Reinforce Learning

Q-learning

소프트웨어 꼰대 강의

노기섭 교수

(kafa46@hongik.ac.kr)

Recap: Taxonomy

Reinforce Learning

Model-free RL

Model-based RL

On-policy (Policy-based)

Learn directly the stochastic

Act by sampling policy

Act by sampling policy
(exploration is built-in)

Eg. Policy Gradient

Off-policy (Value-based)

Learn state (V) or state-action (Q) value

Act by choosing the best action in a state

Exploration is needed
(exploration first, then increase exploitation)

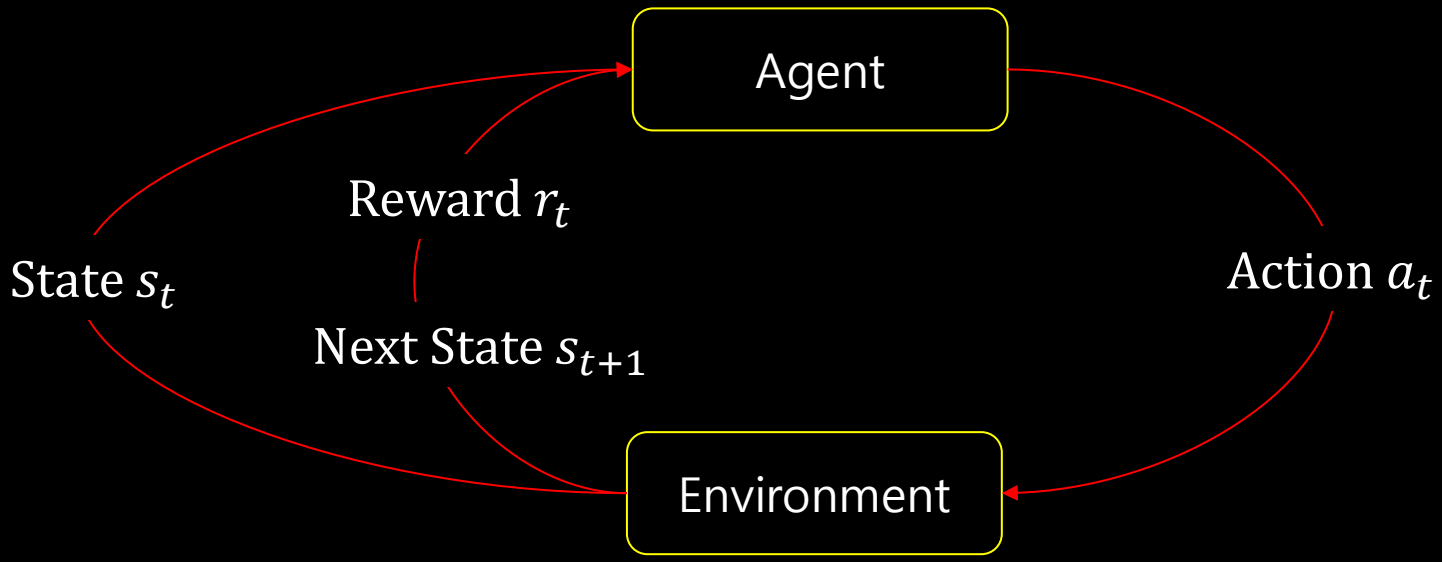
Eg. Q-learning

Define Optimal Policy π^*

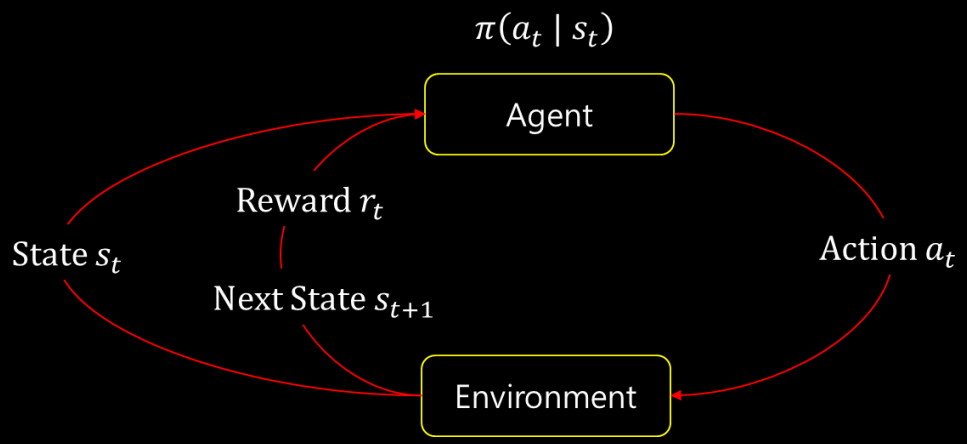
Goal: Learn how to choose action a_t to maximize cumulative rewards

Policy: A function to predict next action
with given state or state-action

$\pi(a_t \mid s_t)$



Formulate Optimal Policy



Cumulative discounted reward

$$R_t = \sum_{t>0} \gamma^t r_t$$

$$, where\ 0 < \gamma < 1$$

Goal: Learn how to choose action a_t to maximize cumulative rewards

Find optimal policy that maximize cumulative discounted reward.

Rewrite using Math,

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right]$$

Applying randomness

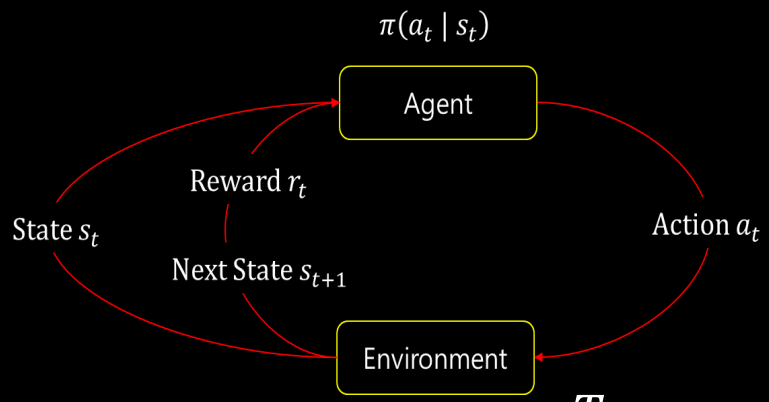
$$s_0 \sim p(s_0)$$
$$a_t \sim \pi(\cdot \mid s_t)$$
$$s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

Stochastic Environment

How to find optimal policy?

Among $\pi(a_t \mid s_t)$, how to find optimal policy $\pi^*(a_t \mid s_t)$?

Let assume π produces sample paths (trajectories)



$$T_1: s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_{n-1}, a_{n-1}, r_n, s_n$$

$$T_2: s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_{n-1}, a_{n-1}, r_n, s_n$$

$$\vdots$$

$$T_n: s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_{n-1}, a_{n-1}, r_n, s_n$$

Rewrite value function with policy π

Keep in mind $\quad \pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right]$

$R(\tau)$ Types

Value Function : $V^\pi(s) = E_{\tau\sim\pi}[R(\tau) \mid s_0 = s\,]$

T: $s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_{n-1}, a_{n-1}, r_n, s_n$

$$V^*(s) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

"이 상태가 얼마나 좋은가?"

Action-Value Function: $Q^\pi(s,a) = E_{\tau\sim\pi}[R(\tau) \mid s_0 = s, a_0 = a]$

T: $s_0, a_0, r_1, s_1, a_1, r_2, \cdots, s_{n-1}, a_{n-1}, r_n, s_n$

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

"이 상태에서 이 행동을 하면 얼마나 좋은가?"

Introducing Bellman Equation

Keep in mind $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid \pi\right]$

$$\pi^* = Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

How to solve this?

Find "Self-consistent Equation (Recursive Formation)"

Why?

By recursive structure, we can apply DP
(Overlapping sub-problem + Optimal sub-structure)

By current estimate, we can bootstrap value function
(Modifying current estimates based on uncertain future estimates)

Thank you very much, prof. Bellman!

The Bellman equation [edit]

$V(x_0)$: Optimal value by maximizing objective.

So far it seems we have only made the problem uglier by separating today's decision from future decisions. But we can simplify by noticing that what is inside the square brackets on the right is *the value* of the time 1 decision problem, starting from state $x_1 = T(x_0, a_0)$.

Therefore, the problem can be rewritten as a recursive definition of the value function:

$$V(x_0) = \max_{a_0} \{F(x_0, a_0) + \beta V(x_1)\}, \text{ subject to the constraints:}$$

$$a_0 \in \Gamma(x_0),\ x_1 = T(x_0, a_0).$$

$T(x, a)$: state changes form x to a new state

This is the Bellman equation. It may be simplified even further if the time subscripts are dropped and the value of the next state is plugged in:

$$V(x) = \max_{a \in \Gamma(x)} \{F(x, a) + \beta V(T(x, a))\}.$$

$F(x, a)$: Current payoff taking action a in state s

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

Rewrite Bellman Eq. using RL notation

$$Q^*(s_0, a_0) = E\left[r + \gamma \max_{a'} Q^*(s', a') \mid s_0, a_0\right]$$

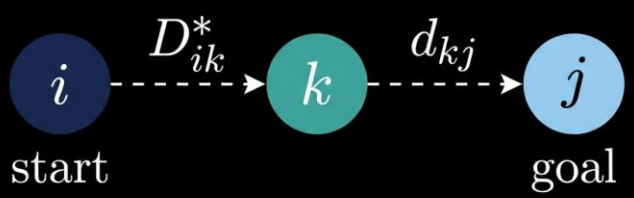
Expectation is taken over stochastic transitions.

Bellman equation in RL is analogous to dynamic programming in shortest path problems.

Dijkstra's Algorithm

$$D^*_{ij} = \min\{D^*_{ik} + d_{kj}\}$$

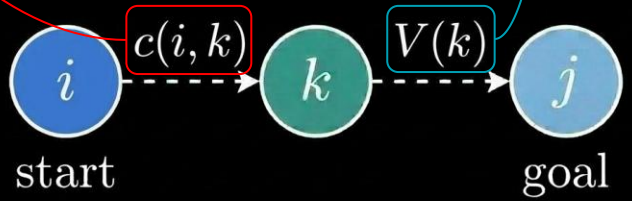
$$D^*_{ij} = \min_k d_{kj}$$



Bellman-Ford Algorithm

$$V(i) = \min_k [c(i,k) + V(k)]$$

Converges via repeated relaxations even with negative edges.



Optimality between Bellman and Reward

$$Q^*(s_0, a_0) = E\left[r + \gamma \max_{a'} Q^*(s', a') \mid s_0, a_0\right]$$

If optimal state-action, (s', a'), value of next step is known, the Bellman equation reduces to:

$$Q^*(s_0, a_0) = r + \gamma Q^*(s', a')$$

Optimal policy chooses the action with highest Q-value:

π^*

- Optimal policy chooses the action that leads to the best future trajectory.
- 지금 상태에서 시작하는 trajectory 중 미래 reward 합이 가장 큰 trajectory를 만들어 줄 첫 행동 선택

$$\pi^* = \arg\max_{\pi} Q^*(s, a)$$

DP vs. TD vs. MC

	Dynamic Programming (DP)	Temporal Difference (TD)	Monte Carlo (MC)
환경 모델	✅ 필요	❌ 필요 없음	❌ 필요 없음
학습 방식	Planning (모델 기반)	Bootstrapping	Sample return
업데이트	전체 sweep	매 step	에피소드 종료 후
Bootstrapping	✅ 사용	✅ 사용	❌ 사용 안 함
Online 학습	❌ 어려움	✅ 가능	⚠️ 제한적
실무 사용	거의 없음	매우 많음	제한적
직관	전체 지도 알고 길 찾기	걸어가며 업데이트	경험 끝까지 보고 평가

Solving RL: Dynamic Programming (DP) (Expectation over transitions)

If the environment model $P(s'|s,a)$ and reward function are known,
we can compute the optimal Q-function using Bellman optimality updates.

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

initialize $Q(s,a)$ arbitrarily

repeat until convergence:

  for each state s:

    for each action a:

      Update: using above Eq.

DP Characteristics

환경의 전이 확률 모델 $P(s' \mid s,a)$ 필요

Updated by Expectation

Perfect in theory,
Intractable in reality!

Solving RL: Q-learning on top of TD

Q-learning Algorithm: Solve optimal Q* using Bellman itratively

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\left(\underbrace{R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a_t)}_{\text{Target}} - \underbrace{Q_t(s_t, a_t)}_{\text{Predict}}\right)$$

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

Choose action a using ε-greedy

repeat:

Take action a

Observe r, s'

Update: using above Eq.

$s \leftarrow s'$

TD Characteristics

환경의 전이 확률 모델 $P(s' \mid s, a)$ 불필요

TD error: $\delta = target - prediction$

TD solves prediction, but not full control in complex environments

To tackle DP,

TD (Q-learning) is a practical solution.

Intractable in DP as well as TD

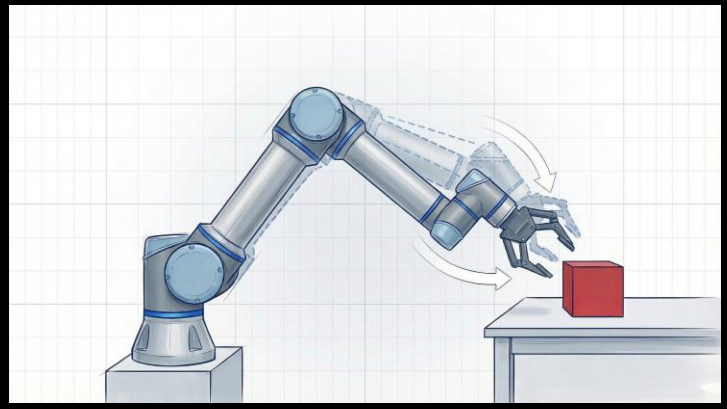
Key Challenges: State Explosion and Continuous Action

State Explosion Problem

- High-dimensional states cause exponential growth in state-action space
- Tabular methods become impractical due to prohibitive memory and computational requirements
- **Function approximation (e.g., neural net) is needed**

Continuous Action Problem

- Infinite action space makes optimization intractable
- Discretization introduces loss of precision or increased complexity

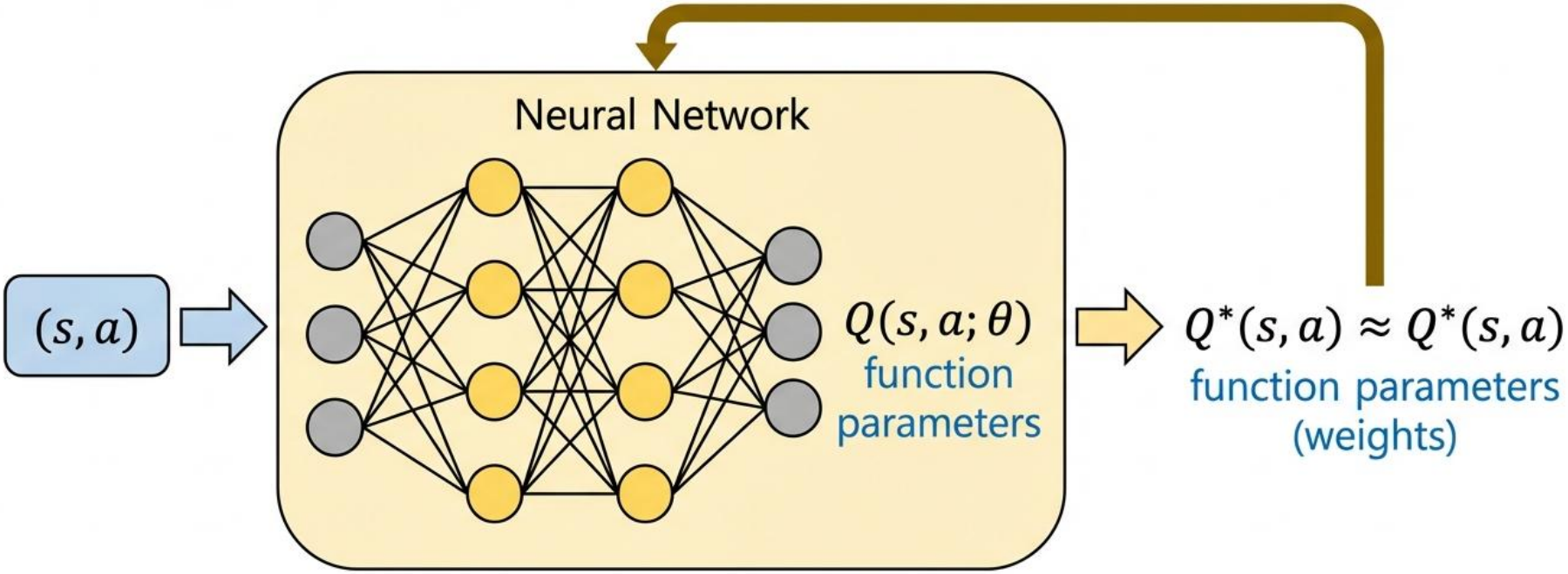


Solution over TD (Q-learning)

Function Approximation!

$Q(s,a)$ modeling function approximator → Q-learning

If function approximator is made with deep neural network → Deep Q Network



수고하셨습니다 ..^^..