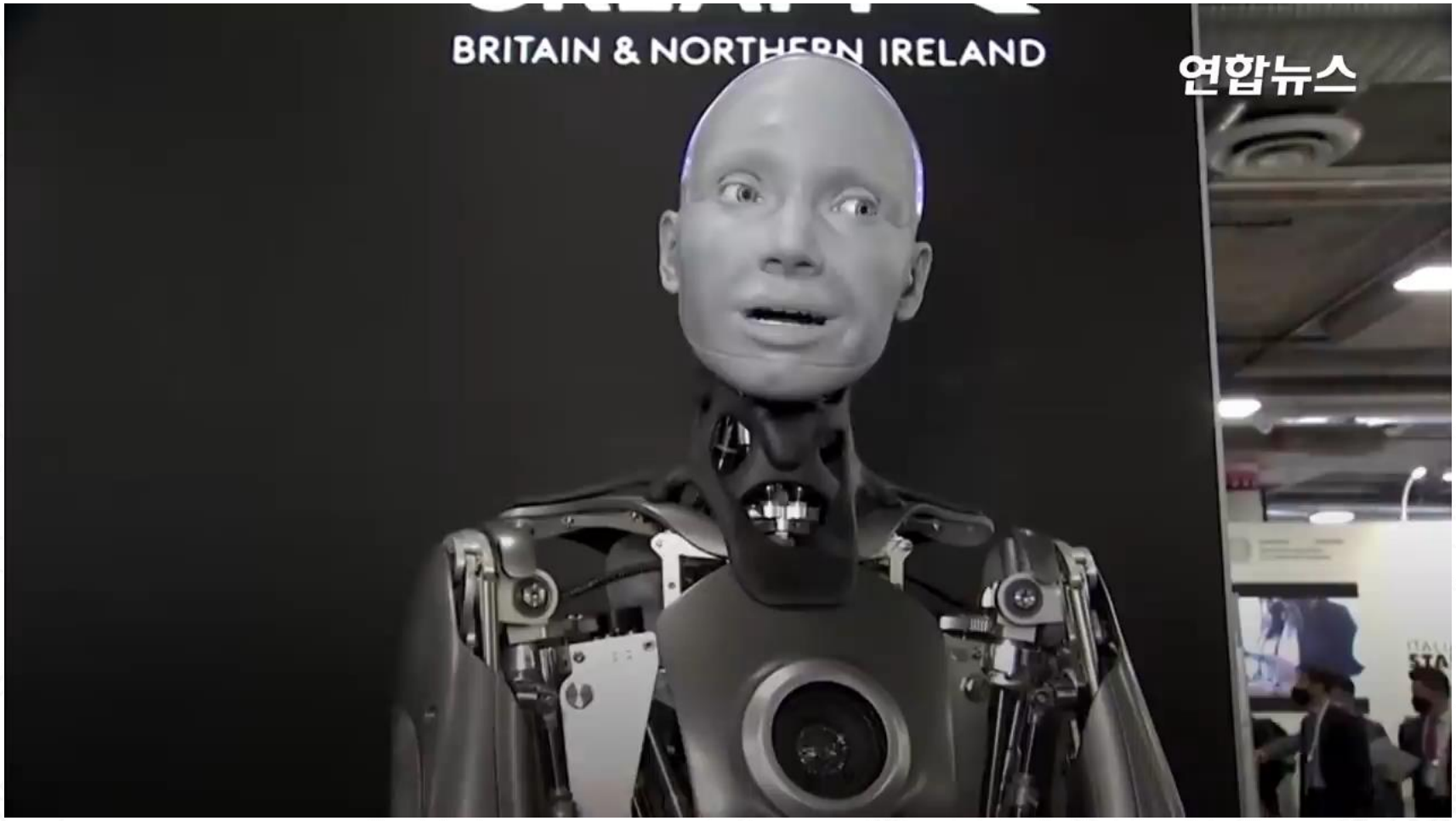
(d) Insect robot

(e) Humanoid

(f) Drone

(g) Autonomous vehicle

Humanoid AI Robot



Humanoid AI Robot

Humanoid AI Robot

인공지능(AI) 로봇 '소피아'

머리 부분 : 투명한 플라스틱 밑으로 전기회로가 보임
(가발까지 씌우면 인간과 구분이 어려워 일부러 머리를 드러냄)

62가지 **감정을 얼굴로 표현**

실시간으로 인간과 **대화 가능**

- 배우 오드리 헵번 얼굴을 참고로 제작
- 인간과 흡사한 외모
- 비교적 능숙한 대화 기술
- 지금까지 개발된 로봇 중 사람과 가장 유사하고 심층 학습 능력이 있어 사람과 대화할수록 더 수준 높은 문장을 구사

눈썹 : 찌푸리거나 눈을 깜빡이는 등 다양한 표정 구사

눈 : 3D 센서 장착
(대화 상대 인식)

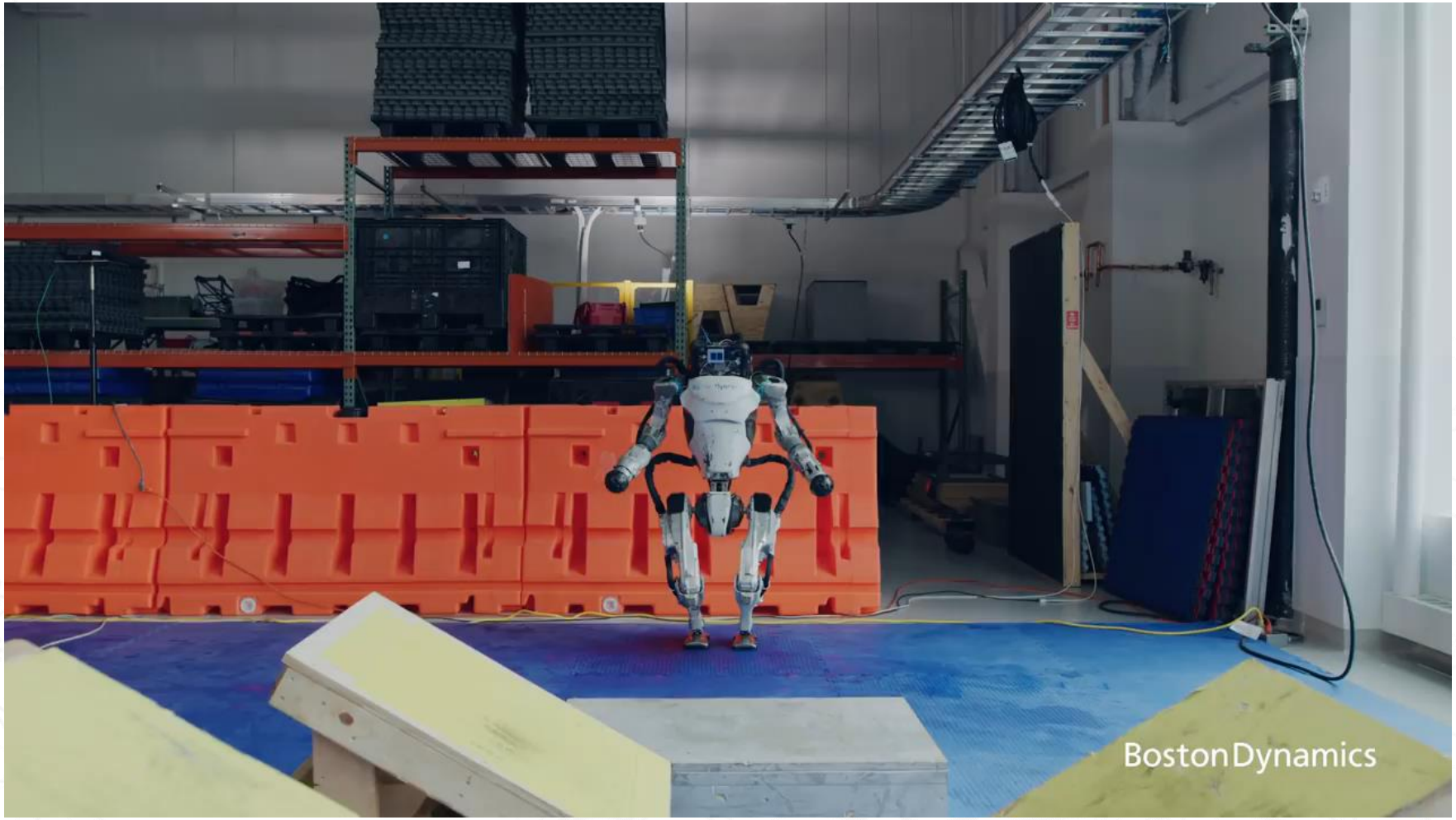
피부 :
'플러버(frubber)' 소재
(인간의 피부와 흡사)

- 2017년 10월 사우디아라비아에서 최초로 로봇으로서 시민권 발급, 같은 달 유엔 경제사회이사회(ECOSOC)에 패널로 등장

홍콩에 본사를 둔 **핸슨 로보틱스(Hanson Robotics)**가 개발한 인공지능(AI) 로봇
자기인식과 상상력을 갖는 등 인간 수준으로 진화시키겠다는 목표로 제작

연합뉴스

AI Industrial Robots

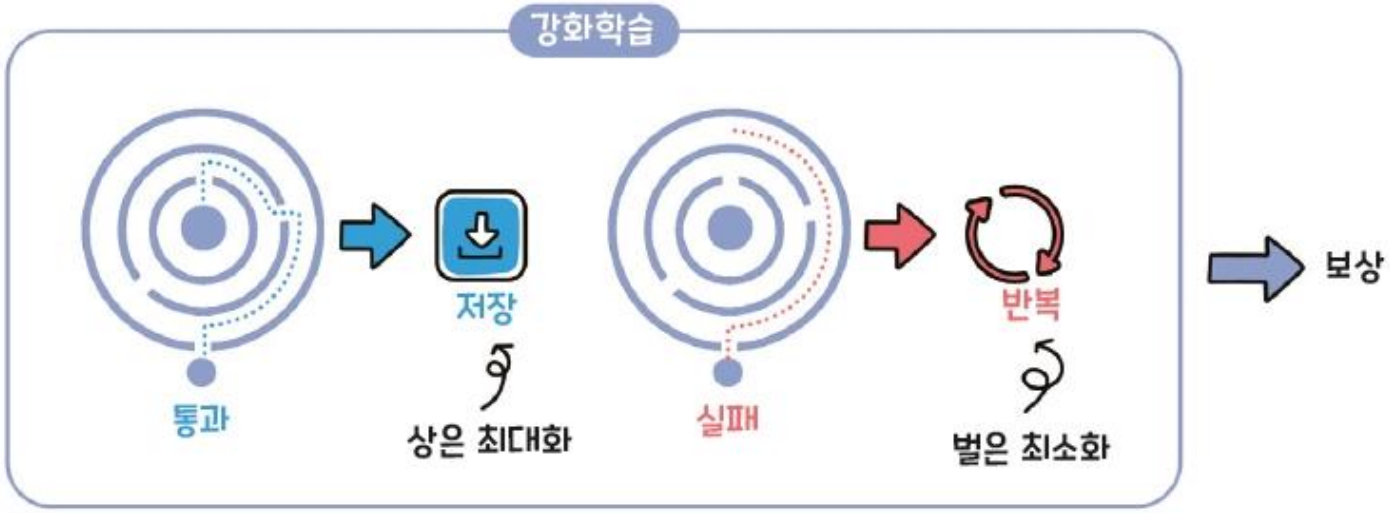


AI Intelligence Robot

Reinforcement Learning (RL)

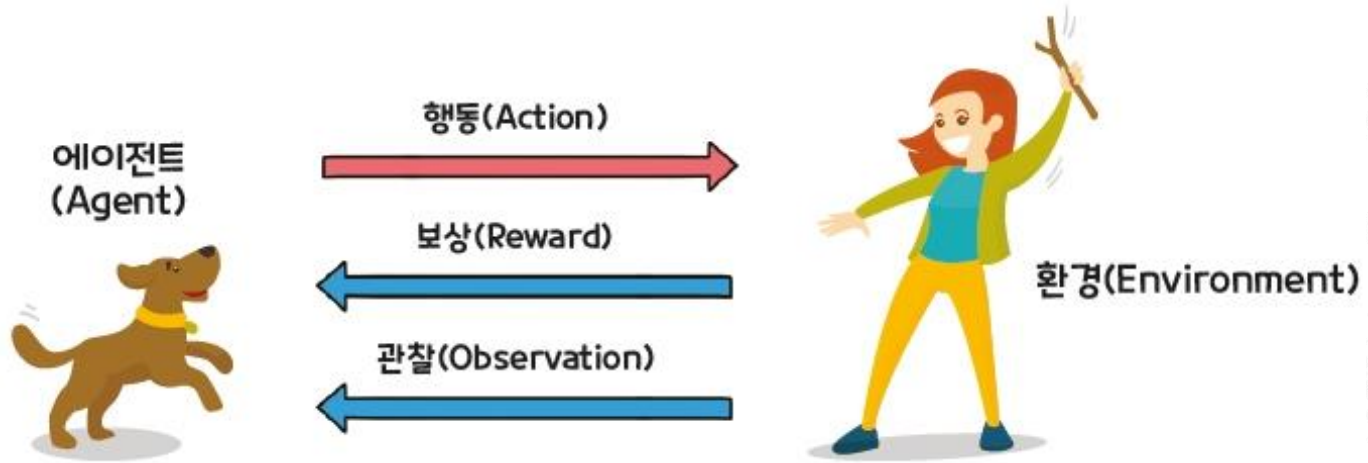
❖ Definition

- Learning by ***receiving rewards*** for one's actions
- Computer learns to select the ***optimal behavior for a given state***



Reinforcement Learning (RL)

- Concepts to understand RL
  - Agent : The entity that takes actions in response to the given situation or problem
  - State : The current situation or condition at a specific point in time
  - Action : The choices or moves the player (agent) can make
  - Reward : The gain or benefit received when the player takes a certain action
  - Environment : The problem or setting in which the agent operates
  - Observation : Information about the environment that the agent collects (sees or hears)



Reinforcement Learning (RL)

- Task execution
  - Tasks are accomplished by adapting to changes in ***state*** and ***behavior*** to achieve goals
  - Task that cannot be solved through supervised or unsupervised learning

    → Solved with ***reinforcement learning***

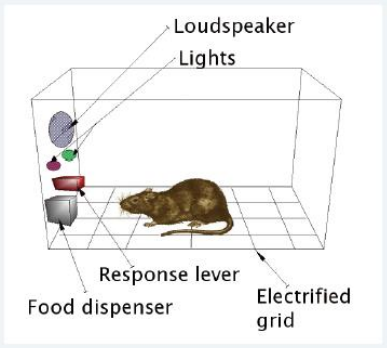
(a) 사람의 걷기 학습 과정

(b) 2족 로봇의 걷기

Reinforcement Learning (RL)

- Humans and animals interacting with the environment
  - Observe the state of the environment and decide actions that benefit oneself
  - If the outcome of an action is favorable, it is remembered and repeated;
    - e.g., Learning to ride a bike
    - e.g., Skinner's behavior psychology experiments
    - e.g., Go, video games, etc
  - ***Cycle: Action → State change → Reward***

(a) 어린이의 자전거 배우기



(b) 스키너 상자(출처: 위키백과)

(c) 학습 사이클

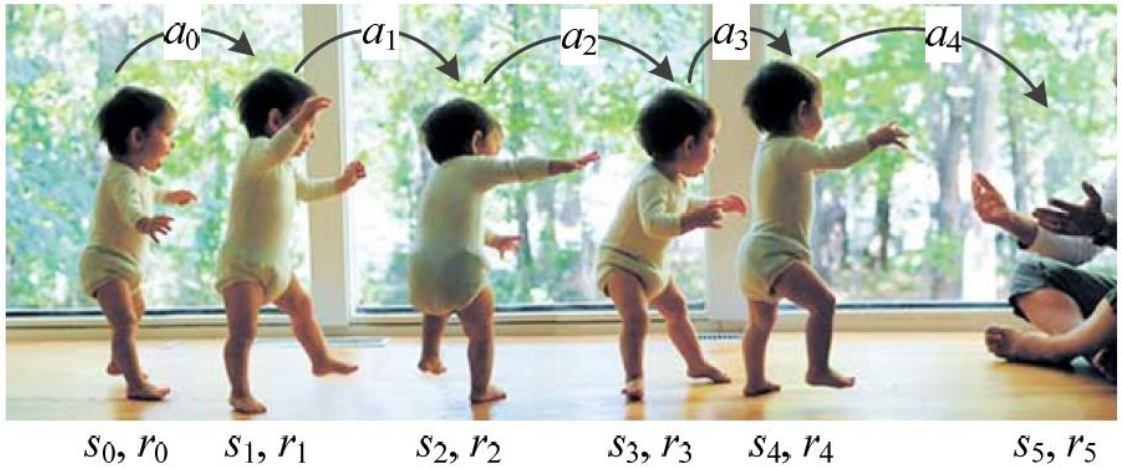
Reinforcement Learning (RL)

- Can a computer learn in this way?
  - Supervised learning methods (e.g., multi-perceptron, convolutional neural networks, recurrent neural networks) are insufficient
  - Reinforcement learning provides a solution
- Application of RL
  - Online games like StarCraft
  - Games like Go (AlphaGo), chess, and other board games
  - Intelligent robot control
  - Autonomous vehicle navigation
  - Smart factory control (precise production line management)

Principles and Characteristics of Learning

- Core equation of RL
  - Sequential processing of state (s_{state}), action (a_{action}), and reward (r_{reward}) over time *t*=0, 1, …, *T*
    - $s_0, r_0 \xrightarrow{a_0} s_1, r_1 \xrightarrow{a_1} s_2, r_2 \xrightarrow{a_2} \cdots\cdots \xrightarrow{a_{T-1}} s_T, r_T$
  - Expressed as a function:

$$f: (s_t, a_t) \rightarrow (s_{t+1}, r_{t+1})$$

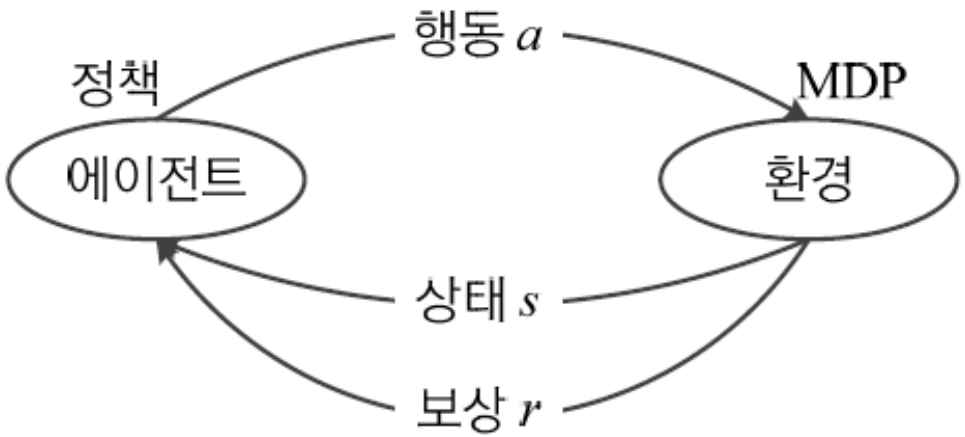


Principles and Characteristics of Learning

- Reward allocation strategy
  - When it is ***difficult to allocate rewards at each moment***, set ***intermediate rewards*** to 0 and determine the reward only at the final moment
    - In a walking task, intermediate rewards are 0 and, at the final moment, give a reward of 1 for reaching the father, or -1 for falling
    - In Go, assign intermediate rewards as 0, with a reward of 1 for winning and -1 for losing

Principles and Characteristics of Learning

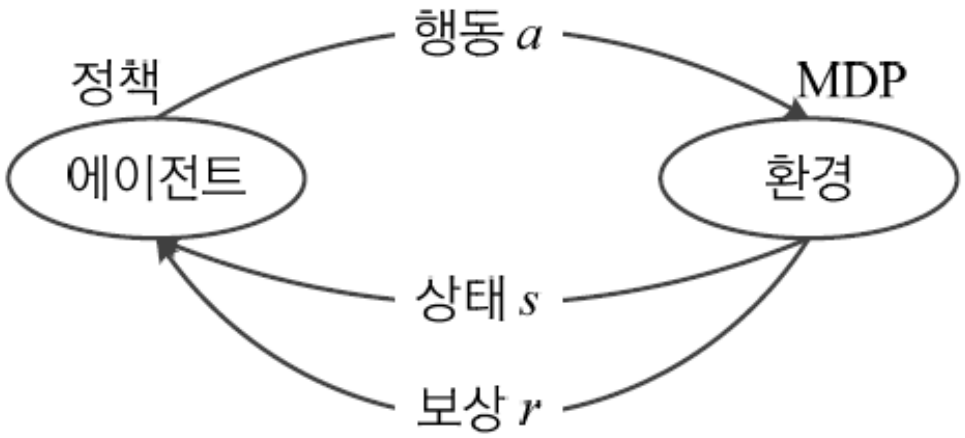
- Agent and Environment
  - Agent has a ***policy*** and determines its action based on it
    - The policy must be learned through training
  - Environment operates based on the MDP(Markov decision process) determining state transitions and rewards
    - MDP is provided: It can be analogous to a supervised learning training set



Computation model of reinforcement learning

Principles and Characteristics of Learning

- Goal of RL is ***to maximize cumulative rewards***
  - To achieve this, good actions must be taken at every moment
    - → A good policy is required to decide on actions
  - RL uses a given MDP to ***find the optimal policy***
    - This can be compared to supervised learning, where the goal is to find the optimal parameters using a training dataset



Computation model of reinforcement learning

Principles and Characteristics of Learning

- Representation of States, Actions, and Rewards
  - In most applications, these take discrete values:
    - State set $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$,
    - Action set $\mathcal{A} = \{a_1, a_2, \cdots, a_m\}$,
    - Reward set $\mathcal{R} = \{r_1, r_2, \cdots, r_l\}$
  - Examples
    - $P(s_{t+1} = s_{58} | s_t = s_{122})$: The probability of transitioning to state s_{58} at time *t + 1 given the state was* s_{122} at time *t*
    - $P(s_{t+1} = s_{58} | s_t = s_{122}, a_t = a_2)$: The probability of transitioning to state s_{58} at time *t + 1 given the state was* s_{122} and action a_2 was taken at time *t*

Principles and Characteristics of Learning

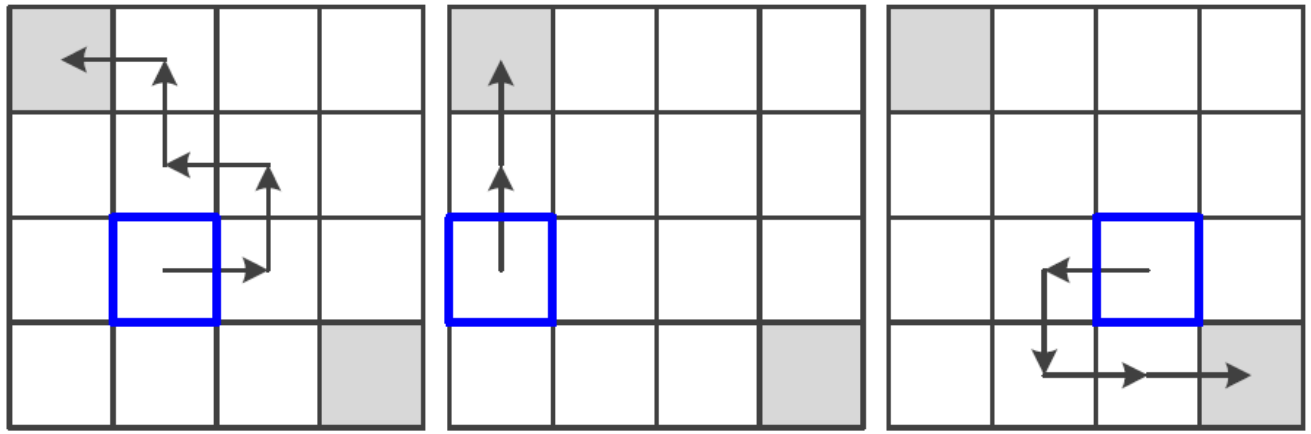
❖ Policy

- An agent determines its actions based on ***probabilistic rules, known as policies***
- For example, if we assume ***equal probabilities*** for four actions (east, west, south, north), the policy can be written as:

$$P(a = 동|s = i) = P(a = 서|s = i) = P(a = 남|s = i) = P(a = 북|s = i) = \frac{1}{4},\ i = 2,3,\cdots,15$$

- Is a policy using equal probabilities a good policy?

$$P(a = 동|s = i) = \frac{1}{3}, P(a = 서|s = i) = \frac{1}{3}, P(a = 남|s = i) = \frac{1}{3}, P(a = 북|s = i) = 0,\ i = 2,3,4$$

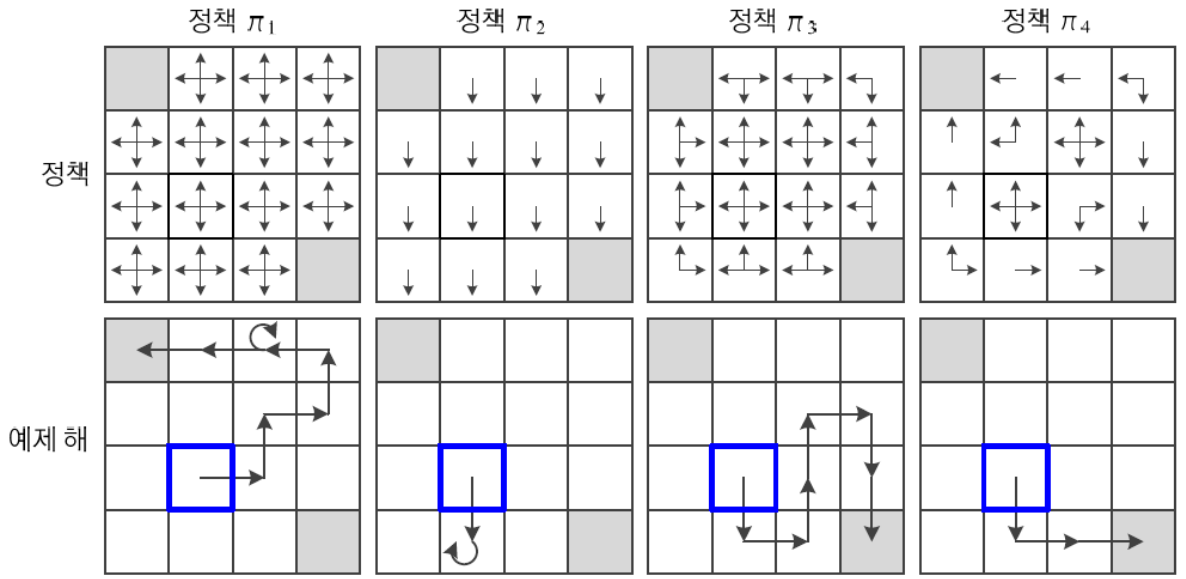


Policy Model Example

❖ Policy (π)

- Specifies the probability of taking an action *a* in a given state *s*

$$\pi(a|s) = P(a|s), \forall s, \forall a$$



남으로만 갈 수 있는 π_2라는 정책을 생각해 보자. π_2는 다음과 같은 식으로 정의된다. 이 정책은 4, 8, 12 중 한 곳에서 시작하지 않는 한 종료 상태에 도달할 수 없다. 따라서 해의 품질을 떠나 아예 채택할 수 없는 정책이라 할 수 있다.

$$\pi_2: \ P(\text{동}|i) = P(\text{서}|i) = P(\text{북}|i) = 0, \ P(\text{남}|i) = 1, \text{ 상태 } i = 2, \cdots, 15$$

Policy Model Example

정책 π_3은 π_1을 개선한 것이다. 가장자리의 칸에서 바깥으로 나가는 방향을 허용하지 않음으로써 −2라는 보상을 피하는 정책이다. π_3 수식은 다음과 같다.

$$\pi_3 \begin{cases} P(\text{동}|i) = P(\text{서}|i) = P(\text{남}|i) = P(\text{북}|i) = \frac{1}{4}, & i = 6, 7, 10, 11 \\ P(\text{서}|i) = P(\text{남}|i) = \frac{1}{2}, & i = 4 \\ P(\text{동}|i) = P(\text{북}|i) = \frac{1}{2}, & i = 13 \\ P(\text{동}|i) = P(\text{남}|i) = P(\text{북}|i) = \frac{1}{3}, & i = 5, 9 \\ P(\text{서}|i) = P(\text{남}|i) = P(\text{북}|i) = \frac{1}{3}, & i = 8, 12 \\ P(\text{동}|i) = P(\text{서}|i) = P(\text{남}|i) = \frac{1}{3}, & i = 2,3 \\ P(\text{동}|i) = P(\text{서}|i) = P(\text{북}|i) = \frac{1}{3}, & i = 14, 15 \end{cases}$$

Policy Model Example

❖ Task of the learning algorithms

학습 알고리즘이 해야 할 일은 최적 정책 $\hat{\pi}$을 찾는 것이다.
즉, $\hat{\pi} = \underset{\pi}{\operatorname{argmax}}\, goodness(\pi)$

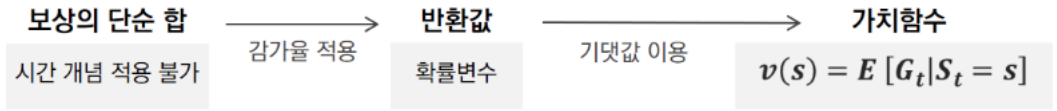
- The policy space is vast, making it impractical to evaluate every possible policy to select the best one

  →Therefore, strategies using value functions are employed

Value Functions and Q-Functions

❖ Value function (***State*** value function)

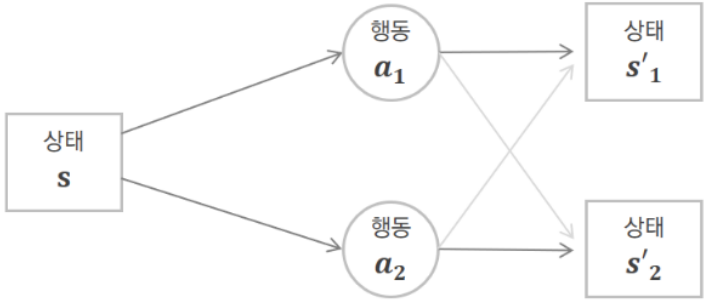
- Helps the agent calculate the “value” of a given state to choose actions



❖ Q-function (***Action*** value function)

- Represents the expected return for taking a specific action a in a specific state s

$$q_\pi(s, a_1) = E_\pi[G_t|S_t = s, A_t = a_1]$$



$$q_\pi(s, a_2) = E_\pi[G_t|S_t = s, A_t = a_2]$$

Exploration and Exploitation

- Conflict between exploration and exploitation
  - **Exploration**: A strategy to search the ***entire space thoroughly***
  - **Exploitation**: A strategy to focus on ***specific areas to maximize rewards***
    - Random search algorithms are extreme exploration methods, while gradient descent is an exploitation method
    - Exploration can take too long, and exploitation risks getting stuck in local optima
- Maintaining a balance between exploration and exploitation
  - ***Multi-armed bandit problem***

Exploration and Exploitation

- Exploration policy *vs.* Exploitation policy
  - Two extremes:
    - Exploration policy: Always selecting actions randomly from the beginning to the end
    - Exploitation policy: After some trials, only selecting the option with the highest observed reward rate
  - ***Finding a balance between the two is important***
    - A policy that favors the slot machine with the highest observed success rate while still occasionally exploring other options at a fixed rate
- If episode is long enough, solution is simple
  - In RL, the entire sequence of actions from start to finish is called an ***episode***
  - With sufficiently long episodes, the success rates can be calculated, and the policy can converge to selecting only the slot machine with the highest success rate
    - Long episodes allow identification of the "***optimal policy***"
    - If the reward probabilities are frequently changed by the user, time and resources may become insufficient to adapt

Exploration and Exploitation

- Strategy 1: Greedy search

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbf{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbf{1}_{A_i=a}}$$

이 슬롯머신으로 얻은 보상의 합 / 전체 슬롯머신 시도 횟수

$$A_t \doteq \arg\max_a Q_t(a),$$

기대 보상을 최대로 만들어주는 슬롯머신을 선택

Exploration and Exploitation

- Strategy 2: e-Greedy search

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Repeat forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Exploration and Exploitation

❖ Strategy 3: Upper-Confidence-Bound

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

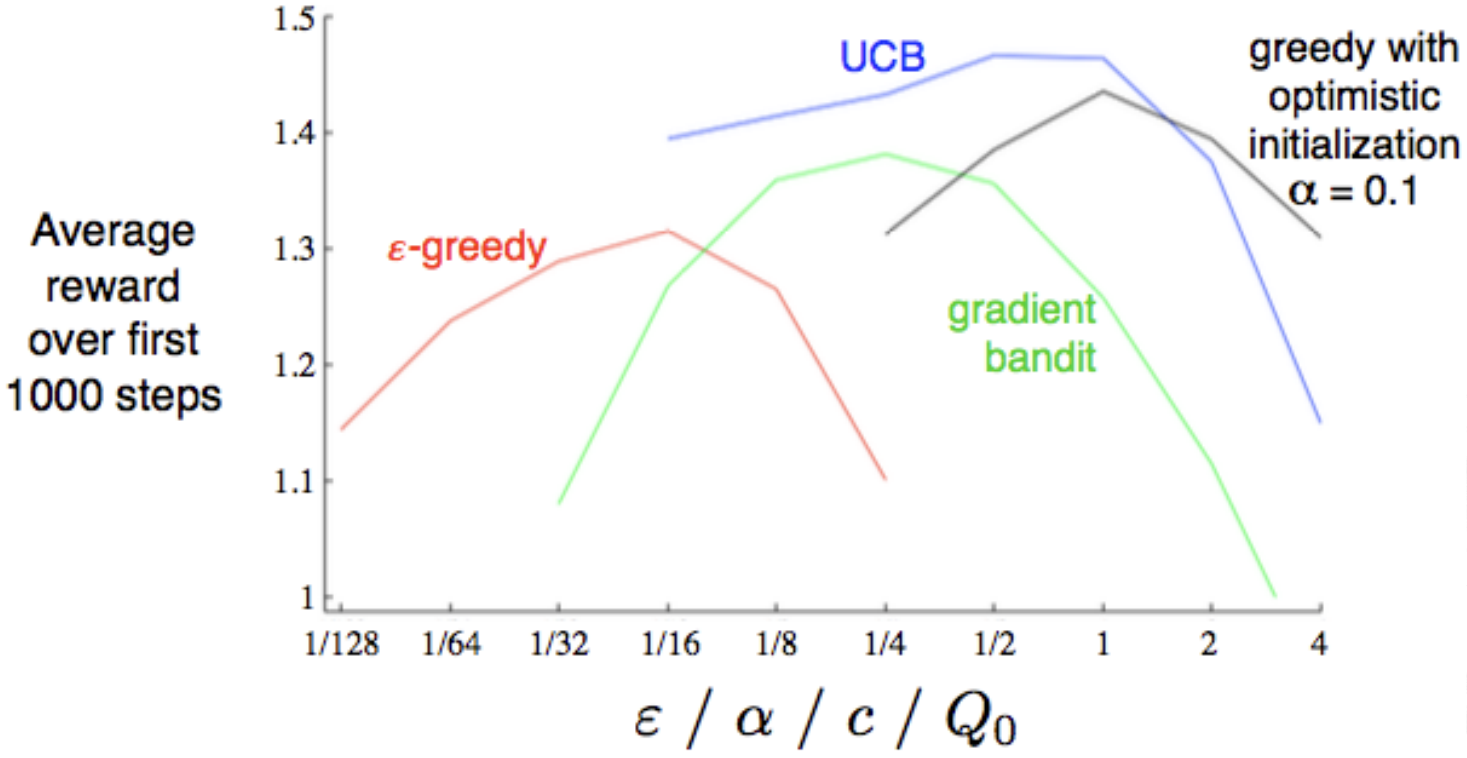
$N_t(a)$: 관측값에서 action a를 선택했던 횟수
c : exploration을 조정하는 하이퍼파라미터

UCB 알고리즘에서 Action 선택 알고리즘

$$A_t \doteq \arg\max_a Q_t(a),$$

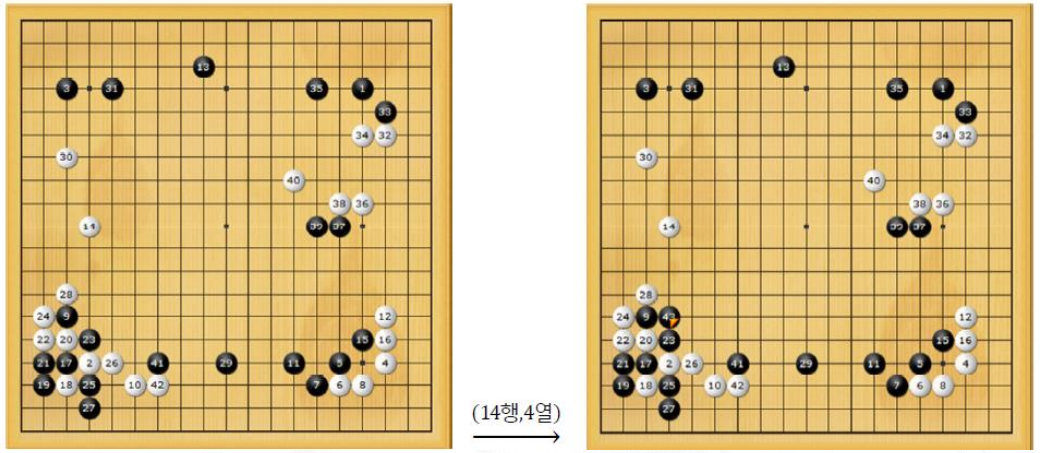
greedy 알고리즘

Exploration and Exploitation



Computational Model

- Monte Carlo method
  - A method that generates random samples to simulate real-world or mathematical phenomena
  - Random functions and ***epsilon-greedy functions*** are examples of Monte Carlo methods
  - In AI, Monte Carlo methods are utilized for various purposes:
    - Monte Carlo tree search algorithms



Computational Model

- Markov decision process (MDP)
  - Defines the rules that govern state types, action types, reward types, and state transitions resulting from actions
- States, Environment, and Rewards

$$\begin{cases} \text{State set: } S = \{s_1, s_2, \cdots, s_l\} \\ \text{Action set: } A = \{a_1, a_2, \cdots, a_m\} \\ \text{Reward set: } R = \{r_1, r_2, \cdots, r_n\} \end{cases}$$

  - When rewards are given
    - ***Immediate*** rewards: Multi-armed bandits
    - ***Delayed*** rewards: FrozenLake, Go, chess, video games etc. (most problems involve delayed rewards)

Policy and Value Functions

- The core of RL is finding a good policy
  - A good policy allows the ***agent to maximize cumulative rewards by choosing optimal actions at every moment***
  - Finding a good policy can be compared to supervised learning's optimization of an objective function
- Good policy
  - A good policy should allow the agent to maximize cumulative rewards, even if it ***involves taking short-term sacrifices*** to achieve long-term gains
    - In Go, sacrificing smaller stones to capture a larger group

Comparison of RL and Supervised Learning

	지도 학습(신경망)	강화 학습
문제(데이터)	훈련 집합 X(특징 벡터)와 Y(레이블)	환경(식 (9.2)의 상태 전이 확률 분포) 또는 환경에서 수집한 데이터(에피소드)
최적화 목표	신경망 출력 **o**와 레이블 **y**의 오차인 $\|\mathbf{o}-\mathbf{y}\|$ 최소화	누적 보상 최대화
학습 알고리즘이 알아내야 하는 것	오차를 최소화하는 신경망의 가중치	누적 보상을 최대화하는 최적 정책
품질을 평가하는 함수	손실 함수	가치 함수
학습 알고리즘	스토케스틱 그레이디언트 하강법 (SGD)	동적 프로그래밍, Sarsa, Q 러닝, DQN 등

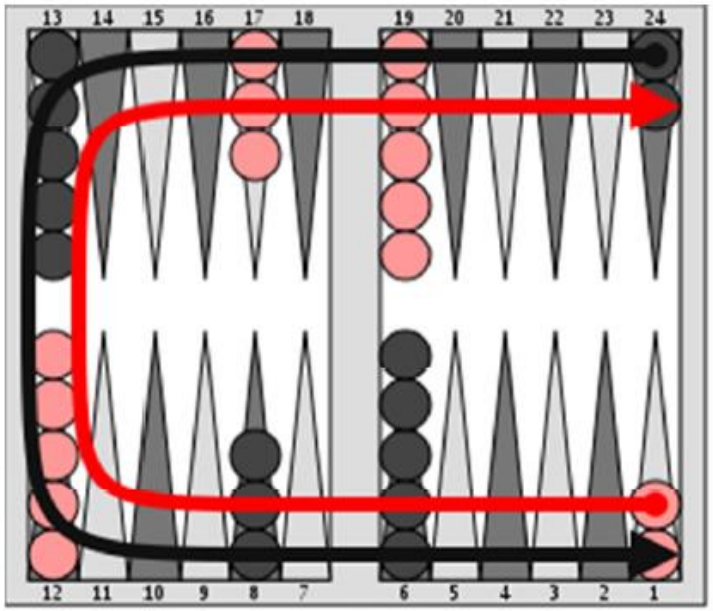
Application Examples

- Significance of these tools in AI
  - **TD-gammon** (Temporal difference gammon): The first case of self-play learning
  - **DQN** (Deep Q-Network): Achieved high performance across various types of games using the same neural network

TD(Temporal Difference)-gammon

❖ Rule of backgammon game

[그림 9-18]은 백가몬backgammon 게임이다. 검은 말과 빨간 말이 대결하는데, 검은 말은 낮은 번호로 이동하고 빨간 말은 높은 번호로 이동한다. 15개의 검은 말이 모두 1~6번으로 이동한 다음 밖으로 빠져 나와야 한다. 빨간 말은 19~24로 이동한 후 빠져 나와야 한다. 먼저 빠져 나오는 말이 이긴다. 게임은 주사위 2개를 던져 진행된다. 현재 검은 말 차례인데 1과 5가 나왔다고 하면, 검은 말 2개를 각각 1과 5만큼 시계 반대 방향으로 이동할 수 있다. 예를 들어, 24번에 있는 말을 각각 23과 19로 옮길 수 있다. 그런데 19번 위치에 빨간 말이 5개 있으므로 19번으로 이동할 수 없다. 빨간 말이 이동을 방해하고 있다고 볼 수 있는데, 상대편을 방해하는 것은 일종의 게임 전략이다. 만약 19번에 빨간 말이 하나뿐이라면 검은 말이 빨간 말을 잡을 수 있다. 잡힌 말은 바깥에 두었다가 처음 위치에서 다시 시작해야 한다. 더 자세한 게임 규칙은 [Tesauro1995]를 참조하라.

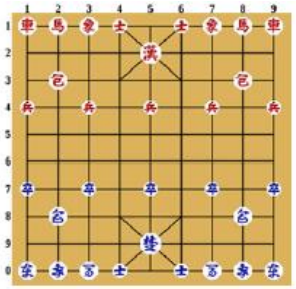


TD-gammon

- Backgammon program
  - Tesauro developed NeuroGammon in the early 1990s using neural networks (low performance)
  - Later evolved into ***TD-gammon by applying RL***
- TD-gammon
  - Based on multi-layer perceptron with a single hidden layer to approximate the value function
    - Input layer: 198 nodes
      - Represents the positions of black and red checkers: $2 \times 24 \times 5 = 192$
      - Represents the bar state: 2 nodes
      - Represents the number of removed checkers: 2 nodes
      - Represents the number of captured checkers: 2 nodes
    - Output layer: 4 nodes
      - 2 nodes to indicate the winning side
      - 2 nodes to indicate the losing side
  - Solves the problem using temporal difference learning
  - Learn via self-play (does not rely on human-provided strategies)

RL and Board Games



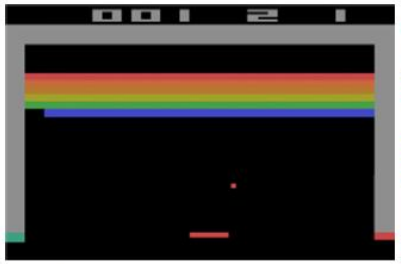




- Significant efforts have been made to develop board game programs at the world champion level
  - Tic-Tac-Toe: With $3^9 = 19683$ states, a simple brute-force program can derive the optimal policy
  - Checkers: Samuel developed a program in t he 1950s using classical algorithms [Samuel, 1959]
  - Chess: Specialized VLSI hardware for evaluating moves was developed
    - Advance machine learning algorithms were combined with supercomputers to achieve world champion status [Campbell2002]
  - Go: AlphaGo combined RL and deep learning to defeat top human players [Silver2016]

DQN: Atari Video Games

- Atari Video Game Program
  - Developed a program to automatically play 49 different games on the Atari 2600 platform
    - Each game varies significantly in terms of state representation, game strategy, and input method


Crazy climber


Breakout

Ms. Pacman

Road runner

  - A program utilizing RL uses:
    - The same input, neural network architecture, and hyperparameters for all games
    - ***Achieved expert-level performance*** in nearly all games
  - Compared to previous methods where neural networks were designed separately for each game, this approach show:
    - Significant improvements in generalization across tasks
    - Humans, when familiar with certain tasks, can perform similarly well on related tasks

DQN: Atari Video Games

- DQN(deep Q-network)
  - A model combining convolutional neural networks and Q-learning
  - Input layer
    - Game screen frames (tensor of the last 4 frames, size $84 \times 84 \times 4$)
    - These tensors represent the game state
  - Output layer
    - Each game requires between 4 to 18 joystick actions, which are applied as needed per game