Machine Learning in Practice
#5: Reinforcement Learning

Sang-Hyun Yoon

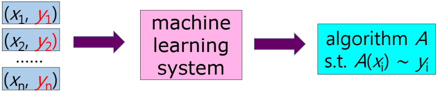
Summer 2019

Outline

Recall: Supervised vs. Unsupervised vs. Reinforcement

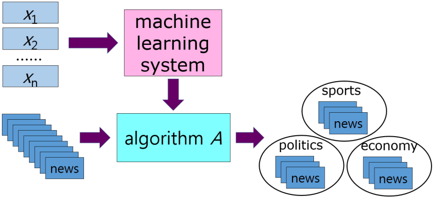
Supervised learning (지도 학습) (가장 널리 사용)

- Input과 output이 모두 off-line에 주어지는 경우



Unsupervised learning (비지도 학습)

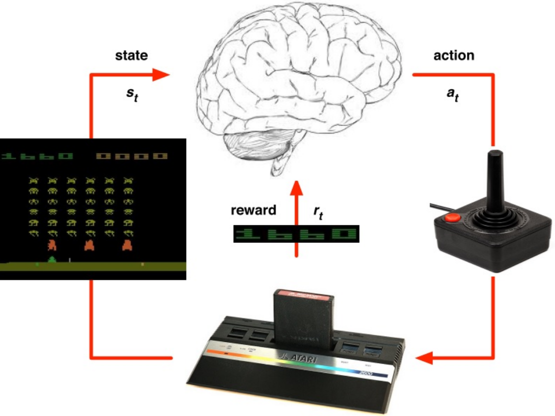
- Input에 대응되는 output이 전혀 주어지지 않는 경우



Reinforcement learning (강화 학습)

- Input에 대응되는 output이 off-lne에 주어지지 않고 environment로부터 on-line으로 받는 reward로부터 계산
  - cf. off-line vs. on-line algorithm

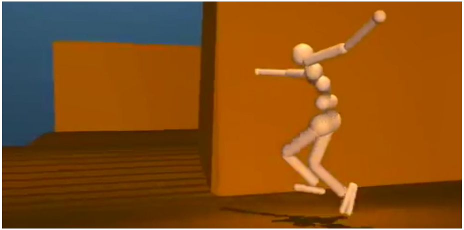
Reinforcement Learning: Atari Example



- Input: 게임 state
  - bitmap 픽셀값들
- Output: 최적 action
  - 게임의 rule조차 모르는 상황
  - 나중에 게임 결과를 봐야 계산할 수 있음
- Environment: 게임 SW
- Reward: 화면 corner에 보이는 점수 변화량

- 게임 rule을 모르므로 output(최적 action)을 미리 알 수 없음
- Reward를 feedback 삼아서 이전에 선택한 action을 평가하고 시행착오를 겪으며 조금씩 향상시킴
- Environment를 **Markov decision process**로 표현가능

Reinforcement Learning: Parkour Example



- Input: 인형 눈에 보이는 주변 환경 bitmap
  - State: input history + 인형의 현재 위치/자세/속도/가속도
- Output: 최적 action (관절들의 각도 조절)
- Environment: 운동/충돌 관련 운동 방정식
  - **Markov decision process**로 표현 가능
  - 관절이 20개가 넘어서 운동 방정식이 매우 복잡
  - 운동 방정식의 변수 갯수가 수십개에 달해 수학적 분석을 통한 제어는 거의 불가능
- Reward: 인형이 통과하는 장애물 갯수 + 생존 시간 등

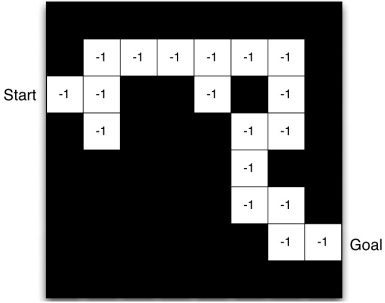
Reinforcement Learning: Big Picture

- 강화학습 대상 문제들의 environment는 대부분 **Markov decision process**(**MDP**)로 표현 가능
  - Stochastic process: random variables with time index
    - random 변수들은 state를 표현
  - Markov process = "historyless" stochastic process
  - Markov reward process = Markov process + reward
  - Markov decision process = Markov reward process + action
  - 수업에서 다룰 문제들은 non-random MDP로 가능
- 강화학습의 목표는 각 state(input+α)에 대응되는 최적 action(output)을 계산하는 것 (+ NN에 저장)
  - **policy**: input-to-output 함수 (random할 수도, 아닐 수도)
- 주어진 MDP와 policy에 대해 **value**를 문제에 맞게 정의
  - MDP의 reward에 의해 value가 결정됨. 즉 reward를 잘 정의
- MDP에 대해 **value를 최대화** 하는 **policy**를 찾는 것이 목표
- MDP를 정확히 알면 DP로, 모르면 **SARSA**/**Q-Learning**

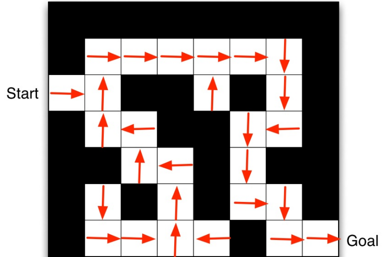
Markov Decision Process: Super-Mario Example

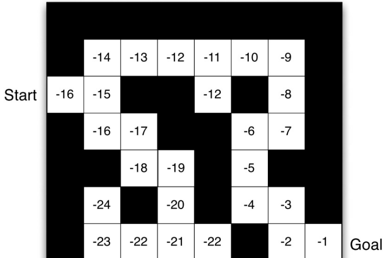
- State: 화면 bitmap $(+\alpha)$
- Markov process: 입력을 주지 않고 Mario를 놔뒀을 때 화면의 변화
  - Stochastic process인데, 과거의 몇개의 state들의 함수로 다음 state가 결정되므로 "historyless"
- Markov reward process: 점수 + 남은 Mario 갯수도 함께 고려한 것
- Markov decision process: Mario에 입력을 주는 것도 함께 고려한 것

Reinforcement Learning: Maze Example



- State: 흰 정사각형
- Action: 인접한 정사각형으로 이동
- Reward: 각 state마다 -1씩 받음
- 좌측하단 그림: **policy**의 예 (최적)
- 우측하단 그림: 이 policy를 적용했을 때의 **value** (최적)
- 최적 policy를 따라가면 최단 경로
- **Non-random** MDP/policy의 예





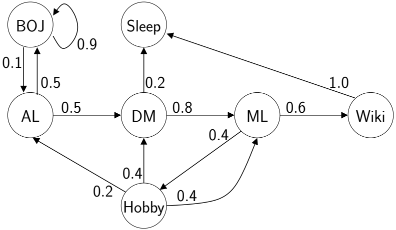
Outline

Markov Processes

Definition (Markov Process)

A **Markov process** (or Markov chain) is a pair $(\mathcal{S}, \mathcal{P})$

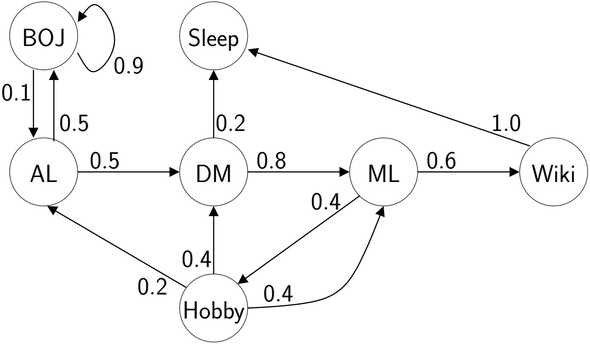
- $\mathcal{S}$: a (finite) set of states
- $\mathcal{P} : \mathcal{S}^2 \to [0, 1]$: a state transition probability



$$\mathcal{P} = \begin{array}{c} \\ \text{AL} \\ \text{DM} \\ \text{ML} \\ \text{Wiki} \\ \text{Hobby} \\ \text{BOJ} \\ \text{Sleep} \end{array} \begin{array}{c} \begin{array}{ccccccc} \text{AL} & \text{DM} & \text{ML} & \text{Wiki} & \text{Hobby} & \text{BOJ} & \text{Sleep} \end{array} \\ \begin{pmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1.0 \end{pmatrix} \end{array}$$

- 위와 같이 "historyless" stochastic system을 model할 때 유용
  - Given the present, the future is independent of the past
- $\mathbb{P}[S_{t+1} = s' \mid S_t = s, S_{t-1}, \cdots, S_1] = \mathbb{P}[S_{t+1} = s' \mid S_t = s] = \mathcal{P}(s, s')$
  - where each S_t is the random variable describing the state at t
  - 위 식이 성립되게 state를 충분히 많은 정보를 담도록 설정

Markov Processes: Episodes



Some episodes for the Markov process starting from the state AL

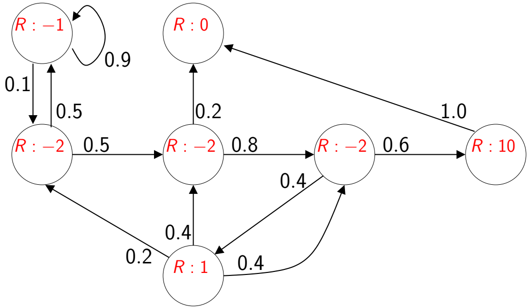
- AL DM ML Wiki Sleep
- AL BOJ BOJ AL DM Sleep
- AL DM ML Hobby DM ML Wiki Sleep
- AL BOJ BOJ AL DM ML Hobby AL BOJ AL DM ML Hobby DM Sleep

Markov Reward Processes (MRPs)

Definition (Markov Reward Process)

A Markov **reward** process (MRP) is a tuple $(\mathcal{S}, \mathcal{P}, R, \gamma)$ where

- $(\mathcal{S}, \mathcal{P})$: a Markov process
- $R : \mathcal{S} \to \mathbb{R}$: a reward function
  - $R(s)$ represents the expected intermediate reward at next state
  - 현재 state s에서의 intermediate reward로 해석해도 됨
- $\gamma \in [0, 1]$: a discount factor



Markov Reward Processes (MRPs): State-Value Functions

Definition (State-Value Function)

Given an MRP $(\mathcal{S}, \mathcal{P}, R, \gamma)$, its **state-value** function $v : \mathcal{S} \to \mathbb{R}$ is

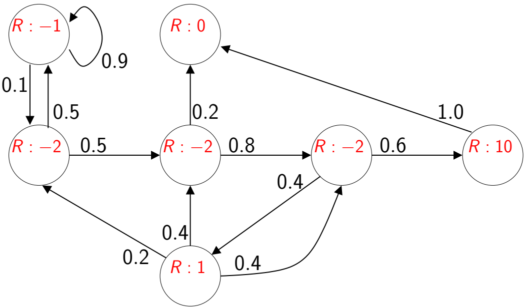
- $v(s) = \mathbb{E}\Big[R(s) + \gamma R(N_1(s)) + \gamma^2 R(N_2(s)) + \cdots\Big]$
  $= \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k R(N_k(s))\Big]$

where $N_k(s)$ is the random variable describing the state after k steps from s, i.e.

- $\mathbb{P}[N_k(s) = s'] = \sum_{s_i \in \mathcal{S}} \left(\mathcal{P}(s, s_1) \cdot \mathcal{P}(s_1, s_2) \cdot \ldots \mathcal{P}(s_{k-1}, s')\right)$

- Discount factor γ가 0에 가까을 수록 미래의 reward를 고려하지 않고 “근시안적”으로 state-value가 결정됨
- Discount factor γ가 1에 가까을 수록 미래의 reward에 가치가 높아지도록 state-value가 결정됨

State-Value Functions of MRPs: Example (1/2)

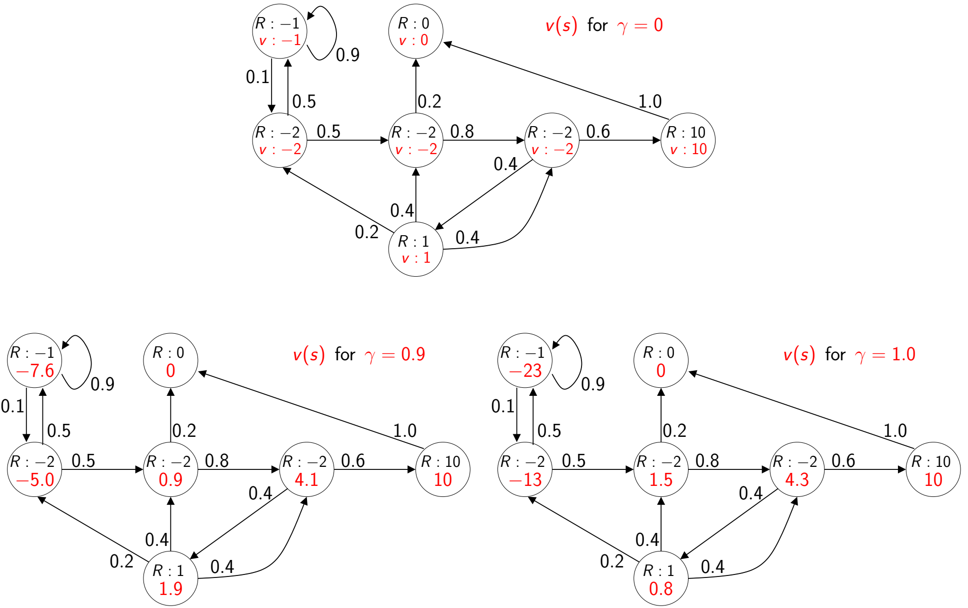


$$v(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(N_k(s))\right]$$

Starting from $s =$ DM with $\gamma = 1/2$:

- AL DM ML Wiki Sleep $\quad -2 - 2 \cdot \frac{1}{2} - 2 \cdot \frac{1}{4} + 10 \cdot \frac{1}{4} = -2.25$
- AL BOJ BOJ AL DM Sleep $\quad -2 - 1 \cdot \frac{1}{2} - 1 \cdot \frac{1}{4} - 2 \cdot \frac{1}{8} - 2 \cdot \frac{1}{16} = -3.125$
- AL DM ML Hobby DM ML Wiki Sleep
  $$-2 - 2 \cdot \frac{1}{2} - 2 \cdot \frac{1}{4} + 1 \cdot \frac{1}{8} - 2 \cdot \frac{1}{16} = -3.41$$
- $\cdots$

State-Value Functions of MRPs: Example (2/2)



State-Value Functions of MRPs: Equation in Matrix Form

Recall: State-Value Function of Markov Reward Process

Given an MRP $(\mathcal{S}, \mathcal{P}, R, \gamma)$, its **state-value** function $v : \mathcal{S} \to \mathbb{R}$ is

- $v(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(N_k(s))\right]$

$$\begin{aligned} v(s) &= \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R(N_k(s))\right] \\ &= \mathbb{E}\left[R(s) + \gamma \sum_{k=0}^{\infty} \gamma^k R(N_k(N(s)))\right] \\ &= \mathbb{E}\left[R(s)\right] + \gamma \cdot v(N(s)) \\ &= R(s) + \gamma \cdot \sum_{s' \in \mathcal{S}} \left(\mathcal{P}(s, s') \cdot v(s')\right) \end{aligned}$$

$$\begin{pmatrix} v(s_1) \\ \vdots \\ v(s_n) \end{pmatrix} = \begin{pmatrix} R(s_1) \\ \vdots \\ R(s_n) \end{pmatrix} + \gamma \begin{pmatrix} \mathcal{P}(s_1, s_1) & \dots & \mathcal{P}(s_1, s_n) \\ \vdots & \ddots & \vdots \\ \mathcal{P}(s_n, s_1) & \dots & \mathcal{P}(s_n, s_n) \end{pmatrix} \begin{pmatrix} v(s_1) \\ \vdots \\ v(s_n) \end{pmatrix}$$

$$\boxed{v = (I - \gamma \mathcal{P})^{-1} R}$$

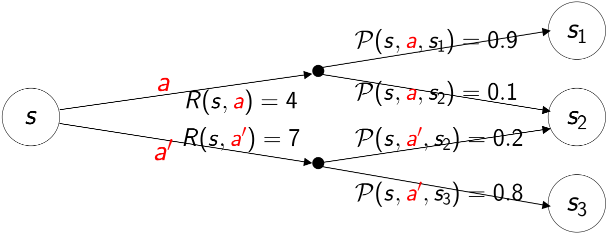
Outline

Markov Decision Processes (MDPs)

Definition (Markov Decision Process)

A Markov **decision** process (MDP) is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ where

- $\mathcal{S}$: a (finite) set of states
- $\mathcal{A}$: a (finite) set of actions
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$: a state transition probability
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: a reward function
  - $R(s, a)$는 현재 state s에서 action a를 택했을 때 다음 state 들에서 받을 수 있는 expected reward를 나타냄
- $\gamma \in [0, 1]$: a discount factor

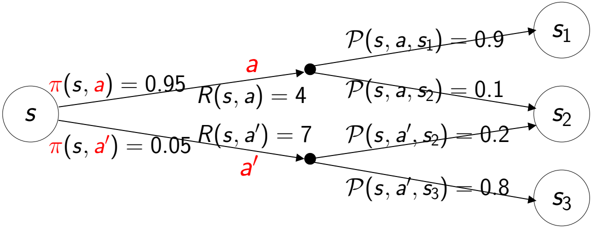


Markov Decision Processes (MDPs): Policies (1/2)

Definition (Policy)

A **policy** of an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ is a probability distribution over actions given states:

- $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ (non-random policy $\fallingdotseq \pi : \mathcal{S} \to \mathcal{A}$)



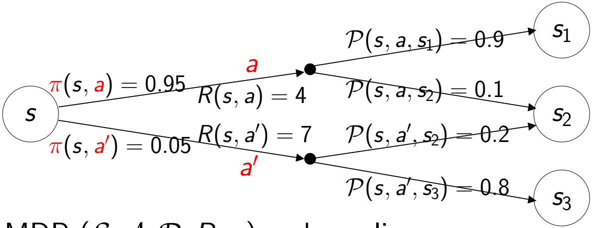
- MDP policies depend on the current state (not the history)
  - thus, time-independent (stationary)
- A policy "completely" defines state transitions of an MDP
  - "completely" = deterministically probabilistic
  - c.f. state transitions in nondeterministic finite automata

Markov Decision Processes (MDPs): Policies (1/2)

Definition (Policy)

A **policy** of an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ is a probability distribution over actions given states:

- $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ (non-random policy $\sqsubseteq$ $\pi : \mathcal{S} \to \mathcal{A}$)



Given an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ and a policy π,

- the state sequence $S_1, S_2, \cdots$ is a Markov process $(S, \mathcal{P}^\pi)$ where $\boxed{\mathcal{P}^\pi(s, s') = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{P}(s, a, s')}$
- the state/reward sequence is a Markov reward process $(S, \mathcal{P}^\pi, R^\pi, \gamma)$ where $\boxed{R^\pi(s) = \sum_{a \in \mathcal{A}} \pi(s, a) \cdot R(s, a)}$

Markov Decision Processes (MDPs): State/Action-Value Functions

Given an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ and a π, let

- $\mathcal{P}^\pi(s, s') \triangleq \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \mathcal{P}(s, a, s')$
- $N_k^\pi(s) \triangleq$ R.V. describing state after k steps from s under π
  - $\mathbb{P}[N_k^\pi(s) = s'] = \sum_{s_i \in \mathcal{S}} \left(\mathcal{P}^\pi(s, s_1) \cdot \mathcal{P}^\pi(s_1, s_2) \cdot \ldots \mathcal{P}^\pi(s_{k-1}, s')\right)$
- $R^\pi(s) \triangleq \sum_{a \in \mathcal{A}} \pi(s, a) \cdot R(s, a)$

Definition (State-Value Function)

The **state-value** function $v^\pi : \mathcal{S} \to \mathbb{R}$ is

- $v^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R^\pi(N_k^\pi(s))\right]$
  - i.e. expected total reward starting from s, and then following π

Definition (Action-Value Function)

The **action-value** function $q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is

- $q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \left(\mathcal{P}(s, a, s') \cdot v^\pi(s')\right)$
  - i.e. expected total reward starting from s, taking action a, and then following π

Outline

Optimal State/Action-Value Functions

Given an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$,

Definition (Optimal State-Value Function)

The optimal state-value function $v^* : \mathcal{S} \to \mathbb{R}$ is defined by

- $v^*(s) = \max\{v^\pi(s) \,|\, \text{policy } \pi \text{ of the MDP}\}$ for each $s \in \mathcal{S}$

Definition (Optimal Action-Value Function)

The optimal action-value function $q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined by

- $q^*(s, a) = \max\{q^\pi(s, a) \,|\, \text{policy } \pi\}$ for each $s \in \mathcal{S}, a \in \mathcal{A}$

- Given an MDP, our goal is to find a policy that makes value functions optimal
- Optimal policy: well-defined? exists?

Optimal Policy

- $v^*(s) = \max\{v^\pi(s) \mid \text{policy } \pi \text{ of the MDP}\}$ for each $s \in \mathcal{S}$
- $q^*(s,a) = \max\{q^\pi(s,a) \mid \text{policy } \pi\}$ for each $s \in \mathcal{S}, a \in \mathcal{A}$

Definition (Optimal Policy)

A policy π^* of an MDP is said to be optimal if,

- $v^{\pi^*}(s) = v^*(s)$ for all $s \in \mathcal{S}$

Theorem

For every MDP,

- *there exists an optimal policy*
- *every optimal policy π^* achieves the optimal action-value function, i.e. $q^{\pi^*}(s,a) = q^*(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$*
- $v^*(s) = \max_{a \in \mathcal{A}} q^*(s,a)$

- 매우 강력한 성질. man-optimal stable match와 유사

Finding an Optimal Policy

Theorem

There exists a deterministic optimal policy for any MDP

Thus, if an optimal action-value function q^* is available, an optimal policy is easily obtained: for each $s \in \mathcal{S}, a \in \mathcal{A}$,

- $$\pi^*(s,a) = \begin{cases} 1 & \text{if } q^*(s,a) = \max_{a' \in \mathcal{A}} q^*(s,a') \\ 0 & \text{otherwise} \end{cases}$$
  - 따라서, 앞으로 deterministic policy만 고려하고, policy 함수를 $\pi : \mathcal{S} \to \mathcal{A}$; $\pi(s) = a$ 형태로 섞어 쓰기도 한다

Then, how to find q^*?

Finding an Optimal Value Functions

Bellman Optimality Equations **(exercise)**

- $v^*(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \cdot v^*(s') \right)$
- $q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') \cdot \left(\max_{a' \in \mathcal{A}} q^*(s',a') \right)$

- Non-linear
- No closed form solution in general
- Iterative methods for find approximate solution
  - dynamic programming: policy iteration, value iteration
  - Monte-Carlo learning (with CNN)
  - temporal-difference control: SARSA, Q-learning (with CNN)

Summary: Equations for State/Action-Value Functions (link)

State/action-value functions: definition

- $v^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R^\pi\left(N_k^\pi(s)\right)\right] = \sum_{a\in\mathcal{A}} \pi(s,a)\cdot q^\pi(s,a)$
- $q^\pi(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} \left(\mathcal{P}(s,a,s')\cdot v^\pi(s')\right)$
  - for policy improvement step of policy iteration (& MC learning)

Inductive formula of value functions (directly from the definition)

- $v^\pi(s) = R^\pi(s) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}^\pi(s,s')\cdot v^\pi(s')$
  - for policy evaluation step of policy iteration

From the definition of optimal state/action-value functions

- $v^*(s) = \max_{a\in\mathcal{A}} q^*(s,a)$ (i.e. $\pi^*(s) = \operatorname{argmax}_{a\in\mathcal{A}} q^*(s,a)$)
  - for greedy policy improvement (of every learning methods)

Bellman optimality equations

- $v^*(s) = \max_{a\in\mathcal{A}} \left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\cdot v^*(s')\right)$
  - for value iteration
- $q^*(s,a) = R(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\cdot \left(\max_{a'\in\mathcal{A}} q^*(s',a')\right)$
  - for Q-learning

Outline

Policy Iteration (1/3)

Recall: State-Value Function in Inductive/Matrix Form

- $v^{\pi}(s) = R^{\pi}(s) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\pi}(s, s') \cdot v^{\pi}(s')$

  $$\left(= \sum_{a \in \mathcal{A}} \pi(s, a) \cdot \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \left(\mathcal{P}(s, a, s') \cdot v^{\pi}(s') \right) \right) \right)$$

  - $v^{\pi} = R^{\pi} + \gamma P^{\pi} v^{\pi}$

Iterative Policy Evaluation

Given a policy π,

- $v_1(s) =$ random value
- $v_{k+1}(s) = R^{\pi}(s) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\pi}(s, s') \cdot v_k(s')$
  - $v_{k+1} = R^{\pi} + \gamma P^{\pi} v_k$

Theorem

$\lim_{k \to \infty} v_k = v^{\pi}$ (by the contraction mapping theorem)

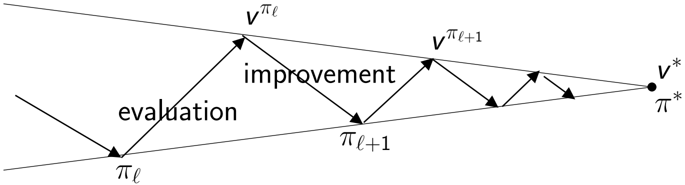
Policy Iteration (2/3)

Policy evaluation

- 주어진 π로부터 v_π의 근사치를 구하기 (앞 슬라이드)

Policy improvement

- 주어진 policy π_ℓ에 대해 policy evaluation으로 계산한 v^{π_ℓ}(의 근사치)를 이용하여 더 나은 policy $\pi_{\ell+1}$을 계산
- $\pi_{\ell+1} = \text{greedy}(v^{\pi_\ell})$ where
  - $\pi_{\ell+1}(s) = \text{argmax}_{a \in \mathcal{A}} \, q^{\pi_\ell}(s, a)$ (deterministic)
    where $q^{\pi_\ell}(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \left(\mathcal{P}(s, a, s') \cdot v^{\pi_\ell}(s')\right)$



Does π_ℓ converge to π^*?

Policy Iteration (3/3)

Does π_ℓ converge to π^*?

Theorem

$$\lim_{\ell \to \infty} \pi_\ell = \pi^*$$

Proof Sketch

1. π_ℓ은 deterministic하므로 $\pi_\ell : \mathcal{S} \to \mathcal{A}$ 형태로 나타내자. 그러면 $\pi_{\ell+1}(s) = \text{argmax}_{a \in \mathcal{A}}\, q^{\pi_\ell}(s,a)$이다
   - where $q^{\pi_\ell}(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \left(\mathcal{P}(s,a,s') \cdot v^{\pi_\ell}(s')\right)$
2. $q^{\pi_\ell}(s, \pi_{\ell+1}(s)) = \max_{a \in A} q^{\pi_\ell}(s,a) \geq q^{\pi_\ell}(s, \pi_\ell(s)) = v^{\pi_\ell}(s)$
3. $v^{\pi_\ell}(s) \leq q^{\pi_\ell}(s, \pi_{\ell+1}(s)) \leq v^{\pi_{\ell+1}}(s)$ (why $\leq$?)
4. If $v^{\pi_\ell}(s) = v^{\pi_{\ell+1}}(s)$, then $v^{\pi_\ell}(s) = \max_{a \in \mathcal{A}} q^{\pi_\ell}(s,a)$
5. Thus, the Bellman optimality equation is satisfied, and so π_ℓ is an optimal policy

Value Iteration

Recall: State-Value Function in Bellman Optimality Equation

- $v^*(s) \;=\; \max_{a\in\mathcal{A}} \left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \cdot v^*(s') \right)$

Iterative Evaluation of Optimal Value Function

- $v_1(s) \;=\;$ random value
- $v_{k+1}(s) \;=\; \max_{a\in\mathcal{A}} \left(R(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \cdot v_k(s') \right)$

Theorem

$\lim_{k\to\infty} v_k \;=\; v^*$ (by the contraction mapping theorem)

Policy iteration과의 차이

- 주어진 policy π에 대한 v^π equation을 사용하는 대신, optimal value function v^* equation을 이용
- 중간에 policy를 계산하지 않고 v^*만을 iterative evaluation한 후 v^*(의 근사값)을 이용해 π^*를 계산 ($\pi^* = \text{greedy}(v^*)$)

Policy/Value Iteration: Summary

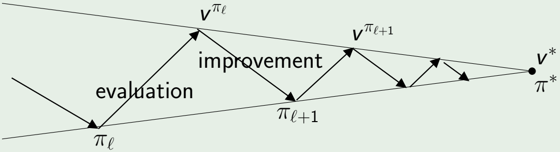
- Policy/value iteration은 **dynamic programming**에 기반한 방식으로 볼 수 있음 (induction on k)
- MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$에 대한 정보가 모두 알려져 있고 $\mathcal{S}$와 $\mathcal{A}$의 크기가 작으면 매우 효과적
  - $\mathcal{P}$와 R를 MDP의 **model**이라고 부름
- 하지만 MDP의 model에 대한 정보를 미리 정확하게는 모르거나 $\mathcal{S}$와 $\mathcal{A}$의 크기가 큰 경우에는 적용하기 힘듬
  - state s에서 action a를 취한 후에나 다음 state s'와 reward 값을 알게 됨
- **Model-free** learning: model을 미리 모르는 상황에서 optimal policy를 계산(학습)하는 방식
  - Monte-Carlo learning
  - temporal difference learning: SARSA, Q-learning

Outline

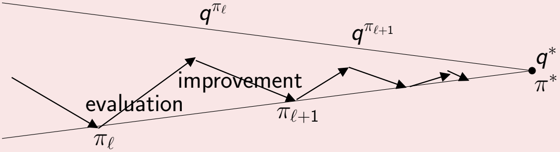
Policy Iteration vs. Monte-Carlo Learning

Recall: Policy Iteration

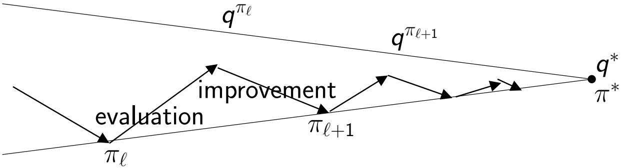
1. $v^{\pi_\ell} = \lim_{k\to\infty} v_k$ where v_k is
   - $v_{k+1}(s) = \sum_{a\in\mathcal{A}} \pi(s,a)\cdot\left(R(s,a)+\gamma\sum_{s'\in\mathcal{S}}\left(\mathcal{P}(s,a,s')\cdot v_k(s')\right)\right)$
2. $\pi_{\ell+1}(s) = \text{argmax}_{a\in\mathcal{A}}\, q^{\pi_\ell}(s,a)$ where (deterministic)
   - $q^{\pi_\ell}(s,a) = R(s,a)+\gamma\sum_{s'\in\mathcal{S}}\left(\mathcal{P}(s,a,s')\cdot v^{\pi_\ell}(s')\right)$



Monte-Carlo evaluation/improvement

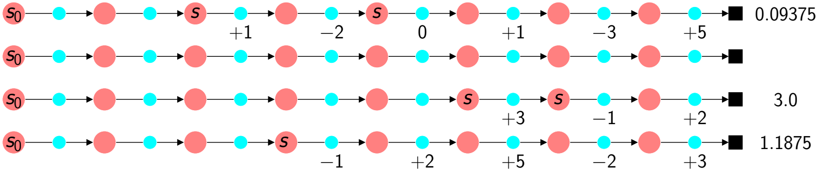


Monte-Carlo Learning (1/2)

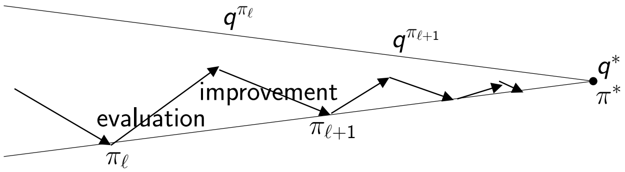


Monte-Carlo policy evaluation:

- Policy iteration처럼 evaluation을 반복해서 제대로 하지 않고, 여러 episode에서 얻은 정보로 q^{π_ℓ}(의 근사치)를 계산
- State-value function $v(s)$ 대신 action-value function $q(s, a)$를 evaluation해 둠 (for model-free learning. 다음 슬라이드 참고)
- Policy π_ℓ을 따라가면서 total reward을 얻고 averaging



Monte-Carlo Learning (2/2)



Monte-Carlo policy improvement: ϵ-greedy improvement

- $1-\epsilon$만큼 exploitation, ϵ만큼 exploration
  - 다른 경험도 해보기 위해 "모험요소"(exploration) 추가
- $\pi_{\ell+1}(s,a) = \begin{cases} \epsilon/|\mathcal{A}| + 1 - \epsilon & \text{if } a = \text{argmax}_{a' \in \mathcal{A}}\, q^{\pi_\ell}(s,a') \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$

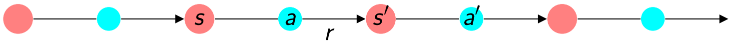
v대신 q에 기반하여 **model-free** learning 가능

- Policy improvement over v requires model of MDP
  - $\pi_{\ell+1}(s) = \text{argmax}_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s') v^{\pi_\ell}(s') \right)$
- Policy improvement over q is **model-free**
  - $\pi_{\ell+1}(s) = \text{argmax}_{a \in \mathcal{A}}\, q^{\pi_\ell}(s,a)$

Temporal-Difference Learning

Combines ideas from DP (policy/value iteration) & Monte-Carlo

- like MC: learns directly from experience (model-free)
  - c.f. 3년 내내 시행착오 겪으며 경험치 쌓기
- like DP: bootstrapping (guess로부터 guess를 업데이트)
  - final state까지 가보지 않고 estimate의 부분만을 보고 학습
  - c.f. 선배들의 경험을 들으며 시행착오 덜하며 경험치 쌓기



SARSA: on-policy temporal-difference learning

- Start with a random policy; iteratively improve

$$\begin{aligned} q(s,a) &:= (1-\alpha)\cdot q(s,a) + \alpha\cdot\left(r + \gamma\cdot q(s',a')\right) \\ &= q(s,a) + \alpha\cdot\left(r + \gamma\cdot q(s',a') - q(s,a)\right) \end{aligned}$$

Q-learning: off-policy temporal-difference learning

- Start with a random policy; iteratively improve

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

SARSA: Pseudocode

$q(s,a) :=$ `random value for each` $s \in \mathcal{S},\ a \in \mathcal{A}$
$q(s_f, a) := 0$ `for each` $a \in \mathcal{A}$ `and final state` s_f
`for (each episode)`
    $s :=$ `start state`
    $a := \pi(s)$ `where` $\pi = \epsilon\text{-greedy}(q)$
    `for (each step of the episode)`
        `take action` a `to get reward` r `& next state` s'
        $a' := \pi(s')$ `where` $\pi = \epsilon\text{-greedy}(q)$
        $q(s,a) := q(s,a) + \alpha \cdot \left(r + \gamma \cdot q(s', a') - q(s,a)\right)$
        $s, a := s', a'$

- ϵ-greedy policy로 따라가는 state sequence상에서 보이는 q 값들을 그대로 사용하면서 learning
- On-policy learning: "Learn on the job"
- `https://dnddnjs.gitbooks.io/rl/content/td_control.html`

Q-Learning: Pseudocode

```
q(s,a) := random value for each s ∈ S, a ∈ A
q(s_f,a) := 0 for each a ∈ A and final state s_f
for (each episode)
    s := start state
    for (each step of the episode)
        a := π(s) where π = ε-greedy(q)
        take action a to get reward r & next state s'
        q(s,a) := q(s,a) + α·(r + γ·max_{a'∈A} q(s',a') − q(s,a))
        s := s'
```

- ϵ-greedy policy로 따라가는 state sequence상의 q값을 그대로 사용하지 않고 $\max_{a' \in \mathcal{A}} q(s', a')$값을 사용하여 learning
- Off-policy learning: "Look over someone's shoulder"
  - off-policy: 움직이는 policy와 학습하는 policy를 분리
- q converges to q^* (under some mild condition)
- 성능이 좋아 가장 널리 사용 (CNN과도 상성이 좋음)

Monte-Carlo vs. Temporal-Difference (1/2)

TD는 마지막 reward를 보기 전에 배울 수 있음

- 매 step마다 on-line으로 학습
- MC는 episode의 마지막까지 기다려 reward를 알아야 함

TD는 마지막 reward이 없이도 배울 수 있음

- TD는 episode가 완전하지 않아도 (끝나지 않아도) 학습
- MC는 episode가 완전할 때만 학습 (예: 바둑)

Bias/Variance trade-off

- TD: low variance, high bias (초기값에 sensitive)
- MC: low bias, high variance (초기값에 insensitive)

Markov property

- TD: Markov property를 만족하는 MDP에 효과적
- MC: non-Markov stochastic process에 효과적

Monte-Carlo vs. Temporal-Difference (2/2)

예: 자동차 운전

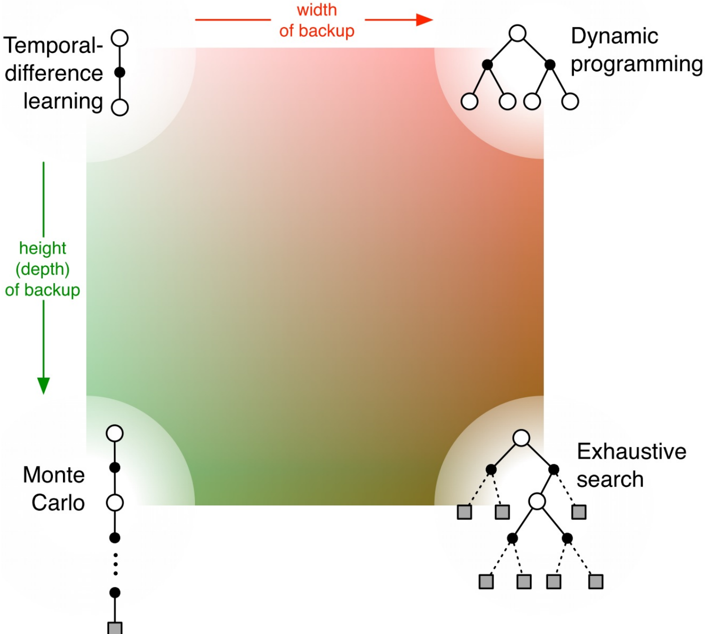
Monte-Carlo:

- 자동차가 사고가 나기 직전에 피하면 total reward $= 0$
- 그때까지의 state들은 사고가 날 수 있는 위험한 state인데 negative reward를 받지 못하고 제대로 update안됨
- 실제로 사고가 난 경우에만 위험한 state를 피하도록 학습

Temporal-difference:

- 사고가 나기 직전이라고 판단이 되면 reward $= -\infty$
- 그 직전의 state는 negative reward로 업데이트 할 수 있음
- 실제로 사고가 나지 않은 경우에도 위험한 state를 피하도록 학습

Classification of RL Methods: Unified View

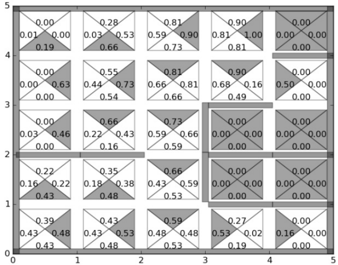


Outline

Q-Learning Example #1: Maze (`maze_MDP/Q_learning.py`)

- BFS로 쉽게 풀리지만 Q-learner가 독학하게 함이 목표
- Environment는 non-random MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:
  - $\mathcal{S} = \{0, 1, \cdots, 24\}$, start state $= 0$, final state $= 24$
  - $\mathcal{A} = \{\text{UP, DOWN, RIGHT, LEFT}\}$
  - $\mathcal{P}(s, a) =$ state s에서 방향 a로 움직였을 때의 state
  - $\mathcal{R}(s, a) = \mathcal{P}(s, a)$가 final state면 1, 아니면 0
  - $\gamma = 0.9$ (1이 아니므로 최단경로로 유도하는 효과)
  - Random maze므로 길이 없을 수도 있음 (끄고 다시 수행)

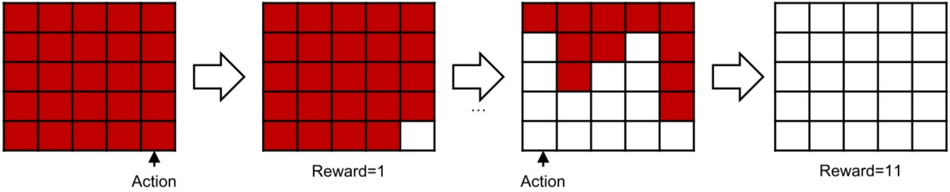
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4



- 각 state s에서 $q(s, a)$를 최대화 하는 a를 따라가면 최단

Example #2: Breakout (breakout_MDP/Q_learning/Q_test.py)

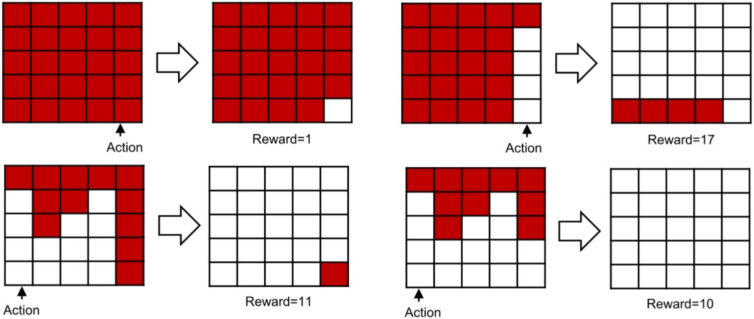
- Maze 문제와 마찬가지로 문제에 대한 지식 없이 독학하게
- Environment #1은 non-random MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:
  - $\mathcal{S} = \{0, 1, \cdots, 5\}^5$ (각 열의 남은 벽돌 갯수)
    start state $= (5, 5, 5, 5, 5)$, final state $= (0, 0, 0, 0, 0)$
  - $\mathcal{A} = \{0, 1, 2, 3, 4\}$ (벽돌 하나를 부수려는 열의 index)
  - $\mathcal{P}(s, a)$ = state s에서 열-a의 벽돌 하나를 부순 후의 state
    열-a의 벽돌이 모두 없어지면 final state로 바로 감
  - $\mathcal{R}(s, a)$ = s에서 $\mathcal{P}(s, a)$로 오면서 부숴진 벽돌 갯수
  - $\gamma = 0.9$ (1이 아니므로 최소횟수로 유도하는 효과)



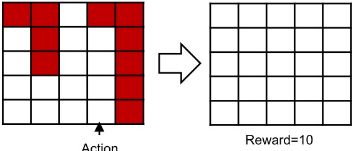
- 각 s에서 $q(s, a)$를 최대화 하는 열-a를 부숨 (최적해: 5회)

Example #2: Breakout (breakout_MDP/Q_learning/Q_test.py)

- Environment #2: 어떤 열의 벽돌이 모두 없어지면 나머지 열들의 상단 4행 벽돌들도 모두 제거됨 (최적해: 9회)

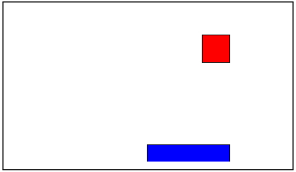


- Environment #3: 어떤 두 열의 벽돌이 모두 없어지면 모든 벽돌들이 제거됨 모두 제거됨 (최적해: 10회)



Example #3: Catch (`catch_MDP/Q_learning/Q_test.py`)

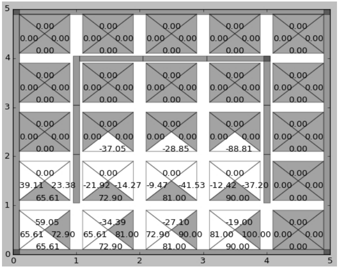
- Maze 문제와 마찬가지로 문제에 대한 지식 없이 독학하게
- Environment는 non-random MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:
  - $\mathcal{S} = \underbrace{\{0, 1, \cdots, 9\}^2}_{\text{과일의 위치}} \times \underbrace{\{0, 1, \cdots, 7\}}_{\text{basket의 위치}},$
  - $\mathcal{A} = \{\text{RIGHT, LEFT, WAIT}\}$ (basket의 이동 방향)
  - $\mathcal{P}(s, a) =$ state s에서 basket을 방향 a로 움직이고 과일이 한칸 떨어졌을 때의 state
  - $\mathcal{R}(s, a) =$ 과일이 바닥에 있고 basket이 받았으면 1, 놓쳤으면 -1, 과일이 바닥에 있지 않으면 0
  - $\gamma = 0.9$

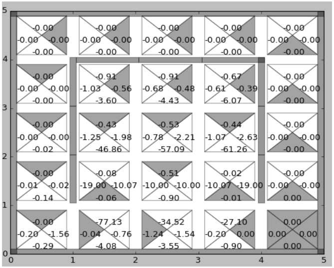


- 3000개의 random episode로 훈련시켜 계산한 Q-table로 테스트 하면 100% 받아냄

Homework: SARSA vs. Q-Learning (Due: TBD)

- 다음과 같이 수정된 maze environment를 고려:
  - state 6,7,8(함정)에 빠지면 reward -100
  - final state인 4에 도달하면 reward 100
  - 그외의 경우는 reward 0
  - 아래 그림과 같이 함정 주위를 벽이 둘러 싸고 있음





- 왼쪽 그림은 Q-learning을 적용했을 때 (최단 경로)
- 오른쪽 그림은 SARSA를 적용했을 때 (안전한 경로)
  - 이렇게 차이가 나는 이유는?
- `maze_SARSA.py`의 SARSA를 구현 (`cs3.ksa@gmail.com`)