

# Algorithms for Reinforcement Learning

---

김응서

February 3, 2026

SNU BI Lab (Seoul National University BioIntelligence Lab)

## **Algorithms for Reinforcement Learning**

**저자:** Csaba Szepesvári

**출판:** Morgan & Claypool Publishers

**시리즈:** Synthesis Lectures on Artificial Intelligence and Machine Learning

**발행일:** June 9, 2009

\* 본 발표는 위 교재를 기반으로 하며, 김응서가 추가적으로 정리한 내용도 포함합니다.

## 1. Overview

## 2. Markov Decision Processes

- Preliminaries
- Markov Decision Processes
- Value functions
- Dynamic programming algorithms for solving MDPs

## 3. Value Prediction Problems

- Temporal difference learning in finite state spaces
  - Tabular TD(0)
  - Every-visit Monte-Carlo
  - $\text{TD}(\lambda)$ : Unifying Monte-Carlo and TD(0)
- Algorithms for large state spaces

## 4. Control

- A catalog of learning problems
- Closed-loop interactive learning
- Direct methods (Q-learning)
- Actor-critic methods

## 5. For Further Exploration

- Further reading
- Applications
- Software

**Appendix:** The theory of discounted Markovian decision processes

## Overview

---

# 강화학습이란?

## Reinforcement Learning (RL)

Learning to control a system so as to **maximize** some numerical value which represents a **long-term objective**;;

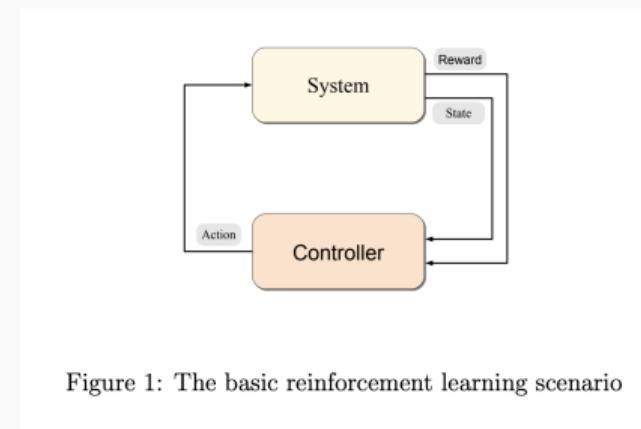


Figure 1: The basic reinforcement learning scenario

*Agent  $\leftrightarrow$  Environment: State, Action, Reward의 순환*

*Sources: Algorithms for Reinforcement Learning; Dayan & Niv (2008); Nature (2015) DQN*

# 주요 간단 용어 (1/3)

**Environment (환경)** Agent가 행동하는 공간

**State (상태)**  $s \in \mathcal{S}$  환경에서 agent가 있을 수 있는 여러 상태 중 하나

**Action (행동)**  $a \in \mathcal{A}$  Agent가 한 상태에서 다른 상태로 전환하기 위해 선택할 수 있는 행동

**Reward (보상)**  $r \in \mathcal{R}$  행동을 취한 후 환경이 피드백으로 제공하는 보상

**Transition Probability (전이 확률)**  $P$  행동 후 어떤 상태에 도달할지 결정하는 확률

## 주요 간단 용어 (2/3)

**Model** 환경이 특정 행동에 어떻게 반응할지 정의

보상 함수와 전이 확률을 포함

**Policy (정책)**  $\pi(s)$  특정 상태에서 최적의 행동을 안내  
총 보상을 최대화하는 것이 목표

**Value Function**  $V(s)$  해당 상태에서 정책을 따를 때 받을 수 있는 미래 보상의 기댓값 예측  
상태가 얼마나 좋은지를 정량화

# 주요 간단 용어 (3/3)

Agent와 environment의 상호작용은 시간에 따른 행동과 관찰된 보상의 **sequence**를 포함:

시간 단계:  $t = 1, 2, \dots, T$

**Notation:**

- $S_t$ : 시간  $t$ 에서의 상태 (state)
- $A_t$ : 시간  $t$ 에서의 행동 (action)
- $R_t$ : 시간  $t$ 에서의 보상 (reward)

## Episode

하나의 완전한 상호작용 시퀀스 (또한 "trial" 또는 "trajectory"로 불림)

Terminal state  $S_T$ 에서 종료:

$$S_1, A_1, R_2, S_2, A_2, \dots, S_T$$

# Model (1/3): Transition and Reward

Model은 환경의 descriptor로, 두 가지 주요 부분으로 구성:

1. Transition Probability Function  $P$
2. Reward Function  $R$

## Transition Step

상태  $s$ 에서 행동  $a$ 를 취해 다음 상태  $s'$ 에 도달하고 보상  $r$ 을 받음

Tuple로 표현:  $(s, a, s', r)$

Transition Function:

$$P(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

$\mathbb{P}$ 는 "probability"를 나타내는 기호

## Model (2/3): State-Transition과 Reward Function

State-Transition Function:

$$P_{ss'}^a = P(s' | s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} P(s', r | s, a)$$

Reward Function:

$$R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r | s, a)$$

- Transition function은 행동  $a$  후 상태  $s$ 에서  $s'$ 로 전환될 확률을 기록
- Reward function은 한 행동에 의해 발생하는 다음 보상을 예측

# Model (3/3): Model-Based vs Model-Free RL

모델을 알고 있는지 여부에 따라 접근 방식이 달라짐:

## Model-Based RL

모델을 알고 있는 경우

- Dynamic Programming (DP) 사용 가능할 수도.

## Model-Free RL

모델을 모르는 경우

- 그러니까 학습 시 모델에 의존적이지 않음.

## 정의

Policy  $\pi$ 는 agent의 행동 함수(behavior function)로, 상태  $s$ 에서 어떤 행동을 취할지 결정

상태  $s$ 에서 행동  $a$ 로의 매핑 (mapping from state to action)

두 가지 유형:

- Deterministic Policy (결정적 정책):

$$\pi(s) = a$$

특정 상태에서 항상 같은 행동을 선택

- Stochastic Policy (확률적 정책):

$$\pi(a \mid s) = \mathbb{P}_\pi[A = a \mid S = s]$$

특정 상태에서 확률 분포에 따라 행동을 선택

# Value Function (1/5)

## 정의

Value function은 상태 또는 행동이 얼마나 좋은지를 미래 보상의 예측으로 측정

미래 보상(future reward)은 **return**이라고도 하며, 할인된 보상의 총합:

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

**Discounting factor:**  $\gamma \in [0, 1]$

## Value Function (2/5): State-Value Function

State-value function  $V_\pi(s)$ : 시간  $t$ 에 상태  $s$ 에 있을 때의 기댓값 return

$$V_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

- 특정 정책  $\pi$ 를 따를 때 상태  $s$ 에서 기대되는 미래 보상의 총량
- "이 상태가 얼마나 좋은가?"에 대한 답

## Value Function (3/5): Action-Value Function (Q-Value)

Action-value function  $Q_\pi(s, a)$ : 상태  $s$ 에서 행동  $a$ 를 취한 후의 기댓값 return

$$Q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

- "Q-value"라고도 불림 (Q는 "Quality"를 의미)
- 특정 상태-행동 쌍의 가치를 평가

State-value와의 관계:

정책  $\pi$ 를 따를 때, Q-value와 가능한 행동들의 확률 분포를 이용해 state-value 복원:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s, a) \pi(a \mid s)$$

## Value Function (4/5): Advantage Function

Advantage function  $A_\pi(s, a)$ : action-value와 state-value의 차이

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

- "A-value"라고도 불림
- 특정 행동이 평균적인 행동보다 얼마나 더 좋은지를 나타냄
- $A_\pi(s, a) > 0$ : 행동  $a$ 가 평균보다 좋음
- $A_\pi(s, a) < 0$ : 행동  $a$ 가 평균보다 나쁨
- $A_\pi(s, a) = 0$ : 행동  $a$ 가 평균 수준

# Value Function (5/5): Optimal Value and Policy

Optimal value function은 최대 return을 생성:

$$V_*(s) = \max_{\pi} V_{\pi}(s), \quad Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

Optimal policy  $\pi_*$ 는 optimal value functions를 달성:

$$\pi_* = \arg \max_{\pi} V_{\pi}(s), \quad \pi_* = \arg \max_{\pi} Q_{\pi}(s, a)$$

당연히, 다음이 성립:

$$V_{\pi_*}(s) = V_*(s) \quad \text{and} \quad Q_{\pi_*}(s, a) = Q_*(s, a)$$

- 모든 상태에서 최대 기댓값 return을 제공하는 정책이 optimal policy
- RL의 목표: optimal policy  $\pi_*$ 를 찾는 것

# **Markov Decision Processes: in an Easy way**

---

# Markov Decision Processes (MDP) (1/2)

거의 모든 RL 문제는 MDP로 표현 가능

MDP의 모든 상태는 "Markov" 속성을 가짐:

## Markov Property

미래는 현재 상태에만 의존하며, 과거 이력에는 의존하지 않음

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

다시 말해, 미래와 과거는 현재 주어진 상황에서 조건부 독립

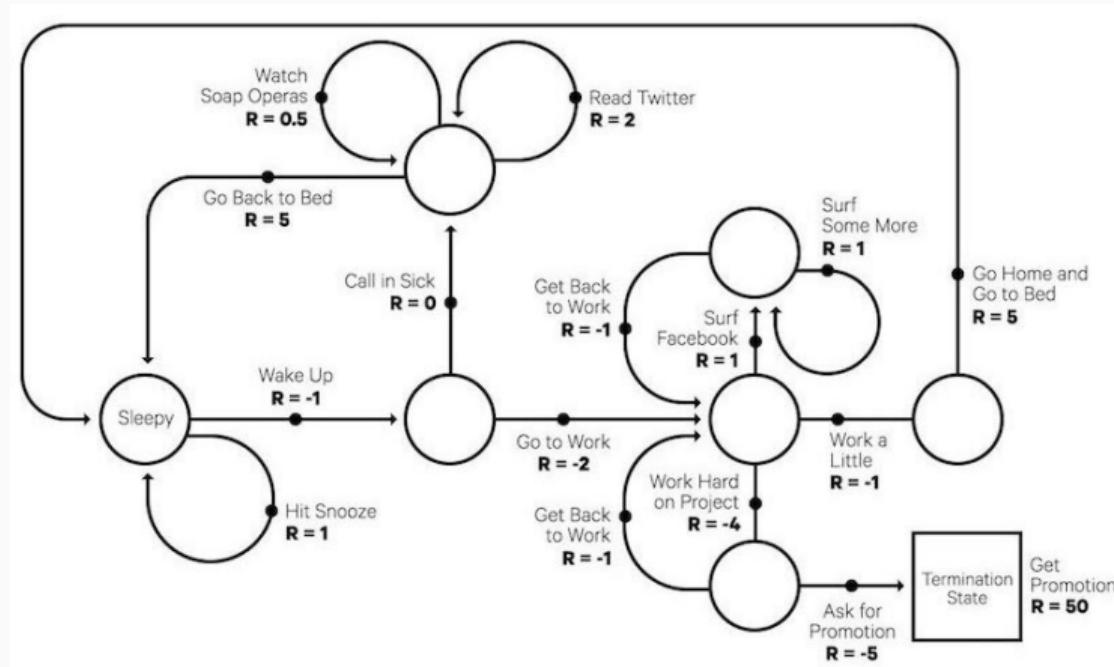
→ 현재 상태가 미래를 결정하는 데 필요한 모든 통계를 포함

## Markov Decision Processes (MDP) (2/2)

MDP는 5개의 요소로 구성:  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$

- $\mathcal{S}$  - 상태의 집합 (a set of states)
- $\mathcal{A}$  - 행동의 집합 (a set of actions)
- $P$  - 전이 확률 함수 (transition probability function)
- $R$  - 보상 함수 (reward function)
- $\gamma$  - 미래 보상에 대한 할인 계수 (discounting factor for future rewards)

# MDP Example: A Typical Work Day



A fun example of Markov decision process: a typical work day. Image source: randomant.net/reinforcement-learning-concepts

# Bellman Equations

**Bellman equations** decompose the value function into **immediate reward** and **discounted future values**:

**State-Value Function:**

$$\begin{aligned} V(s) &= \mathbb{E}[G_t \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s] \end{aligned}$$

**Action-Value Function (Q-value):**

$$\begin{aligned} Q(s, a) &= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}_{a \sim \pi} Q(S_{t+1}, a) \mid S_t = s, A_t = a] \end{aligned}$$

# Built-in Assumptions

**Core Idea:** The optimal value of a state is the best expected return achievable from that state onward.

$$V^\pi(s) = E_\pi \left[ \sum_{t=0}^T \gamma^t r_t \mid s_0 = s \right]$$

$$V^*(s) = \max_a E_{s'} [r(s, a) + \gamma V^*(s')]$$

**Built-in Assumptions:**

- **Markovian** - 미래는 현재 상태에만 의존 (과거 이력 무관)
- **Stationarity** - 환경의 동역학과 보상 함수가 시간에 따라 변하지 않음
- **Scalar rewards** - 보상은 단일 스칼라 값
- **Additivity of rewards** - 보상은 시간에 따라 가산적으로 누적
- **Estimation** - Value function을 근사 가능하다고 가정
- **Uniqueness** - 최적 value function이 존재

*Source: Joelle Pineau: Beyond Bellman's Legacy: Rethinking What we Value*

# **Markov Decision Processes: in a Harder way**

---

# Preliminaries: 표기

## Sets and Spaces:

- $\mathbb{N}$  - Natural numbers:  $\mathbb{N} = \{0, 1, 2, \dots\}$
- $\mathbb{R}$  - Real numbers
- $v$  - Column vector,  $v^\top$  - Row vector

## Inner Product and Norms:

- Inner product:  $\langle u, v \rangle = \sum_{i=1}^d u_i v_i$
- 2-norm:  $\|u\|_2 = \sqrt{\langle u, u \rangle}$
- Max norm:  $\|u\|_\infty = \max_{i=1, \dots, d} |u_i|$
- Function norm:  $\|f\|_\infty = \sup_{x \in X} |f(x)|$

# Markov Decision Processes (MDP): Recap

**Concept:** A framework for modeling sequential decision-making problems where an agent observes states, selects actions, and probabilistically receives next states and rewards.

**An MDP consists of 5 components:**  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$

- $\mathcal{S}$ : set of states,  $\mathcal{A}$ : set of actions
- $P$ : transition probability function  $P(s, a, s') = \Pr(S_{t+1} = s' | S_t = s, A_t = a)$
- $R$ : reward function  $R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma \in [0, 1]$ : discount factor

**Transition dynamics:**

$$S_0 \sim \rho, A_t \sim \pi(\cdot | S_t), S_{t+1} \sim P(\cdot | S_t, A_t), R_{t+1} \sim R(S_t, A_t)$$

**Objective:** Maximize expected discounted cumulative reward

$$G = \sum_{t=0}^{\infty} \gamma^t R_{t+1}, \quad \pi^* \in \arg \max_{\pi} \mathbb{E}_{\pi}[G]$$

# Value Functions: Motivation

Why value functions?

**Naive approach:** List all possible behaviors and identify optimal ones for each initial state

- The number of deterministic policies:  $|\Pi| = |\mathcal{A}|^{|\mathcal{S}|}$ <sup>1</sup>
- Problem: **exponential complexity** in the number of states → computationally infeasible

**Better approach:** Compute value functions

1. First, compute the **optimal value function**
2. Then, derive an optimal policy with relative ease

---

<sup>1</sup> <https://math.stackexchange.com/questions/4449548>

## Stationary Policies (1/2)

**Deterministic stationary policy:** A mapping  $\pi : \mathcal{S} \rightarrow \mathcal{A}$

$$A_t = \pi(S_t), \quad \forall t \geq 0$$

**Stochastic stationary policy:** Maps states to distributions over actions

$$A_t \sim \pi(\cdot | S_t), \quad \forall t \geq 0$$

where  $\pi(a|s)$  denotes the probability of selecting action  $a$  in state  $s$ .

**Notation:**

- $\Pi_{\text{stat}}$  denotes the set of all stationary policies
- In what follows, "policy" means "stationary policy" unless stated otherwise

## Stationary Policies (2/2): Markov Reward Process

**Markov Reward Process (MRP):** Induced by a stationary policy  $\pi$  and MDP  $\mathcal{M}$

**Definition:** Determined by  $(\mathcal{S}, P^\pi)$  where

$$P^\pi(\cdot|s) = \sum_{a \in \mathcal{A}} \pi(a|s)P(\cdot|s, a)$$

**Key properties:**

- The state process  $(S_t; t \geq 0)$  becomes a time-homogeneous Markov chain
- Generates stochastic process  $((S_t, R_{t+1}); t \geq 0)$
- Simplifies analysis by removing action selection decisions

## Value Functions (1/2): Policy Value Functions

Fix a policy  $\pi \in \Pi_{\text{stat}}$ . Assume  $\mathbb{P}(S_0 = s) > 0$  for all  $s \in \mathcal{S}$ .

**State-value function**  $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$  underlying policy  $\pi$ :

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s \right], \quad s \in \mathcal{S}$$

**Action-value function**  $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  underlying policy  $\pi$ :

$$Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = s, A_0 = a \right]$$

where  $(s, a) \in \mathcal{S} \times \mathcal{A}$ .

## Value Functions (2/2): Optimal Value Functions

Optimal state-value function  $V_* : \mathcal{S} \rightarrow \mathbb{R}$ :

$$V_*(s) = \sup_{\pi \in \Pi_{\text{stat}}} V^\pi(s), \quad s \in \mathcal{S}$$

Optimal action-value function  $Q_* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ :

$$Q_*(s, a) = \sup_{\pi \in \Pi_{\text{stat}}} Q^\pi(s, a), \quad (s, a) \in \mathcal{S} \times \mathcal{A}$$

Interpretation:

- $V_*(s)$ : highest expected return achievable from state  $s$
- $Q_*(s, a)$ : highest expected return when starting from  $s$  and taking action  $a$  first

# Optimal Value Functions: Properties

Connection between  $V_*$  and  $Q_*$ :

$$V_*(s) = \sup_{a \in \mathcal{A}} Q_*(s, a), \quad s \in \mathcal{S}$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') V_*(s'), \quad (s, a) \in \mathcal{S} \times \mathcal{A}$$

Optimal policy existence:

- An optimal stationary policy always exists in the class of MDPs considered
- We have:  $V_*(s) = \sup_{\pi \in \Pi_{\text{stat}}} V^\pi(s)$  for all  $s \in \mathcal{S}$

# Optimal Policies and Greedy Actions

**Characterization of optimal policies:**

A policy  $\pi \in \Pi_{\text{stat}}$  is optimal if and only if

$$\sum_{a \in \mathcal{A}} \pi(a|s) Q_*(s, a) = V_*(s), \quad \forall s \in \mathcal{S}$$

**Greedy action:** An action  $a$  is *greedy* w.r.t.  $Q$  in state  $s$  if

$$a \in \arg \max_{a' \in \mathcal{A}} Q(s, a')$$

**Key result:** A greedy policy w.r.t.  $Q_*$  is optimal

- Knowing  $Q_*$  alone is sufficient for finding an optimal policy
- Knowing  $V_*$ ,  $R$ , and  $P$  also suffices to act optimally

# Bellman Equations for Policy Evaluation (1/2)

**Goal:** Compute  $V^\pi$  for a given policy  $\pi$

**Bellman equation for  $V^\pi$**  (deterministic policy):

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s, \pi(s), s') V^\pi(s'), \quad \forall s \in \mathcal{S}$$

**Bellman operator  $T^\pi$**  :  $\mathbb{R}^{\mathcal{S}} \rightarrow \mathbb{R}^{\mathcal{S}}$ :

$$(T^\pi V)(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s, \pi(s), s') V(s')$$

Compact form:  $T^\pi V^\pi = V^\pi$  (fixed-point equation)

## Bellman Equations for Policy Evaluation (2/2)

Properties of  $T^\pi$ :

- $T^\pi$  is an **affine linear operator**
- If  $0 < \gamma < 1$ , then  $T^\pi$  is a **maximum-norm contraction**
- The fixed-point equation  $T^\pi V = V$  has a **unique solution**

Finite state space:

For  $|\mathcal{S}| = D$ , we can write

$$V^\pi = (I - \gamma P^\pi)^{-1} r^\pi$$

where  $r^\pi \in \mathbb{R}^D$  and  $P^\pi \in \mathbb{R}^{D \times D}$  are appropriately defined.

# Bellman Optimality Equations (1/2)

Bellman optimality equation for  $V_*$ :

$$V_*(s) = \sup_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') V_*(s') \right\}, \quad \forall s \in \mathcal{S}$$

Bellman optimality operator  $T_* : \mathbb{R}^{\mathcal{S}} \rightarrow \mathbb{R}^{\mathcal{S}}$ :

$$(T_* V)(s) = \sup_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') V(s') \right\}$$

Compact form:  $T_* V_* = V_*$  (fixed-point equation)

## Bellman Optimality Equations (2/2)

Properties of  $T_*$ :

- $T_*$  is a **nonlinear** operator (due to sup)
- If  $0 < \gamma < 1$ , then  $T_*$  is a **maximum-norm contraction**
- The fixed-point equation  $T_* V = V$  has a **unique solution**

Bellman optimality equation for  $Q_*$ :

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') \sup_{a' \in \mathcal{A}} Q_*(s', a')$$

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

# Value Iteration (VI)

**Key idea:** Instead of enumerating policies, directly update the value function  $V$

**Update rule:**

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') V_k(s') \right)$$

**Convergence property:**

- With sufficient iterations:  $V_k \rightarrow V^*$  (optimal value function)
- Guaranteed by contraction mapping theorem (when  $0 < \gamma < 1$ )

**Extracting optimal policy:** After convergence, use greedy action selection:

$$\pi^*(s) \in \arg \max_{a \in \mathcal{A}} \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s, a, s') V^*(s') \right)$$

# Value Iteration: Convergence Rate

**Geometric convergence:** By Banach's fixed-point theorem,

The sequence  $V_{k+1} = T_* V_k$  converges to  $V^*$  at a **geometric rate**:

$$\|V_k - V^*\|_\infty \leq \gamma^k \|V_0 - V^*\|_\infty$$

**Q-value version:** VI can also be applied to action-value functions:

$$Q_{k+1} = T_* Q_k, \quad k \geq 0$$

which also converges to  $Q^*$  at a geometric rate.

**Approximate optimality guarantee:** For greedy policy  $\pi$  w.r.t.  $Q$ :

$$V^\pi(s) \geq V^*(s) - \frac{2}{1-\gamma} \|Q - Q^*\|_\infty, \quad \forall s \in \mathcal{S}$$

This means if  $Q_k \approx Q^*$ , then a greedy policy w.r.t.  $Q_k$  is near-optimal.

## VI: Iteration Complexity (1/2)

**Question:** How many iterations to achieve  $\varepsilon$ -accuracy?

**Contraction property:**  $T_*$  is a  $\gamma$ -contraction in sup-norm:

$$\|T_* U - T_* V\|_\infty \leq \gamma \|U - V\|_\infty$$

여기서  $U, V : \mathcal{S} \rightarrow \mathbb{R}$ 는 임의의 value function 후보들이고,

$$\|U - V\|_\infty = \max_{s \in \mathcal{S}} |U(s) - V(s)| \text{ (최대 차이).}$$

즉, 임의의 두 value function에  $T_*$ 를 적용하면 그 최대 차이가  $\gamma$ 배 이하로 줄어든단 소리.

Error bound after  $k$  iterations:

$$\|V_k - V^*\|_\infty \leq \gamma^k \|V_0 - V^*\|_\infty$$

This shows the error decreases by a factor of  $\gamma$  at each iteration.

## VI: Iteration Complexity (2/2)

**Iteration count for  $\varepsilon$ -accuracy:** To achieve  $\|V_k - V^*\|_\infty \leq \varepsilon$ :

$$\gamma^k \|V_0 - V^*\|_\infty \leq \varepsilon \quad \Rightarrow \quad k \geq \frac{\log(\|V_0 - V^*\|_\infty / \varepsilon)}{-\log \gamma}$$

Since  $-\log \gamma \approx 1 - \gamma$  when  $\gamma \rightarrow 1$ :

$$K(\varepsilon) = O\left(\frac{\log(1/\varepsilon)}{1 - \gamma}\right)$$

**Interpretation:**

- **Logarithmic in  $1/\varepsilon$ :** Very fast convergence!
- **Linear in  $1/(1 - \gamma)$ :** Slower when  $\gamma$  is close to 1
- Standard result in RL/MDP literature

## VI: Total Computational Cost

Total cost = (Cost per iteration)  $\times$  (Number of iterations)

Cost per iteration:

- Dense transitions:  $O(|\mathcal{S}|^2 |\mathcal{A}|)$
- Sparse transitions (branching factor  $d$ ):  $O(|\mathcal{S}| |\mathcal{A}| d)$

Number of iterations for  $\varepsilon$ -accuracy:  $K(\varepsilon) = O\left(\frac{\log(1/\varepsilon)}{1-\gamma}\right)$

Total computational complexity:

- Dense:  $\tilde{O}\left(\frac{|\mathcal{S}|^2 |\mathcal{A}|}{1-\gamma}\right)$  to reach  $\varepsilon$ -accuracy
- Sparse:  $\tilde{O}\left(\frac{|\mathcal{S}| |\mathcal{A}| d}{1-\gamma}\right)$  to reach  $\varepsilon$ -accuracy

(The  $\tilde{O}$  notation hides logarithmic factors in  $1/\varepsilon$ )

## VI Complexity (1/2): Dense Transitions

**Transition probability:**  $P(s, a, s') = \Pr(S_{t+1} = s' \mid S_t = s, A_t = a)$

**Dense transition:** For many  $(s, a)$ , **many**  $s'$  have non-zero probability

When computing the expectation:

$$\sum_{s' \in \mathcal{S}} P(s, a, s') V(s')$$

we must sum over nearly all  $s' \in \mathcal{S}$  since  $P(s, a, s') \neq 0$  for most  $s'$ .

**Computational cost:**

- Per  $(s, a)$  pair:  $O(|\mathcal{S}|)$  operations
- Total states:  $|\mathcal{S}|$ , total actions:  $|\mathcal{A}|$
- One VI iteration:  $O(|\mathcal{S}|^2 |\mathcal{A}|)$

## VI Complexity (2/2): Sparse Transitions

**Sparse transition:** For each  $(s, a)$ , reachable next states are limited to a few

**Definition of  $d$ :** Number of non-zero transition probabilities

$$d = \left| \{s' \in \mathcal{S} : P(s, a, s') > 0\} \right| \quad \text{where } d \ll |\mathcal{S}|$$

When computing the expectation:

$$\sum_{s' \in \mathcal{S}} P(s, a, s') V(s')$$

we only need to sum over  $d$  terms (non-zero probabilities).

**Computational cost:**

- Per  $(s, a)$  pair:  $O(d)$  operations
- One VI iteration:  $O(|\mathcal{S}| |\mathcal{A}| d)$
- Much faster when  $d \ll |\mathcal{S}|$

# Policy Iteration (PI)

**Alternative approach:** Alternate between policy evaluation and policy improvement

**Algorithm:** Start with arbitrary policy  $\pi_0$ . At iteration  $k \geq 0$ :

1. **Policy Evaluation:** Compute  $Q^{\pi_k}$  (or  $V^{\pi_k}$ ) by solving:

$$Q^{\pi_k} = T^{\pi_k} Q^{\pi_k}$$

2. **Policy Improvement:** Define greedy policy w.r.t.  $Q^{\pi_k}$ :

$$\pi_{k+1}(s) \in \arg \max_{a \in \mathcal{A}} Q^{\pi_k}(s, a)$$

**Key property:** Each iteration is guaranteed to improve (or maintain) the policy:

$$V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s), \quad \forall s \in \mathcal{S}$$

# Value Iteration vs. Policy Iteration

Aspect	Value Iteration	Policy Iteration
Update	$V_{k+1} = T_* V_k$	Solve $V^{\pi_k} = T^{\pi_k} V^{\pi_k}$
Cost per step	Low: single sweep	High: solve linear system
Convergence	Geometric rate	Fewer iterations
Policy quality	Implicit (via greedy)	Explicit improvement

Trade-off:

- VI: Fast per iteration, but may need many iterations
- PI: Slow per iteration (policy evaluation), but converges in fewer iterations
- PI gives a policy not worse than VI after  $k$  iterations (if started with same  $V_0$ )

## A. The Theory of Discounted Markovian Decision Processes

---

## A.1. Contractions and Banach's fixed-point theorem

---

## Definition 1 (Norm)

정의: 실수 위의 벡터 공간  $V$ 에 대해, 함수  $f : V \rightarrow \mathbb{R}_0^+$ 가 다음 조건들을 만족하면  $V$  위의 **norm**이라고 한다:

1. 어떤  $v \in V$ 에 대해  $f(v) = 0$ 이면  $v = 0$ 이다;
2. 임의의  $\lambda \in \mathbb{R}$ ,  $v \in V$ 에 대해  $f(\lambda v) = |\lambda|f(v);$
3. 임의의  $v, u \in V$ 에 대해  $f(v + u) \leq f(v) + f(u).$

여기서  $f$  또는 Norm이 정의된 벡터 공간을 **normed vector space**라고 한다.

참고: Norm은 각 벡터에 음이 아닌 실수를 할당하는 함수이다. 이 값을 벡터의 "길이(length)" 또는 "norm"이라고 부르며, 벡터  $v$ 의 norm을  $\|v\|$ 로 표기한다.

## Example: Norm의 예시 (1)

벡터 공간  $V = (\mathbb{R}^d, +, \lambda \cdot)$  위의 norm들:

1.  $\ell_p$  norm:  $p \geq 1$ 에 대해,

$$\|v\|_p = \left( \sum_{i=1}^d |v_i|^p \right)^{1/p}$$

$p = 2$ 일 때 Euclidean distance,  $p = 1$ 일 때 Manhattan distance

2.  $\ell_\infty$  norm:

$$\|v\|_\infty = \max_{1 \leq i \leq d} |v_i|$$

최대 성분 값으로 정의되는 Chebyshev distance (또는 supremum norm)

쉽게는, 각 성분의 절댓값을 전부 보고, 그중에서 가장 큰 값 하나를 고른다는 소리

여기서  $d$ 는 벡터 공간의 차원(dimension)을 나타냄

## Example: Norm의 예시 (2)

3. Weighted norms:

$$\|v\|_p = \begin{cases} \left( \sum_{i=1}^d \frac{|v_i|^p}{w_i} \right)^{1/p}, & \text{if } 1 \leq p < \infty; \\ \max_{1 \leq i \leq d} \frac{|v_i|}{w_i}, & \text{if } p = \infty, \end{cases}$$

여기서  $w_i > 0$ . 각 차원에 서로 다른 가중치를 부여하여 중요도를 조절

4. Matrix-weighted 2-norm:

$$\|v\|_{2,P} = \sqrt{v^T P v}$$

여기서  $P$ 는 고정된 positive definite matrix이다. ( $P$ 는 대칭행렬 + 모든 고유값이 양수)

Positive definite matrix로 각 차원 간 상관관계를 반영한 거리

Matrix-weighted 2-norm은 행렬  $P$ 로 공간을 변형한 뒤, 그 변형된 공간에서 벡터의 길이를 재는 norm이다.

# 함수 공간에서의 Norm

벡터 공간뿐만 아니라 함수 공간에서도 norm을 정의할 수 있다.

예시:  $V$ 가 정의역  $X$  위에서 bounded인 함수들의 벡터 공간이면,

$$\|f\|_{\infty} = \sup_{x \in X} |f(x)|$$

참고: 함수  $f$ 가 bounded인 것은 정확히  $\|f\|_{\infty} < +\infty$ 를 의미한다.

## Definition 2 (Convergence in norm)

정의:  $\mathcal{V} = (V, \|\cdot\|)$ 을 normed vector space라 하고,  $v_n \in V$ 를 벡터들의 수열( $n \in \mathbb{N}$ )이라 하자.

수열  $(v_n; n \geq 0)$ 이 벡터  $v$ 로 노름  $\|\cdot\|$ 에서 수렴한다(**converge to  $v$  in the norm**)는 것은 다음을 의미한다:

$$\lim_{n \rightarrow \infty} \|v_n - v\| = 0$$

이를  $v_n \rightarrow \|\cdot\| v$ 로 표기한다.

참고:  $d$ -차원 벡터 공간에서  $v_n \rightarrow \|\cdot\| v$ 는 모든 성분이 수렴하는 것, 즉 각  $1 \leq i \leq d$ 에 대해  $v_{n,i} \rightarrow v_i$ 가 성립하는 것과 같다. (단,  $v_{n,i}$ 는  $v_n$ 의  $i$ 번째 성분)

# 무한차원 벡터 공간에서의 수렴

주의: 무한차원 벡터 공간에서는 pointwise convergence와 norm convergence가 다르다!

예시:  $X = [0, 1]$ 이고  $X$  위의 bounded 함수 공간을 고려하자. 다음과 같이 정의하면:

$$f_n(x) = \begin{cases} 1, & \text{if } x < 1/n; \\ 0, & \text{otherwise.} \end{cases}$$

그리고  $f$ 를 다음과 같이 정의하자:

$$f(x) = \begin{cases} 1, & \text{if } x = 0; \\ 0, & \text{if } x \neq 0. \end{cases}$$

# Pointwise vs Norm Convergence

앞의 예시에서:

**Pointwise convergence:** 각  $x$ 에 대해  $f_n(x) \rightarrow f(x)$

- 즉,  $f_n$ 은  $f$ 로 pointwise하게 수렴한다

그러나, **Norm convergence**는 성립하지 않는다:

$$\|f_n - f\|_{\infty} = 1 \not\rightarrow 0$$

⇒ 무한차원 공간에서는 pointwise convergence가 norm convergence를 함의하지 않는다!

## 추가 예시: 박스 이동

함수 정의:

$$f_n(x) = \begin{cases} 1, & \text{if } n \leq x \leq n+1; \\ 0, & \text{otherwise.} \end{cases}$$

높이 1짜리 박스가 오른쪽으로 계속 이동

왜 pointwise 수렴?

- $x$ 를 하나 고정하면, 어느 순간부터 박스가  $x$ 를 지나쳐감
- 이후로는 계속 0:  $f_n(x) \rightarrow 0$
- 모든  $x$ 에서 0으로 감

Pointwise convergence to 0

# 박스 이동: Norm Convergence는?

앞의 예시에서:

Norm convergence는 성립하지 않는다:

$$\|f_n - 0\|_{\infty} = \|f_n\|_{\infty} = \sup_x |f_n(x)| = 1 \not\rightarrow 0$$

왜?

- 각  $n$ 마다 함수  $f_n$ 은 어딘가( $[n, n + 1]$  구간)에서 여전히 1의 값을 가짐
- Supremum norm은 함수가 가질 수 있는 최댓값을 측정
- 박스가 오른쪽으로 계속 이동해도, 높이는 항상 1로 유지됨

Norm convergence 실패

# Cauchy Sequence

실수 수열 ( $a_n : n \geq 0$ )의 경우, 극한값을 모르더라도 수렴성을 검증할 수 있다.

**Cauchy sequence:** 다음을 만족하는 수열

$$\lim_{n \rightarrow \infty} \sup_{m \geq n} |a_n - a_m| = 0$$

("진동이 사라지는 수열"이라는 이름이 더 직관적일 수 있다)

**실수의 중요한 성질:** 모든 실수의 Cauchy sequence는 극한을 가진다.

Cauchy sequence 개념은 normed vector space로 자연스럽게 확장된다.

## Definition 3 (Cauchy sequence)

정의:  $(v_n; n \geq 0)$ 을 normed vector space  $\mathcal{V} = (V, \| \cdot \|)$ 의 벡터들의 수열이라 하자.

$v_n$ 이 Cauchy sequence라는 것은 다음을 의미한다:

$$\lim_{n \rightarrow \infty} \sup_{m \geq n} \|v_n - v_m\| = 0$$

중요한 사실: 모든 Cauchy sequence가 수렴하는 normed vector space는 특별하다!

일부 Cauchy sequence가 극한을 가지지 않는 normed vector space의 예시도 존재한다.

## Definition 4 (Completeness)

정의: Normed vector space  $\mathcal{V}$ 가 **complete**라는 것은 다음을 의미한다:

Every Cauchy sequence in  $\mathcal{V}$  converges with respect to the norm of the vector space.

의미:

- Completeness는 "빈 곳이 없다"는 느낌(?)
- 해석학과 최적화 이론에서 매우 중요한 성질

## Definition 5 (Banach space)

정의: Complete normed vector space를 **Banach space**라고 한다.

**Banach space**의 예시:

- $(\mathbb{R}^d, \|\cdot\|_p)$  (모든  $p \geq 1$ 에 대해)
- Bounded function space  $(B(X), \|\cdot\|_\infty)$
- $L^p$  spaces (이건 뭔지 모르겠네..)

## Definition 6 (Lipschitz, Contraction)

정의:  $\mathcal{V} = (V, \|\cdot\|)$ 은 normed vector space라 하자. Mapping  $T : V \rightarrow V$ 에 대해:

1. **L-Lipschitz**: 임의의  $u, v \in V$ 에 대해

$$\|Tu - Tv\| \leq L\|u - v\|$$

두 점 사이의 거리를 최대  $L$ 배까지만 변화시킴

2. **Non-expansion**:  $L \leq 1$ 인 Lipschitz mapping 거리를 늘리지 않음 (유지하거나 줄임)

3. **Contraction**:  $L < 1$ 인 Lipschitz mapping 거리를 반드시 줄임! 이때  $L$ 을 **contraction factor**라 하고,  $T$ 를  $L$ -contraction이라 함

# Lipschitz Mapping의 연속성

중요한 성질:  $T$ 가 Lipschitz이면 연속(continuous)이다.

즉,  $v_n \rightarrow \| \cdot \|_v$ 이면  $Tv_n \rightarrow \| \cdot \|_T v$ 도 성립한다.

증명:

$$\|Tv_n - Tv\| \leq L\|v_n - v\| \rightarrow 0 \text{ as } n \rightarrow \infty$$

입력이 가까우면 출력도 가까워진다는 의미

## Definition 7 (Fixed Point)

정의:  $T : V \rightarrow V$ 를 어떤 mapping이라 하자.

벡터  $v \in V$ 가  $T$ 의 **fixed point**라는 것은 다음을 의미한다:

$$Tv = v$$

의미:

- Fixed point는 mapping  $T$ 를 적용해도 변하지 않는 점
- 일종의 "평형점" 또는 "불동점"
- RL에서는 optimal value function이나 Bellman operator의 fixed point가 됨

# Theorem 1 (Banach's Fixed-Point Theorem)

정리:  $V$ 를 Banach space라 하고,  $T : V \rightarrow V$ 를 contraction mapping이라 하자.

그러면:

1.  $T$ 는 유일한(unique) fixed point를 가진다
2. 임의의  $v_0 \in V$ 에 대해,  $v_{n+1} = Tv_n$ 으로 정의하면

$$v_n \rightarrow \| \cdot \|^{v^*}$$

여기서  $v^*$ 는  $T$ 의 유일한 fixed point

3. 수렴은 기하급수적(geometric)이다:

$$\|v_n - v^*\| \leq \gamma^n \|v_0 - v^*\|$$

의미: Contraction을 반복 적용하면 항상 유일한 fixed point로 빠르게 수렴!

# 증명 개요

증명 전략:  $T$ 를  $\gamma$ -contraction이라 하자 ( $\gamma < 1$ ).

**Step 1:** 수열  $(v_n)$ 이 수렴함을 보이기

- $(v_n)$ 이 Cauchy sequence임을 보임
- Banach space는 complete이므로 수렴

**Step 2:** 극한  $v^*$ 가 fixed point임을 보이기

- $v_{n+1} = Tv_n$ 의 양변에 극한을 취함

**Step 3:** Fixed point의 uniqueness 증명

- 두 개의 fixed point가 있다고 가정하면 모순

**Step 4:** Geometric convergence 증명

# 증명 Part 1: Cauchy Sequence

목표:  $(v_n)$ 이 Cauchy sequence임을 보이기

임의의  $k \geq 0$ 에 대해:

$$\begin{aligned}\|v_{n+k} - v_n\| &= \|Tv_{n-1+k} - Tv_{n-1}\| \\ &\leq \gamma \|v_{n-1+k} - v_{n-1}\| \\ &= \gamma \|Tv_{n-2+k} - Tv_{n-2}\| \\ &\leq \gamma^2 \|v_{n-2+k} - v_{n-2}\| \\ &\quad \vdots \\ &\leq \gamma^n \|v_k - v_0\|\end{aligned}$$

이제  $\|v_k - v_0\|$ 을 bound해야 함...

## 증명 Part 2: Cauchy Sequence (계속)

삼각부등식을 이용하면:

$$\|v_k - v_0\| \leq \|v_k - v_{k-1}\| + \|v_{k-1} - v_{k-2}\| + \cdots + \|v_1 - v_0\|$$

각 항에 대해:  $\|v_i - v_{i-1}\| \leq \gamma^{i-1} \|v_1 - v_0\|$

따라서:

$$\begin{aligned}\|v_k - v_0\| &\leq (\gamma^{k-1} + \gamma^{k-2} + \cdots + 1) \|v_1 - v_0\| \\ &\leq \frac{1}{1-\gamma} \|v_1 - v_0\|\end{aligned}$$

결론:

$$\|v_{n+k} - v_n\| \leq \gamma^n \cdot \frac{1}{1-\gamma} \|v_1 - v_0\| \rightarrow 0 \text{ as } n \rightarrow \infty$$

$(v_n)$ 은 Cauchy sequence! Banach space이므로 수렴. 극한을  $v^*$ 라 하자.

## 증명 Part 3: Fixed Point & Uniqueness

$v^*$ 가 fixed point임:

$v_{n+1} = Tv_n$ 의 양변에 극한을 취하면:

- 좌변:  $v_{n+1} \rightarrow \|\cdot\| v^*$
- 우변:  $Tv_n \rightarrow \|\cdot\| Tv^*$  (contraction은 연속)

따라서  $v^* = Tv^*$

Uniqueness:  $v, v'$ 가 모두 fixed point라 가정하면:

$$\begin{aligned}\|v - v'\| &= \|Tv - Tv'\| \\ &\leq \gamma \|v - v'\|\end{aligned}$$

$$(1 - \gamma) \|v - v'\| \leq 0$$
이고  $\gamma < 1$ 이므로  $\|v - v'\| = 0$

따라서  $v = v'$

## 증명 Part 4: Geometric Convergence

목표:  $\|v_n - v^*\| \leq \gamma^n \|v_0 - v^*\|$  를 보이기

$$\begin{aligned}\|v_n - v^*\| &= \|Tv_{n-1} - Tv^*\| \quad (\because v^* = Tv^*) \\ &\leq \gamma \|v_{n-1} - v^*\| \\ &= \gamma \|Tv_{n-2} - Tv^*\| \\ &\leq \gamma^2 \|v_{n-2} - v^*\| \\ &\quad \vdots \\ &\leq \gamma^n \|v_0 - v^*\|\end{aligned}$$

결론:  $\gamma < 1$ 으로  $\gamma^n \rightarrow 0$  exponentially fast!

오차가 기하급수적으로 감소  $\Rightarrow$  매우 빠른 수렴

# Banach Fixed-Point Theorem의 의의

이 정리는 세 가지를 모두 보장한다:

1. **Existence:** Fixed point가 존재한다
2. **Uniqueness:** Fixed point가 유일하다
3. **계산 방법 (Computation):** 임의의 점에서 시작해서  $v_{n+1} = T v_n$ 을 반복하면 fixed point로 수렴한다

RL에서의 중요성:

- Bellman operator가 contraction임을 보이면
- Value iteration, Policy iteration 등의 알고리즘이 수렴함을 보장
- Optimal value function의 existence와 uniqueness 보장

## A.2. Application to MDPs

---

# Optimal Value Function 정의

이 섹션에서는  $V^*$ 를 다음과 같이 정의한다:

$$V^*(s) = \sup_{\pi \in \Pi_{\text{stat}}} V^\pi(s), \quad s \in S$$

의미:

- $V^*(s)$ 는 stationary policy를 선택하여 달성할 수 있는 값의 상한

# Bounded Function Space $B(S)$

정의:  $B(S)$ 를 정의역이  $S$ 인 bounded 함수들의 공간이라 하자:

$$B(S) = \{V : S \rightarrow \mathbb{R} : \|V\|_{\infty} < +\infty\}$$

(강화학습 문맥에서는  $V$ 를 value function으로 봐도 무방)

앞으로  $B(S)$ 를 norm  $\|\cdot\|_{\infty}$ 가 주어진 normed vector space로 볼 것이다.

중요한 성질:  $(B(S), \|\cdot\|_{\infty})$ 는 **complete**이다!

즉,  $B(S)$ 는 Banach space이다.

# $B(S)$ 의 Completeness 증명

목표:  $(V_n; n \geq 0)$ 이  $B(S)$ 의 Cauchy sequence이면 수렴함을 보이기

증명:

- 임의의  $s \in S$ 에 대해,  $(V_n(s); n \geq 0)$ 은 실수의 Cauchy sequence
- 실수는 complete이므로  $V_n(s) \rightarrow V(s)$ 로 수렴
- $V(s)$ 를 각  $s$ 의 극한으로 정의하면  $\|V_n - V\|_\infty \rightarrow 0$

직관:

- $(V_n)$  norm  $\|\cdot\|_\infty$ 에서 Cauchy sequence이므로
- $V_n(s)$ 가  $V(s)$ 로 수렴하는 속도가  $s$ 에 무관
- 따라서 uniform convergence 보장

# Bellman Operator $T^\pi$ 의 Recap

임의의 stationary policy  $\pi$ 에 대해, Bellman operator  $T^\pi : B(S) \rightarrow B(S)$ 을 다음과 같이 정의:

$$(T^\pi V)(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} P(s, \pi(s), s')V(s'), \quad s \in S$$

**Well-definedness:**  $U \in B(S)$ 이면  $T^\pi U \in B(S)$ 도 성립

핵심 성질:

1.  $V^\pi$ 는  $T^\pi$ 의 fixed point
2.  $T^\pi$ 는  $\gamma$ -contraction

# $V^\pi$ 는 $T^\pi$ 의 Fixed Point

식 (7)로 정의된  $V^\pi$ 는  $T^\pi$ 의 fixed point이다:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R_1 | S_0 = s] + \gamma \sum_{s' \in S} P(s, \pi(s), s') \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+2} \mid S_1 = s' \right] \\ &= T^\pi V^\pi(s) \end{aligned}$$

의미:

- Immediate reward + Discounted future value
- Bellman equation의 해

# $T^\pi$ 는 Contraction

정리:  $T^\pi$ 는  $\gamma$ -contraction이다.

증명:

$$\begin{aligned}\|T^\pi U - T^\pi V\|_\infty &= \gamma \sup_{s \in S} \left| \sum_{s' \in S} P(s, \pi(s), s') (U(s') - V(s')) \right| \\ &\leq \gamma \sup_{s \in S} \sum_{s' \in S} P(s, \pi(s), s') |U(s') - V(s')| \quad (\text{Triangle ineq.}) \\ &\leq \gamma \sup_{s \in S} \sum_{s' \in S} P(s, \pi(s), s') \|U - V\|_\infty \quad (\sup \text{ norm 정의}) \\ &= \gamma \|U - V\|_\infty \quad \left( \sum_{s'} P = 1 \right)\end{aligned}$$

# $V^\pi$ 계산 방법

Banach fixed-point theorem에 의해:

임의의  $v_0 \in B(S)$ 에서 시작하여 수열을 구성:

$$v_0, \quad T^\pi v_0, \quad (T^\pi)^2 v_0, \quad (T^\pi)^3 v_0, \quad \dots$$

이 수열은  $V^\pi$ 로 기하급수적 속도로 수렴한다!

수렴 속도:

$$\|(T^\pi)^n v_0 - V^\pi\|_\infty \leq \gamma^n \|v_0 - V^\pi\|_\infty$$

⇒ Policy evaluation의 이론적 근거

# Bellman Optimality Operator $T^*$

Bellman optimality operator  $T^* : B(S) \rightarrow B(S)$ 를 다음과 같이 정의:

$$(T^* V)(s) = \sup_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V(s') \right\}, \quad s \in S$$

$T^*$ 도 well-defined:  $V \in B(S)$ 이면  $T^* V \in B(S)$

목표:  $T^*$ 도  $\gamma$ -contraction임을 보이자!

# 보조 부등식

먼저 다음 부등식을 주목하자:

$$\left| \sup_{a \in A} f(a) - \sup_{a \in A} g(a) \right| \leq \sup_{a \in A} |f(a) - g(a)|$$

증명 : case analysis로 보일 수 있음

직관:

- 두 supremum의 차이는
- 각 점에서의 차이의 supremum보다 클 수 없다

이 부등식을  $T^*$ 의 contraction 증명에 사용할 것이다.

# $T^*$ 는 Contraction

정리:  $T^*$ 는  $\gamma$ -contraction이다.

증명:

$$\begin{aligned}\|T^*U - T^*V\|_\infty &\leq \gamma \sup_{(s,a) \in S \times A} \sum_{s' \in S} P(s, a, s') |U(s') - V(s')| \\ &\leq \gamma \sup_{(s,a) \in S \times A} \sum_{s' \in S} P(s, a, s') \|U - V\|_\infty \\ &= \gamma \|U - V\|_\infty\end{aligned}$$

마지막 등식:  $\sum_{s' \in S} P(s, a, s') = 1$

$\Rightarrow T^*$ 도 유일한 fixed point를 가지며, 반복 적용으로 기하급수적 수렴!

## Theorem 2: Characterization of Optimal Policy

정리:  $V$ 를  $T^*$ 의 fixed point라 하고, policy  $\pi$ 가  $V$ 에 대해 greedy라고 하자:

$$T^\pi V = T^* V$$

그러면  $V = V^*$ 이고,  $\pi$ 는 optimal policy이다.

의미:

- $T^*$ 의 fixed point를 찾으면 그것이 곧  $V^*$
- 그 fixed point에 대해 greedy한 policy가 optimal policy
- Value iteration의 이론적 근거

## Theorem 2 증명 (1/2)

Step 1: 임의의 stationary policy  $\pi$ 에 대해  $V^\pi \leq V$ 임을 보이기

$$T^\pi \leq T^* \text{임에 주목 (임의의 } W \in B(S) \text{에 대해 } T^\pi W \leq T^* W\text{)}$$

따라서:

$$V^\pi = T^\pi V^\pi \leq T^* V^\pi$$

$T^*$ 가 monotone이므로 ( $U \leq W \Rightarrow T^* U \leq T^* W$ ):

$$T^* V^\pi \leq (T^*)^2 V^\pi$$

부등식을 연쇄시키면:

$$V^\pi \leq (T^*)^n V^\pi \quad \text{for all } n \geq 0$$

$T^*$ 가 contraction이므로 우변은  $V$ 로 수렴. 따라서  $V^\pi \leq V$ .

$\pi$ 가 임의였으므로:  $V^* \leq V$

## Theorem 2 증명 (2/2)

Step 2:  $T^\pi V = T^* V$ 인 policy  $\pi$ 를 선택

$V$ 가  $T^*$ 의 fixed point이므로:

$$T^\pi V = T^* V = V$$

$T^\pi$ 는 유일한 fixed point  $V^\pi$ 를 가지므로:

$$V^\pi = V$$

결론:

- $V^* \leq V = V^\pi \leq V^*$
- 따라서  $V = V^*$
- $\pi$ 는 optimal policy

□

# Existence of Greedy Policy

주의: Theorem 2에서  $V$ 에 대해 greedy한 policy가 존재해야 함

존재 조건:

- Finite action space: 항상 존재
- Infinite action space: 추가적인 연속성 가정 필요

Greedy policy:

$$\pi(s) \in \arg \max_{a \in A} \left\{ r(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V(s') \right\}$$

## Theorem 3 (Policy Improvement Theorem)

정리: Stationary policy  $\pi_0$ 를 선택하고,  $\pi$ 가  $V^{\pi_0}$ 에 대해 greedy라 하자:

$$T^\pi V^{\pi_0} = T^* V^{\pi_0}$$

그러면  $V^\pi \geq V^{\pi_0}$  (즉,  $\pi$ 는  $\pi_0$ 의 improvement)

특히:

- 어떤 state  $s$ 에서  $T^* V^{\pi_0}(s) > V^{\pi_0}(s)$ 이면:  $V^\pi(s) > V^{\pi_0}(s)$  (strict improvement)
- $T^* V^{\pi_0} = V^{\pi_0}$ 이면:  $\pi_0$ 은 이미 optimal policy

⇒ Policy iteration의 이론적 근거

## Theorem 3 증명

증명:

$$T^\pi V^{\pi_0} = T^* V^{\pi_0} \geq T^{\pi_0} V^{\pi_0} = V^{\pi_0}$$

양변에  $T^\pi$ 를 적용:

$$(T^\pi)^2 V^{\pi_0} \geq T^\pi V^{\pi_0} \geq V^{\pi_0}$$

계속 반복하면:

$$(T^\pi)^n V^{\pi_0} \geq V^{\pi_0} \quad \text{for all } n \geq 0$$

양변의 극한을 취하면:  $V^\pi \geq V^{\pi_0}$

□

**Strict improvement:**  $(T^\pi)^n V^{\pi_0}(s) \geq T^* V^{\pi_0}(s) > V^{\pi_0}(s)$

극한을 취하면:  $V^\pi(s) > V^{\pi_0}(s)$

## Theorem 3 증명: 세 번째 부분

$T^* V^{\pi_0} = V^{\pi_0}$  이면  $\pi_0$ 가 optimal:

$T^* V^{\pi_0} = V^{\pi_0}$  이면  $V^{\pi_0}$ 는  $T^*$ 의 fixed point

$T^*$ 은 contraction이므로 유일한 fixed point  $V$ 를 가짐

따라서  $V = V^{\pi_0}$

또한  $V^{\pi_0} \leq V^* \leq V$ 이므로:

$$V^{\pi_0} = V^* = V$$

$\Rightarrow \pi_0$ 는 optimal policy

□

## Corollary 4: Termination of Policy Iteration

Corollary: MDP가 finite이면:

1. Policy iteration은 유한 단계에서 terminate
2. Termination 시 optimal policy를 반환

또한: Stationary policy가 optimal  $\Leftrightarrow$  그 value function이  $T^*$ 의 fixed point

증명:

- Theorem 3에 의해 policy 수열은 strictly improving
- Finite MDP에서는 policy의 개수가 유한
- 따라서 반드시 terminate
- Termination 시  $T^* V^\pi = T^\pi V^\pi = V^\pi \Rightarrow \pi$ 는 optimal

□

## Corollary 5: Optimality of Greedy Policy

**Corollary:**  $V$ 를  $T^*$ 의 유일한 fixed point라 하자.

**Part 1:**  $V$ 에 대해 greedy한 모든 policy는 optimal policy이다.

**Part 2:** Optimal stationary policy  $\pi^*$ 가 존재하면:

- $V = V^*$
- $\pi^*$ 는  $V^*$ 에 대해 greedy

**의미:** Optimal policy는 정확히  $V^*$ 에 대해 greedy한 policy들이다!

## Corollary 5 증명

Part 1: Theorem 2에서 바로 따름

Part 2 증명:

$\pi^*$ 가 optimal stationary policy라 가정. 따라서  $V^{\pi^*} = V^*$

$$V^{\pi^*} = T^{\pi^*} V^{\pi^*} \leq T^* V^{\pi^*}$$

Corollary 4의 두 번째 부분에 의해:  $T^* V^{\pi^*} = V^{\pi^*}$

따라서:

$$V^{\pi^*} \leq V^* \leq V = V^{\pi^*}$$

모두 같으며:  $T^{\pi^*} V^* = T^* V^*$

핵심: Optimal policy  $\Leftrightarrow V^*$ 에 대해 greedy

□

# **Value Prediction Problems**

---

# What is Value Prediction?

How do we estimate the **Value Function**  $V$  in an MRP (Markov Reward Process)?

**Problem Setup:**

- States evolve according to Markov dynamics
- Rewards are generated based on states
- Goal: Estimate the discounted sum of future rewards starting from state  $s$

**Value Function:**

$$V(s) = \mathbb{E}[G \mid S_0 = s]$$

where  $G = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$  (Return)

# Why "Value Prediction"?

Many real-world problems take the form of "expected cumulative future outcomes":

- Probability of future events
  - e.g., machine failure probability, customer churn probability
- Expected time until an event occurs
  - e.g., mean waiting time, mean time to failure
- Value Function under a policy
  - State value  $V^\pi(s)$ , Action value  $Q^\pi(s, a)$

# Monte-Carlo (MC) Methods

핵심 아이디어:  $V(s) = \mathbb{E}[G_t \mid S_t = s]$ 를 경험에서 학습

MC는 간단한 아이디어를 사용:

- 환경 dynamics를 모델링하지 않고 raw experience의 episode로부터 학습
- 관측된 평균 return을 기대 return의 근사로 계산

중요: Empirical return  $G_t$ 를 계산하려면

- Complete episodes  $S_1, A_1, R_2, \dots, S_T$ 가 필요
- $$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$
- 모든 episode는 결국 종료되어야 함

# MC의 Empirical Mean Return

State  $s$ 에 대한 empirical mean return:

$$V(s) = \frac{\sum_{t=1}^T 1[S_t = s] G_t}{\sum_{t=1}^T 1[S_t = s]}$$

여기서  $1[S_t = s]$ 는 binary indicator function

두 가지 방식:

- Every-visit: State  $s$ 를 방문할 때마다 카운트
  - 한 episode에서 같은 state의 여러 방문을 모두 포함
- First-visit: Episode에서 처음 만난 때만 카운트
  - 한 episode에서 state를 처음 만났을 때만 포함

# 숫자 예제

State  $s$ 가 3번 등장했고, 그때 `return`이:

$$G_1 = 10, \quad G_2 = 4, \quad G_3 = 7$$

MC Value 계산:

$$V(s) = \frac{G_1 + G_2 + G_3}{3} = \frac{10 + 4 + 7}{3} = 7$$

해석:

- State  $s$ 에서 시작했을 때 평균적으로 7의 `return`을 받음
- 실제 관측된 `return`들의 단순 평균
- 더 많은 `sample`이 모이면 추정이 더 정확해짐

# Every-visit vs First-visit 예시

Episode 예시:  $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2$  (state  $s_0$ 가 두 번 등장)

Reward:  $R_1 = 2, R_2 = 3, R_3 = 5$  (할인율  $\gamma = 1$ 로 단순화)

First-visit MC:

- $s_0$ 의 첫 방문 (시간 0):  $G_0 = R_1 + R_2 + R_3 = 2 + 3 + 5 = 10$
- 두 번째 방문은 무시
- $V(s_0) = 10$

Every-visit MC:

- 첫 번째 방문 (시간 0):  $G_0 = 2 + 3 + 5 = 10$
- 두 번째 방문 (시간 2):  $G_2 = 5$
- $V(s_0) = \frac{10+5}{2} = 7.5$

---

**Algorithm 2** The function that implements the every-visit Monte-Carlo algorithm to estimate value functions in episodic MDPs. This routine must be called at the end of each episode with the state-reward sequence collected during the episode. Note that the algorithm as shown here has linear time- and space-complexity in the length of the episodes.

---

**function** EVERYVISITMC( $X_0, R_1, X_1, R_2, \dots, X_{T-1}, R_T, V$ )

**Input:**  $X_t$  is the state at time  $t$ ,  $R_{t+1}$  is the reward associated with the  $t^{\text{th}}$  transition,  $T$  is the length of the episode,  $V$  is the array storing the current value function estimate

```
1: sum  $\leftarrow 0$ 
2: for  $t \leftarrow T - 1$  downto 0 do
3:   sum  $\leftarrow R_{t+1} + \gamma \cdot \text{sum}$ 
4:   target[ $X_t$ ]  $\leftarrow$  sum
5:    $V[X_t] \leftarrow V[X_t] + \alpha \cdot (\text{target}[X_t] - V[X_t])$ 
6: end for
7: return  $V$ 
```

---

Every-visit MC는 episode에서 같은 state를 여러 번 방문할 때마다 모두 카운트

# Action-Value Function으로 확장

이 근사 방법은 action-value function으로 쉽게 확장됨:

$$Q(s, a) = \frac{\sum_{t=1}^T 1[S_t = s, A_t = a] G_t}{\sum_{t=1}^T 1[S_t = s, A_t = a]}$$

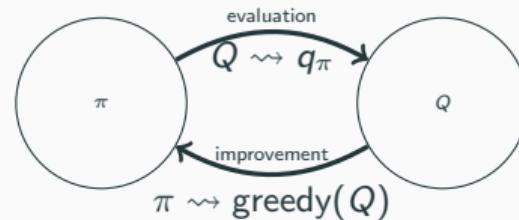
$(s, a)$  pair를 카운트

왜  $Q$ 가 중요한가?

- Model-free 환경에서 policy improvement를 위해 필요
- $V$ 만으로는 다음 action을 선택하기 어려움 (transition model 필요)
- $Q$ 가 있으면  $\pi(s) = \arg \max_a Q(s, a)$ 로 바로 선택 가능

# MC로 Optimal Policy 학습

MC로 optimal policy를 학습하려면, GPI (Generalized Policy Iteration)와 유사한 아이디어를 따름:



반복 과정:

1. **Policy Evaluation:** 현재 policy  $\pi$ 에 대해  $Q^\pi(s, a)$  추정
2. **Policy Improvement:**  $Q$ 에 대해 greedy하게 policy 개선

# MC Control Algorithm

1. Policy Improvement: 현재 value function에 대해 greedy하게 policy 개선

$$\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

2. Generate Episode: 새 policy  $\pi$ 로 새 episode 생성

- $\varepsilon$ -greedy 같은 알고리즘 사용하여 exploration과 exploitation 균형

3. Estimate  $Q$ : 새 episode로  $Q$  추정

$$q_\pi(s, a) = \frac{\sum_{t=1}^T \left( 1[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^T 1[S_t = s, A_t = a]}$$

이 과정을 수렴할 때까지 반복

# $\varepsilon$ -Greedy Exploration

문제: 순수 greedy policy는 exploration이 부족할 수 있음

$\varepsilon$ -Greedy Solution:

- 확률  $1 - \varepsilon$ 로: 현재 최선의 action 선택 (exploitation)

$$a = \arg \max_{a'} Q(s, a')$$

- 확률  $\varepsilon$ 로: 랜덤하게 action 선택 (exploration)

$$a \sim \text{Uniform}(\mathcal{A})$$

Trade-off:

- $\varepsilon$ 가 크면: 더 많은 exploration, 느린 수렴
- $\varepsilon$ 가 작으면: 더 많은 exploitation, local optimum 위험

# MC의 한계 (1): 높은 Variance

문제: Return의 variance가 크면 sample이 많이 필요함

이유:

- Return  $G = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$
- 미래에 일어나는 여러 random 요소가 쌓임
- $\Rightarrow$  Variance가 커지기 쉬움

결과:

- 평균이 안정되려면 sample을 많이 모아야 함
- 추정 품질이 천천히 좋아짐
- Data efficiency가 낮음

## MC의 한계 (2): Reset 불가

문제: State를 특정 위치로 "reset"하는 게 어려운 경우가 많음

MC는 "state  $s$ 에서 시작하는 독립 실험"이 필요한데...

현실에서는:

- 특정 state로 돌아가는 게 불가능
  - 예: 공항 운영, 전력망, 금융 시장
- Reset 비용이 너무 비쌈
- Online/실시간으로 계속 돌아가야 함
  - " $s$ 로 돌아가서 다시 해볼게요"가 안 됨

⇒ 억지로 MC 쓰면 bias가 생길 수 있음

# 대안: TD (Temporal Difference) Learning

핵심 아이디어:

Episode 끝날 때까지 기다리지 말고,  
한 step 보고 바로 update하자!

TD(0) Update Rule:

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))}_{\text{TD target}}$$

TD Error:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Bellman equation을 써서 다음 state의 estimate로 **bootstrapping**

TD는 **Markov property** 활용: 현재 state  $S_t$ 와 다음 state  $S_{t+1}$ 만으로 update

# MC vs TD: Target 비교

MC Target: 실제 Return (episode 끝까지 기다림)

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

TD Target: Bootstrapped estimate (한 step만 기다림)

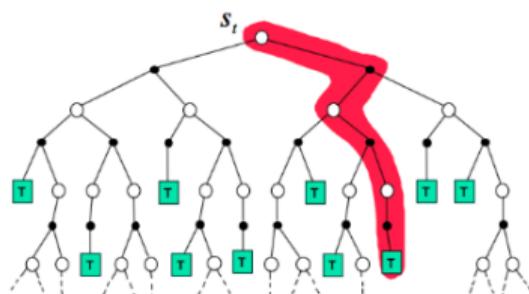
$$R_{t+1} + \gamma V(S_{t+1})$$

	MC	TD
Target	실제 $G_t$	$R_{t+1} + \gamma V(S_{t+1})$
기다리는 시간	Episode 끝	한 step
Bias	없음	있음 (bootstrapping)
Variance	높음	낮음

# MC vs TD: 시각적 비교

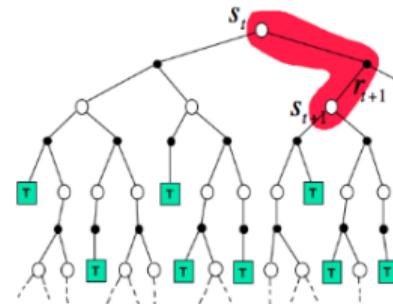
Monte-Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



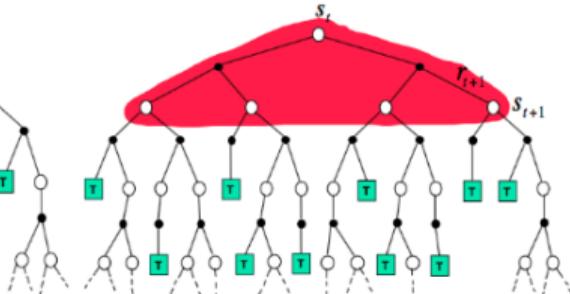
Temporal-Difference

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



Dynamic Programming

$$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$$



# Bias-Variance Trade-off

	Bias	Variance
MC	Low	High
TD	High	Low

MC:

- 실제 return을 쓰므로 unbiased
- 하지만 return에 많은 random 요소가 쌓여서 → high variance

TD:

- 현재 estimate  $V(S_{t+1})$ 을 쓰므로 biased
- 하지만 한 step의 randomness만 관여해서 → low variance

$\gamma$ 가 작거나 episode가 길수록 TD의 이점이 커짐

# When are Monte Carlo methods preferred over TD?

Question from Stack Exchange<sup>2</sup>:

"TD-learning uses bootstrapping to approximate the action-value function and Monte Carlo uses an average to accomplish this. I just can't really think of a scenario when MC is the better way to go."

Main answer:

TD learning과 DP의 주요 문제는 step updates가 초기 조건에 biased되어 있다는 점. Bootstrapping 과정에서  $Q(s', a')$ 를 사용하는데, 학습 초기에는 실제 보상 정보를 포함하지 않음.

Bias는 점진적으로 감소하지만, 특히 off-policy methods(e.g. Q-Learning)와 function approximators를 함께 사용할 때 문제 발생 → deadly triad

**Monte Carlo methods:** 실제 sample을 사용하므로 unbiased. 하지만 **high variance** → 더 많은 samples 필요

**실용적 관점:** TD learning이 더 효율적으로 학습 (deadly triad를 극복할 수 있다면). Experience replay와 frozen estimators가 해결책 (e.g., DQN)

**MC의 실용적 장점:** 개념적으로 단순하고 robust하며 구현이 쉬움. Policy evaluation에 적합 (unbiased measure로 여러 agent 비교 시)

---

<sup>1</sup> <https://stats.stackexchange.com/questions/336974>

# Tabular TD(0) (1/5): 알고리즘

**목표:** 유한한 MRP  $M$ 의 value function  $V$ 를 추정하기

$((X_t, R_{t+1}); t \geq 0)$ : MRP에서 실제로 관측한 데이터

$\hat{V}_t(x)$ : 시간  $t$ 에서 상태  $x$ 의 value 추정값 (처음엔  $\hat{V}_0 \equiv 0$ )

**TD(0) 업데이트:**

$$\delta_{t+1} = R_{t+1} + \gamma \hat{V}_t(X_{t+1}) - \hat{V}_t(X_t)$$

$$\hat{V}_{t+1}(x) = \hat{V}_t(x) + \alpha_t \delta_{t+1} \mathbb{I}\{X_t = x\}, \quad x \in \mathcal{X}$$

핵심:

- 방금 방문한 상태  $X_t$ 의 값만 업데이트함
- $\alpha_t \leq 1$ 면: 현재 값을 목표값  $R_{t+1} + \gamma \hat{V}_t(X_{t+1})$  쪽으로 조금씩 이동
- TD error**  $\delta_{t+1}$ : 연속된 두 시점의 상태 가치 차이

# TD(0)

---

**Algorithm 1** The function implementing the tabular TD(0) algorithm. This function must be called after each transition.

---

**function** TD0( $X, R, Y, V$ )

**Input:**  $X$  is the last state,  $Y$  is the next state,  $R$  is the immediate reward associated with this transition,  $V$  is the array storing the current value estimates

- 1:  $\delta \leftarrow R + \gamma \cdot V[Y] - V[X]$
  - 2:  $V[X] \leftarrow V[X] + \alpha \cdot \delta$
  - 3: **return**  $V$
- 

TD(0)는 한 step만 보고 바로 업데이트 (bootstrapping)

## Tabular TD(0) (2/5): 의사코드

Listing 1: TD(0) Implementation

```
def TD0(X, R, Y, V, alpha, gamma):
    """
    X: last state
    R: immediate reward
    Y: next state
    V: array storing current value estimates
    """
    delta = R + gamma * V[Y] - V[X]
    V[X] = V[X] + alpha * delta
    return V
```

특징:

- 매 transition마다 호출됨
- Bootstrapping 사용: 다음 상태의 추정값  $\hat{V}(X_{t+1})$ 을 활용
- Stochastic Approximation (확률적 근사) 알고리즘

## Tabular TD(0) (3/5): 수렴성

수렴 원리: TD(0)가 수렴하면  $\hat{V}$ 는 다음을 만족:

$$F_{\hat{V}}(x) = \mathbb{E}[R_{t+1} + \gamma \hat{V}(X_{t+1}) - \hat{V}(X_t) \mid X_t = x] = 0$$

간단히 계산하면:  $F_{\hat{V}} = T\hat{V} - \hat{V}$  ( $T$ 는 Bellman operator)

$F_{\hat{V}} = 0$ 의 유일한 해는 참 value function  $V \Rightarrow$  TD(0)는  $V$ 로 수렴

수렴 조건:  $(X_t; t \in \mathbb{N})$ 이 stationary, ergodic Markov chain이고, step-size가 Robbins-Monro 조건을 만족하면:

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < +\infty$$

$\hat{V}_t$ 는 거의 확실히  $V$ 로 수렴

# Tabular TD(0) (4/5): Step-Size 설정하기

이론적으로 좋은 step-size (Robbins-Monro 조건):

- $\alpha_t = c/t$  (가장 단순한 형태)
- 일반적으로:  $\alpha_t = ct^{-\eta}$ , 단  $1/2 < \eta \leq 1$ 
  - $\eta = 1$ : 장기적으로 최선 (step-size가 제일 천천히 감소)
  - $\eta \rightarrow 1/2$ : 초기 학습이 빠름 (step-size가 천천히 감소)

실전에선 고정 step-size를 많이 씀:

- 이론적 조건은 안 맞지만, 실용적인 이유가 있음:
  1. 환경이 변할 때: Policy가 계속 바뀌는 경우
  2. 데이터가 적을 때: 샘플이 충분하지 않은 경우
- 고정 step-size 쓰면: 분포 수렴 (분산이 step-size에 비례)

수렴 속도: Linear SA이므로  $O(1/\sqrt{t})$

## Tabular TD(0) (5/5): Off-Policy 학습

더 일반적인 형태:  $((X_t, R_{t+1}, Y_{t+1}); t \geq 0)$  데이터에도 적용 가능

$(X_t; t \geq 0)$ 은 임의의 ergodic Markov chain,  $(Y_{t+1}, R_{t+1}) \sim P(\cdot | X_t)$

TD error만 바꾸면 됨:

$$\delta_{t+1} = R_{t+1} + \gamma \hat{V}(Y_{t+1}) - \hat{V}(X_t)$$

이렇게 쓸 수 있음:

1. 시뮬레이터로 상태 제어: MRP와 무관하게  $\{X_t\}$  분포를 조절 가능
2. Off-policy 학습: 한 policy를 따라가면서 다른 policy를 평가
  - 여러 policy를 동시에 학습할 수 있음
3. 에피소드 문제: 종료 상태에 도달하면 초기 분포  $P_0$ 에서 다시 시작
  - "재시작이 있는 연속 샘플링"

# TD(0) vs MC: 수렴 속도 비교 (1/3)

상황에 따라 다름. 구체적 예시로 살펴보자.

예제 MRP (Figure 4):

- 초기 state: state 1 (확률 0.9) 또는 state 2 (확률 0.1)
- State 1 → 0 → 3 → 4 (종료)
- State 2 → 3 → 4 (종료)
- State 3 → 4로 갈 때만 보상:  $R \sim \text{Bernoulli}(0.5)$
- 나머지 transition은 보상 0

핵심 관찰:

- State 1은 자주 방문 (90%)
- State 2는 가끔 방문 (10%)
- State 3은 state 2보다 약 10배 자주 방문

# TD(0) vs MC: 수렴 속도 비교 (2/3)

## Case 1: TD(0)가 더 빠른 경우

State 3에서 보상이 Bernoulli(0.5)로 random일 때:

### TD(0)의 장점:

- State 2를  $k$ 번째 방문 시, state 3은 약  $10k$ 번 방문
- State 3의 estimate 정확:  $\text{Var}[\hat{V}_t(3)] \approx 1/(10k)$
- State 2는 정확한 estimate를 target으로 사용
- 시간에 따라 target 정확도 향상

### MC의 한계:

- State 3 estimate 무시, Bernoulli 보상 직접 사용
- $\text{Var}[R_t | X_t = 2] = 0.25$  계속 유지
- Target variance 불변  $\Rightarrow$  수렴 느림

$\Rightarrow$  Bootstrapping이 도움됨!

## TD(0) vs MC: 수렴 속도 비교 (3/3)

### Case 2: MC가 더 빠른 경우

State 3 → 4의 보상이 deterministic하게 1일 때:

#### MC의 장점:

- $R_t = 1$ 이 정확한 target
- 바로 정확한 값으로 업데이트, 빠르게 수렴

#### TD(0)의 한계:

- State 2 값이 정확해지려면 State 3 estimate가 먼저 정확해져야 함
- 연쇄 과정으로 수렴 느림

극단적 예: Chain  $1 \rightarrow 2 \rightarrow \dots \rightarrow N$

- MC:  $N$ 과 무관하게 일정
- TD(0):  $N$  증가 시 느려짐

⇒ 상황에 따라 선택.

# TD( $\lambda$ ): MC와 TD(0)의 통합 (1/5)

핵심 아이디어: MC와 TD(0)를 통합하는 방법이 존재

TD( $\lambda$ ) family (Sutton, 1984, 1988):

- $\lambda \in [0, 1]$ : MC와 TD(0) 사이를 보간하는 파라미터
- $\lambda = 0$ : TD(0)
- $\lambda = 1$ : MC (every-visit)
- $0 < \lambda < 1$ : 중간 형태 (multi-step method)

Multi-step return:

$$R_{t:k} = \sum_{s=t}^{t+k} \gamma^{s-t} R_{s+1} + \gamma^{k+1} \hat{V}_t(X_{t+k+1})$$

$k$ -step 동안의 실제 보상 + bootstrapping

TD( $\lambda$ )는 이러한 multi-step returns를 exponential weights  $(1 - \lambda)\lambda^k$ 로 혼합

---

**Algorithm 3** The function that implements the tabular TD( $\lambda$ ) algorithm with replacing traces. This function must be called after each transition.

---

**function** TDLAMBDA( $X, R, Y, V, z$ )

**Input:**  $X$  is the last state,  $Y$  is the next state,  $R$  is the immediate reward associated with this transition,  $V$  is the array storing the current value function estimate,  $z$  is the array storing the eligibility traces

- 1:  $\delta \leftarrow R + \gamma \cdot V[Y] - V[X]$
- 2: **for all**  $x \in \mathcal{X}$  **do**
- 3:      $z[x] \leftarrow \gamma \cdot \lambda \cdot z[x]$
- 4:     **if**  $X = x$  **then**
- 5:          $z[x] \leftarrow 1$
- 6:     **end if**
- 7:      $V[x] \leftarrow V[x] + \alpha \cdot \delta \cdot z[x]$
- 8: **end for**
- 9: **return**  $(V, z)$

---

TD( $\lambda$ )는  $\lambda$  값에 따라 MC( $\lambda = 1$ )와 TD(0)( $\lambda = 0$ ) 사이를 보간

# TD( $\lambda$ ): MC와 TD(0)의 통합 (2/5)

Eligibility Traces: Incremental 구현의 핵심

Accumulating traces:

$$\begin{aligned}\delta_{t+1} &= R_{t+1} + \gamma \hat{V}_t(X_{t+1}) - \hat{V}_t(X_t) \\ z_{t+1}(x) &= \mathbb{I}\{x = X_t\} + \gamma \lambda z_t(x) \\ \hat{V}_{t+1}(x) &= \hat{V}_t(x) + \alpha_t \delta_{t+1} z_{t+1}(x)\end{aligned}$$

초기값:  $z_0(x) = 0$

Trace  $z_t(x)$ :

- TD error가 value 업데이트에 미치는 영향 조절
- 최근 방문 state일수록 큰 값
- $\gamma \lambda$ 로 지수적 감쇠

# TD( $\lambda$ ): MC와 TD(0)의 통합 (3/5)

Replacing traces:

$$z_{t+1}(x) = \max(\mathbb{I}\{x = X_t\}, \gamma \lambda z_t(x))$$

Accumulating vs Replacing:

- **Accumulating:** 방문마다 누적
- **Replacing:** 방문 시 1로 리셋
- 실전에선 replacing이 더 나은 경우 많음

Parameter  $\lambda$ :

- Bootstrapping 정도 조절
- $\lambda = 0$ : TD(0)
- $\lambda = 1$ : MC

# TD( $\lambda$ ): MC와 TD(0)의 통합 (4/5)

Pseudocode (Replacing traces):

Listing 2: TD(lambda) with Replacing Traces

```
def TDLambda(X, R, Y, V, z, alpha, gamma, lam):
    """
    z: array storing eligibility traces
    lam: lambda parameter
    """
    delta = R + gamma * V[Y] - V[X]

    for x in States:
        z[x] = gamma * lam * z[x]
        if X == x:
            z[x] = 1
        V[x] = V[x] + alpha * delta * z[x]

    return V, z
```

## TD( $\lambda$ ): MC와 TD(0)의 통합 (5/5)

극한 경우의 동작:

- $\lambda = 0$ : TD(0)와 동일
- $\lambda = 1$  + accumulating traces: Every-visit MC와 동일 (episodic)
- $\lambda = 1$  + replacing traces: First-visit MC와 동일 (undiscounted)

실전 팁:

- $\lambda$ 는 trial and error로 결정
- 학습 중에  $\lambda$ 를 변경해도 수렴성 유지

장점:

- MRP의 value function 추정
- Non-episodic 문제에 사용 가능
- 적절한  $\lambda$ 로 MC나 TD(0)보다 훨씬 빠른 수렴

# Large State Space 문제 (1/3)

문제: State space가 크거나 무한할 때 tabular 방식 불가능

해결책: Function Approximation

$$V_{\theta}(x) = \theta^T \phi(x), \quad x \in \mathcal{X}$$

- $\theta \in \mathbb{R}^d$ : Parameter vector
- $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$ : Feature extraction
- $\phi_i(x)$ : State  $x$ 의 features

핵심:

- 모든 state 값 개별 저장  $\rightarrow$  불가능
- Parameter  $\theta$ 로 근사  $\rightarrow$  가능!
- Feature  $\phi(x)$ 가 중요 정보 압축

# Large State Space 문제 (2/3)

Feature Extraction 방법들:

## 1. Polynomial/Fourier/Wavelet Basis

- 예:  $\phi(x) = (1, x, x^2, \dots, x^{d-1})^\top$

## 2. Radial Basis Functions (RBF)

- $\phi_i(x) = \exp(-\eta \|x - x^{(i)}\|^2)$ , Grid에 Gaussian 배치

## 3. State Aggregation

- $\phi(x) \in \{0, 1\}^d$ , Binary features
- 계산 효율적:  $V_\theta(x) = \sum_{i:\phi_i(x)=1} \theta_i$

## 4. Tile Coding (CMAC)

- Multiple shifted tilings,  $s$ -sparse features

## Large State Space 문제 (3/3)

### Curse of Dimensionality:

- $D$ -차원을  $\epsilon$  간격으로 discretize:  $d = \epsilon^{-D}$
- 예:  $\epsilon = 1/2$ ,  $D = 100 \Rightarrow d \approx 10^{30}$

### 왜 때로는 가능한가?

- Intrinsic complexity는 낮을 수 있음
- 많은 variable이 irrelevant
- State가 low-dim submanifold에 존재

# TD( $\lambda$ ) with Function Approximation (1/5)

문제: Large state space에서 TD( $\lambda$ ) 사용하기

Parametric function approximation ( $V_\theta; \theta \in \mathbb{R}^d$ ) 사용:

- $V_\theta : \mathcal{X} \rightarrow \mathbb{R}$ , smooth ( $\nabla_\theta V_\theta(x)$  exists)

Tabular TD( $\lambda$ )의 일반화 (Sutton, 1984, 1988):

$$\delta_{t+1} = R_{t+1} + \gamma V_{\theta_t}(x_{t+1}) - V_{\theta_t}(x_t)$$

$$z_{t+1} = \nabla_\theta V_{\theta_t}(x_t) + \gamma \lambda z_t$$

$$\theta_{t+1} = \theta_t + \alpha_t \delta_{t+1} z_{t+1}$$

초기값:  $z_0 = 0 \in \mathbb{R}^d$

핵심 차이:

- Tabular:  $z_t(x)$  (각 state마다)
- Function approx:  $z_t \in \mathbb{R}^d$  (parameter space)

## TD( $\lambda$ ) with Function Approximation

---

**Algorithm 4** The function implementing the TD( $\lambda$ ) algorithm with linear function approximation. This function must be called after each transition.

---

**function** TDLAMBDAINFAPP( $X, R, Y, \theta, z$ )

**Input:**  $X$  is the last state,  $Y$  is the next state,  $R$  is the immediate reward associated with this transition,  $\theta \in \mathbb{R}^d$  is the parameter vector of the linear function approximation,  $z \in \mathbb{R}^d$  is the vector of eligibility traces

1:  $\delta \leftarrow R + \gamma \cdot \theta^\top \varphi[Y] - \theta^\top \varphi[X]$

2:  $z \leftarrow \varphi[X] + \gamma \cdot \lambda \cdot z$

3:  $\theta \leftarrow \theta + \alpha \cdot \delta \cdot z$

4: **return** ( $\theta, z$ )

---

TD( $\lambda$ )를 function approximation과 결합하여 large state space에서 사용

## TD( $\lambda$ ) with Function Approximation (2/5)

**Linear Function Approximation:**  $V_\theta(x) = \theta^\top \phi(x)$

$\nabla_\theta V_\theta = \phi$ 이므로 업데이트가 단순해짐:

Listing 3: TD(lambda) with Linear Function Approx

```
def TDLambdaLinFA(X, R, Y, theta, z, phi, alpha, gamma, lam):
    """
    phi: feature function
    """
    delta = R + gamma * theta.dot(phi(Y)) - theta.dot(phi(X))
    z = phi(X) + gamma * lam * z
    theta = theta + alpha * delta * z
    return theta, z
```

Tabular case 복원:

- $\mathcal{X} = \{x_1, \dots, x_D\}$ ,  $\phi_i(x) = \mathbb{I}\{x = x_i\}$
- $z_{t,i} \leftrightarrow z_t(x_i)$ ,  $\theta_{t,i} \leftrightarrow \hat{V}_t(x_i)$

## TD( $\lambda$ ) with Function Approximation (3/5)

수렴 조건 (Linear case):

TD( $\lambda$ )가 거의 확실하게 수렴하려면:

1. Linear function approximation:  $V_\theta = \theta^\top \phi$
2.  $(X_t; t \geq 0) \mid$  ergodic Markov process
3. Stationary distribution  $\mu \models$  MRP와 동일
4. Step-size가 RM 조건 만족

수렴 시 만족하는 식: Projected fixed-point equation

$$V_{\theta(\lambda)} = \Pi_{F,\mu} T^{(\lambda)} V_{\theta(\lambda)}$$

- $F = \{V_\theta | \theta \in \mathbb{R}^d\}$ : Representable functions
- $T^{(\lambda)}$ : Exponentially weighted Bellman operator
- $\Pi_{F,\mu}$ : Projection onto  $F$  w.r.t.  $\|\cdot\|_\mu$

주의: Off-policy  $\nmid$  nonlinear function approx 시 발산 가능!

## TD( $\lambda$ ) with Function Approximation (4/5)

**Multi-step Bellman Operator:**

$$T^{[m]} \hat{V}(x) = \mathbb{E} \left[ \sum_{t=0}^m \gamma^t R_{t+1} + \gamma^{m+1} \hat{V}(X_{m+1}) \mid X_0 = x \right]$$

**TD( $\lambda$ ) Operator:**

$$T^{(\lambda)} \hat{V}(x) = (1 - \lambda) \sum_{m=0}^{\infty} \lambda^m T^{[m]} \hat{V}(x)$$

특수한 경우:  $\lambda = 0 \Rightarrow T^{(0)} = T$ ,  $\lambda = 1 \Rightarrow T^{(1)} = V$

**Error Bound:**

$$\|V_{\theta(\lambda)} - V\|_{\mu} \leq \frac{1}{\sqrt{1 - \gamma_{\lambda}}} \|\Pi_{F,\mu} V - V\|_{\mu}$$

여기서  $\gamma_{\lambda} = \gamma(1 - \lambda)/(1 - \lambda\gamma)$  (contraction modulus)

- $\lambda = 1$ : Best approximation in  $F$
- $\lambda \rightarrow 0$ : Bound가 커짐 (더 큰 error 허용)

## TD( $\lambda$ ) with Function Approximation (5/5)

TD( $\lambda$ )가 해결하는 Model:

$$\text{Bellman error: } \Delta^{(\lambda)}(\hat{V}) = T^{(\lambda)}\hat{V} - \hat{V}$$

Error decomposition:

$$\Delta^{(\lambda)}(V_{\theta^{(\lambda)}}) = (1 - \lambda) \sum_{m \geq 0} \lambda^m \Delta_m^{[r]} + \gamma \left( (1 - \lambda) \sum_{m \geq 0} \lambda^m \Delta_m^{[\phi]} \right) \theta^{(\lambda)}$$

- $\Delta_m^{[r]}$ :  $m$ -step reward modeling error
- $\Delta_m^{[\phi]}$ :  $m$ -step transition modeling error

해석:

- $\lambda \rightarrow 1$ : Features가 value function 구조를 잘 잡아야 함
- $\lambda \rightarrow 0$ : Features가 immediate rewards/transitions를 잘 잡아야 함
- Best  $\lambda$ 는 features가 short-term vs long-term 중 어느 것을 더 잘 capture하는지에 따라 달라짐

실전: TD( $\lambda$ ) with  $\lambda < 10$  | TD(1)보다 훨씬 빠르게 수렴

# Gradient Temporal Difference Learning (1/4)

문제: TD( $\lambda$ )의 off-policy divergence

TD( $\lambda$ )는 off-policy 상황에서 발산 가능  $\Rightarrow$  안정성 문제

해결책: GTD2 & TDC (Sutton et al., 2009)

- Off-policy에서도 수렴 보장
- On-policy에서 TD( $\lambda$ ) 솔루션으로 수렴
- TD( $\lambda$ )와 거의 같은 계산 효율성

설정:  $\lambda = 0$  (TD(0)), linear function approximation

- $(X_t, R_{t+1}, Y_{t+1})$ : Stationary process
- $X_t \sim \nu$  ( $\nu$ 는 stationary distribution과 다를 수 있음)
- Features  $\phi$ 는 linearly independent

핵심 아이디어: Projected fixed-point equation의 solution  $\theta^{(0)}$ 를 찾되, gradient 기반 방법 사용

## Gradient Temporal Difference Learning (2/4)

**Objective Function:**

$$J(\theta) = \|V_\theta - \Pi_{F,\nu} TV_\theta\|_\nu^2$$

- Projected fixed-point equation의 모든 solution은  $J$ 의 minimizer
- Linear independent features  $\Rightarrow$  unique minimizer  $\theta^*$

**Notation:**

$$\begin{aligned}\delta_{t+1}(\theta) &= R_{t+1} + \gamma V_\theta(Y_{t+1}) - V_\theta(X_t) \\ &= R_{t+1} + \gamma \theta^\top \phi'_{t+1} - \theta^\top \phi_t\end{aligned}$$

여기서  $\phi_t = \phi(X_t)$ ,  $\phi'_{t+1} = \phi(Y_{t+1})$

**Gradient:**

$$\nabla_\theta J(\theta) = -2\mathbb{E}[(\phi_t - \gamma\phi'_{t+1})\phi_t^\top]w(\theta)$$

여기서  $w(\theta) = \mathbb{E}[\phi_t\phi_t^\top]^{-1}\mathbb{E}[\delta_{t+1}(\theta)\phi_t]$

## Gradient Temporal Difference Learning (3/4)

**GTD2 Algorithm:** Two sets of weights -  $\theta_t$  (primary),  $w_t$  (auxiliary)

Update rules:

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t(\phi_t - \gamma\phi'_{t+1})\phi_t^\top w_t \\ w_{t+1} &= w_t + \beta_t(\delta_{t+1}(\theta_t) - \phi_t^\top w_t)\phi_t\end{aligned}$$

Listing 4: GTD2 Implementation

```
def GTD2(X, R, Y, theta, w, phi, alpha, beta, gamma):
    f = phi(X)
    f_prime = phi(Y)
    delta = R + gamma * theta.dot(f_prime) - theta.dot(f)
    a = f.dot(w)
    theta = theta + alpha * (f - gamma * f_prime) * a
    w = w + beta * (delta - a) * f
    return theta, w
```

특징:  $w_t$  업데이트는 LMS (Least-Mean Square) rule 사용

---

**Algorithm 5** The function implementing the GTD2 algorithm. This function must be called after each transition.

---

**function** GTD2( $X, R, Y, \theta, w$ )

**Input:**  $X$  is the last state,  $Y$  is the next state,  $R$  is the immediate reward associated with this transition,  $\theta \in \mathbb{R}^d$  is the parameter vector of the linear function approximation,  $w \in \mathbb{R}^d$  is the auxiliary weight

- 1:  $f \leftarrow \varphi[X]$
  - 2:  $f' \leftarrow \varphi[Y]$
  - 3:  $\delta \leftarrow R + \gamma \cdot \theta^\top f' - \theta^\top f$
  - 4:  $a \leftarrow f^\top w$
  - 5:  $\theta \leftarrow \theta + \alpha \cdot (f - \gamma \cdot f') \cdot a$
  - 6:  $w \leftarrow w + \beta \cdot (\delta - a) \cdot f$
  - 7: **return**  $(\theta, w)$
-

# Gradient Temporal Difference Learning (5/5)

TDC (TD with Corrections):

Alternative gradient formulation:

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha_t(\delta_{t+1}(\theta_t)\phi_t - \gamma\phi'_{t+1}\phi_t^\top w_t) \\ w_{t+1} &= w_t + \beta_t(\delta_{t+1}(\theta_t) - \phi_t^\top w_t)\phi_t\end{aligned}$$

주요 차이:

- GTD2:  $\theta$  update에서  $(\phi_t - \gamma\phi'_{t+1})\phi_t^\top w_t$
- TDC:  $\theta$  update에서  $\delta_{t+1}\phi_t - \gamma\phi'_{t+1}\phi_t^\top w_t$
- TDC는 two-timescale SA:  $\alpha_t = o(\beta_t)$  필요

수렴 보장:

- RM 조건 하에서  $\theta_t \rightarrow \theta^*$  almost surely
- Distribution  $\nu$ 와 무관하게 수렴 (off-policy 안전!)
- Computational cost: TD(0)의 약 2배

확장: Eligibility traces, nonlinear function approximation으로 확장 가능

# Least-Squares Methods (1/9)

기존 방법들의 한계:

TD( $\lambda$ ), GTD2, TDC는 모두 LMS 스타일의 작은 스텝 업데이트

- Step-size 선택에 민감
- 초기값과 limit point  $\theta(\lambda)$ 의 거리에 민감
- Matrix  $A$ 의 eigenvalue 구조에 영향 받음  
(예: TD(0)의 경우  $A = \mathbb{E}[\phi_t(\phi_t - \gamma\phi'_{t+1})^\top]$ )

개선 시도들:

- Adaptive step-sizes (Sutton, 1992)
- Normalizing updates (Bradtko, 1994)
- Reusing previous samples (Lin, 1992)
- 각각 장단점 존재

해결책: Adaptive filtering의 LS (Least-Squares) 알고리즘을 RL에 적용  
⇒ LMS의 모든 결점 해결

## LSTD: Least-Squares TD Learning (2/9)

핵심 아이디어:

TD(0)는 무한 샘플의 극한에서 다음을 만족하는  $\theta$  찾음:

$$\mathbb{E}[\phi_t \delta_{t+1}(\theta)] = 0$$

유한 샘플  $D_n = ((X_0, R_1, Y_1), \dots, (X_{n-1}, R_n, Y_n))$ 로 근사:

$$\frac{1}{n} \sum_{t=0}^{n-1} \phi_t \delta_{t+1}(\theta) = 0$$

$\delta_{t+1}(\theta) = R_{t+1} - (\phi_t - \gamma \phi'_{t+1})^\top \theta$ 를 대입하면  $\theta$ 에 대한 선형 방정식!

$\hat{A}_n = \frac{1}{n} \sum_{t=0}^{n-1} \phi_t (\phi_t - \gamma \phi'_{t+1})^\top$ 가 non-singular이면:

$$\theta_n = \hat{A}_n^{-1} \hat{b}_n$$

여기서  $\hat{b}_n = \frac{1}{n} \sum_{t=0}^{n-1} R_{t+1} \phi_t$

# LSTD: Least-Squares TD Learning (3/9)

LSTD의 장점:

- TD(0)보다 빠른 수렴 (eigenvalue spread에 영향 받지 않음)
- Sample average approximation 사용
- Projected squared Bellman error의 empirical approximation 최소화:  
 $\|\Pi_{F,\mu}(TV - V)\|_\mu^2$

LSTD의 단점:

- Matrix inversion 필요 ( $d$ 가 크면 비용 증가)
- 자주 호출되면 계산 부담

제안: Bradtke & Barto (1996)

통계적 관점:

- Stochastic programming: Sample average approximation
- Statistics: Z-estimation family

## RLSTD: Recursive LSTD (4/9)

Sherman-Morrison formula 활용: Incremental version of LSTD

초기화:  $\theta_0 \in \mathbb{R}^d$ ,  $C_0 = \beta I$  ( $\beta > 0$  small)

업데이트 ( $t \geq 0$ ):

$$C_{t+1} = C_t - \frac{C_t \phi_t (\phi_t - \gamma \phi'_{t+1})^\top C_t}{1 + (\phi_t - \gamma \phi'_{t+1})^\top C_t \phi_t}$$
$$\theta_{t+1} = \theta_t + \frac{C_t}{1 + (\phi_t - \gamma \phi'_{t+1})^\top C_t \phi_t} \delta_{t+1}(\theta_t) \phi_t$$

계산 복잡도:  $O(d^2)$  per update

특징:

- Adaptive filtering의 RLS (Recursive Least-Squares)와 유사
- Online learning에 적합
- Matrix inversion을 incremental하게 수행

---

**Algorithm 6** The function implementing the RLSTD algorithm. This function must be called after each transition. Initially,  $C$  should be set to a diagonal matrix with small positive diagonal elements:  $C = \beta I$ , with  $\beta > 0$ .

---

**function** RLSTD( $X, R, Y, C, \theta$ )

**Input:**  $X$  is the last state,  $Y$  is the next state,  $R$  is the immediate reward associated with this transition,  $C \in \mathbb{R}^{d \times d}$ , and  $\theta \in \mathbb{R}^d$  is the parameter vector of the linear function approximation

- 1:  $f \leftarrow \varphi[X]$
  - 2:  $f' \leftarrow \varphi[Y]$
  - 3:  $g \leftarrow (f - \gamma f')^\top C$  ▷  $g$  is a  $1 \times d$  row vector
  - 4:  $a \leftarrow 1 + gf$
  - 5:  $v \leftarrow Cf$
  - 6:  $\delta \leftarrow R + \gamma \cdot \theta^\top f' - \theta^\top f$
  - 7:  $\theta \leftarrow \theta + \delta / a \cdot v$
  - 8:  $C \leftarrow C - v g / a$
  - 9: **return**  $(C, \theta)$
-

## LSTD( $\lambda$ ): Extension to Eligibility Traces (5/9)

LSTD( $\lambda$ ) (Boyan, 2002):  $\lambda$  parameter 추가

전제조건:  $\lambda > 0$ 일 때  $X_{t+1} = Y_{t+1}$  필요 (TD errors가 telescope하려면)

Solution은 다음을 만족:

$$\frac{1}{n} \sum_{t=0}^{n-1} \delta_{t+1}(\theta) z_{t+1} = 0$$

여기서  $z_{t+1} = \sum_{s=0}^t (\gamma \lambda)^{t-s} \phi_s$  (eligibility traces)

Recursive form: RLSTD( $\lambda$ ) (Xu et al., 2002; Nedić & Bertsekas, 2003)

문제점: Equation (30)이 해를 갖지 않을 수 있음

- On-policy + 큰 샘플: 항상 해 존재
- 해가 없을 때의 trick: Small positive diagonal matrix 추가  
(보장 안됨)
- Better approach: Projected Bellman error 최소화  
(항상 well-defined)

# LSTD( $\lambda$ ): Convergence Properties (6/9)

수렴성:

Standard assumptions 하에서 LSTD( $\lambda$ ) (및 recursive variants)는 projected fixed-point equation (22)의 solution으로 almost surely 수렴

이론적 결과:

- $\lambda = 0$ : Bradtko & Barto (1996)
- $\lambda > 0$ : Xu et al. (2002), Nedić & Bertsekas (2003)
- On-policy case에서 증명되었으나, off-policy에서도 성립  
(limiting solution이 존재한다면)

(R)LSTD( $\lambda$ )의 장점:

- Step-size 튜닝 불필요
- Matrix  $A$ 의 eigenvalue 구조에 민감하지 않음
- 초기값  $\theta$ 의 선택에 민감하지 않음

실험적 확인: TD( $\lambda$ )보다 빠른 수렴 (Bradtko & Barto, 1996; Boyan, 2002; Xu et al., 2002)

## LSPE: Least-Squares Policy Evaluation (7/9)

$\lambda$ -**LSPE** (Bertsekas & Ioffe, 1996): Multi-step value iteration 모방

$(n - s)$ -step prediction:

$$\hat{V}_{s,n}^{(\lambda)}(\theta) = \theta^\top \phi_s + \sum_{q=s}^{n-1} (\gamma \lambda)^{q-s} \delta_{q+1}(\theta)$$

Loss function:

$$J_n(\hat{\theta}, \theta) = \frac{1}{n} \sum_{s=0}^{n-1} \left( \hat{\theta}^\top \phi_s - \hat{V}_{s,n}^{(\lambda)}(\theta) \right)^2$$

Update:

$$\theta_{t+1} = \theta_t + \alpha_t \left( \arg \min_{\hat{\theta}} J_{n_t}(\hat{\theta}, \theta_t) - \theta_t \right)$$

여기서  $(n_t; t \geq 0)$ 은 non-decreasing integer sequence

**특징:**  $J_n$ 은  $\hat{\theta}$ 에 대해 quadratic  $\Rightarrow$  closed form solution

---

**Algorithm 7** The function implementing the batch-mode  $\lambda$ -LSPE update. This function must be called repeatedly until convergence.

---

**function** LAMBDA LSPE( $D, \theta$ )

**Input:**  $D = ((X_t, A_t, R_{t+1}, Y_{t+1}); t = 0, \dots, n - 1)$  is a list of transitions,  $\theta \in \mathbb{R}^d$  is the parameter vector

```
1:  $A, b, \delta \leftarrow 0$   $\triangleright A \in \mathbb{R}^{d \times d}, b \in \mathbb{R}^d, \delta \in \mathbb{R}$ 
2: for  $t = n - 1$  downto 0 do
3:    $f \leftarrow \varphi[X_t]$ 
4:    $v \leftarrow \theta^\top f$ 
5:    $\delta \leftarrow \gamma \cdot \lambda \cdot \delta + (R_{t+1} + \gamma \cdot \theta^\top \varphi[Y_{t+1}] - v)$ 
6:    $b \leftarrow b + (v + \delta) \cdot f$ 
7:    $A \leftarrow A + f \cdot f^\top$ 
8: end for
9:  $\theta' \leftarrow A^{-1}b$ 
10:  $\theta \leftarrow \theta + \alpha \cdot (\theta' - \theta)$ 
11: return  $\theta$ 
```

---

## LSPE: Properties and Comparison (9/9)

특별 케이스 ( $\lambda = 0$ ,  $\alpha_t = 1$ ):

$$\theta_{t+1} = \arg \min_{\hat{\theta}} \frac{1}{n_t} \sum_{s=0}^{n_t-1} \{ \hat{\theta}^\top \phi(X_s) - (R_{s+1} + \gamma V_{\theta_t}(Y_{s+1})) \}^2$$

⇒ Fitted value iteration with linear function approximation

$\alpha_t < 1$ 의 역할:

- 작은 샘플에서 parameter 안정화
- Control algorithm의 subroutine으로 사용 시 policy를 점진적으로 변경

수렴성: Standard assumptions +  $n_t = t$  하에서 almost surely 수렴

- Decreasing step-sizes (Nedić & Bertsekas, 2003)
- Constant step-sizes:  $0 < \alpha < \frac{2-2\gamma\lambda}{1+\gamma-2\gamma\lambda}$  (항상 1 포함)

## Least-Squares vs TD-like Methods (1/3)

계산 복잡도 비교 (샘플 크기  $n$ ):

Method	Complexity	비고
LSTD	$O(nd^2 + d^3)$	Matrix inversion
RLSTD	$O(nd^2)$	Recursive update
LSPE	$O(nd^2)$	
TD( $\lambda$ )	$O(nd)$	Lightweight
GTD2/TDC	$O(nd)$	Lightweight

Trade-off:

- Least-squares: 높은 안정성 & 정확도  $\Leftrightarrow$  높은 계산 복잡도
- Lightweight: 낮은 복잡도  $\Leftrightarrow$  step-size 민감

Observation: Lightweight는 least-squares가 1번 pass하는 동안  $d$ 번 pass 가능!

Experience Replay (Lin, 1992): 관측을 저장/재사용하여 정확도 향상

## Least-Squares vs TD-like Methods (2/3)

$d$ 가 매우 큰 경우:

- Lightweight methods가 같은 계산 시간 내에 least-squares만큼 성능 발휘 가능
- 예: Go 게임의 value function (Silver et al., 2007) - 백만 개 이상의 features 사용
- 이 경우 least-squares methods는 실행 불가능

분석 시나리오: 새로운 관측이 negligible cost로 가능한 경우

- 데이터 저장/재사용 불필요
- 성능은 계산 속도에 의존

계산 시간  $T$  고정 시:

- Least-squares: 샘플 크기  $n \approx T/d^2$  처리 가능
- Lightweight methods: 샘플 크기  $n' \approx nd = T/d$  처리 가능

⇒ Lightweight methods가  $d$ 배 많은 샘플 처리!

# Least-Squares vs TD-like Methods (3/3)

정확도 비교: Limit parameter  $\theta^*$

수렴 속도:

$$\|\theta_t - \theta^*\| \approx C_1 t^{-1/2} \quad (\text{LSTD})$$

$$\|\theta'_t - \theta^*\| \approx C_2 t^{-1/2} \quad (\text{TD-like})$$

같은 계산 시간  $T$  후의 정확도 비율:

$$\frac{\|\theta'_{n'} - \theta^*\|}{\|\theta_n - \theta^*\|} \approx \frac{C_2}{C_1} d^{-1/2}$$

결론:

- $C_2/C_1 < d^{1/2}$ : Lightweight TD-like 방법이 더 정확
- $C_2/C_1 > d^{1/2}$ : Least-squares 방법이 더 정확
- 경험적 rule:  $d$  작으면 least-squares,  $d$  크면 lightweight

iLSTD (Geramifard et al., 2007): Incremental LSTD

- Sparse features ( $s$  nonzero): Complexity  $O(sd)$  per iteration
- Storage:  $O(\min(ns^2 + d, d^2))$
- $ns^2 \ll d^2$  일 때 경쟁적

# Function Space 선택 (1/8)

**Value function 품질 측정:**

목표가 value-prediction일 때, 적절한 state distribution  $\mu$ 에 대한 MSE (Mean-Squared Error) 사용

**Learning의 관점:**

Function space  $F$ 에서 함수 선택:  $F = \{V_\theta \mid \theta \in \mathbb{R}^d\}$

**Approximation Error:**

$$\inf_{V_\theta \in F} \|V_\theta - V^*\|_\mu$$

- 더 큰 function space  $\Rightarrow$  작은 approximation error
- 예: Linear approximation에서 독립적인 features 추가

**문제:** Incomplete information을 사용하는 learning에서 function space 크기 증가는 양날의 검!

- Approximation error  $\downarrow$ , but Estimation error  $\uparrow$

## Function Space 선택 (2/8)

단순화된 분석:

Linear approximation + LSTD,  $\gamma = 0$ ,  $(X_t, R_{t+1})$  i.i.d.,  $X_t \sim \mu$

이 경우:  $V(x) = r(x) = \mathbb{E}[R_{t+1} | X_t = x]$

LSTD는 empirical loss의 minimizer 계산:

$$L_n(\theta) = \frac{1}{n} \sum_{t=0}^{n-1} (\theta^\top \phi(X_t) - R_{t+1})^2$$

핵심 질문:

- Feature dimensionality  $d$ 를 때 어떤 일이 발생하는가?
- Sample size  $n$ 과  $d$ 의 관계는?

# Function Space 선택 (3/8)

Overfitting 시나리오:

$d \geq n$ 이고 feature matrix가 full row rank라면:

- $L_n$ 의 minimum = 0
- LSTD solution  $\theta_n$ :  $\theta_n^\top \phi(X_t) = R_{t+1}$  for all  $t$
- Perfect fit on training data!

문제점:

- Noisy rewards  $\Rightarrow$  큰 estimation error
- $\|\theta_n^\top \phi - V\|_\mu$  매우 큼
- Overfitting: Model이 "noise"에 fitting됨

해결책: 더 작은  $d$  (더 작은  $F$ ) 선택

- Overfitting 감소
- But: Approximation error 증가
- $\Rightarrow$  Tradeoff 존재!

## Function Space 선택 (4/8)

Tradeoff 정량화:

$\theta^*$ : True loss의 minimizer

$$\theta^* = \arg \min_{\theta} L(\theta), \quad L(\theta) = \mathbb{E}[(\theta^\top \phi(X_t) - R_{t+1})^2]$$

$V_{\theta^*}$  는  $V$ 의  $F$ 로의 projection

핵심 아이디어:

- LSTD는  $\theta^*$ 를 근사
- 하지만 유한 샘플  $\Rightarrow$  Estimation error 발생
- Function space 크기에 따라 error가 달라짐

## Function Space 선택 (5/8)

통계적 bound (Györfi et al., 2002):

Bounded rewards ( $|R| \leq R$ ) + truncation 가정:

$$\mathbb{E}[\|\theta_n^\top \phi - V\|_\mu^2] \leq C_1 \frac{d(1 + \log n)}{n} + C_2 \|(\theta^*)^\top \phi - V\|_\mu^2$$

항목 분석:

- 첫 번째 항: Estimation error bound
  - $d$  증가  $\Rightarrow$  증가
  - $n$  증가  $\Rightarrow$  감소
- 두 번째 항: Approximation error
  - $d$  증가  $\Rightarrow$  (일반적으로) 감소

Constants:

- $C_1$ : Reward의 variance & range에 비례
- $C_2$ : Universal constant

## Function Space 선택 (6/8)

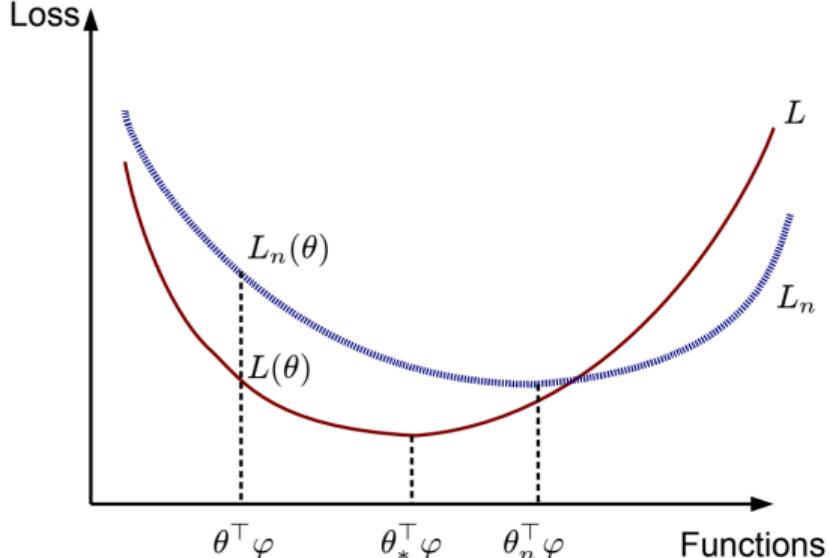


Figure 5: Convergence of  $L_n(\cdot)$  to  $L(\cdot)$ . Shown are the curves of the empirical loss,  $L_n$  and the true loss  $L$ , as a function of the parameter  $\theta$ . If the two curves are uniformly close to each other (i.e., for every  $\theta$ ,  $|L_n(\theta) - L(\theta)|$  is small), then one can expect that the loss of  $\theta_n^\top \varphi$  will be close to the loss of  $\theta_*^\top \varphi$ .

## Function Space 선택 (7/8)

일반화:

Similar bounds는 다음 경우에도 성립:

- $\gamma > 0$
- Dependent sequence ( $X_t; t \geq 0$ ) (잘 mixing하면)
- $\gamma \neq 0$ : Noise는  $R_{t+1}$ 과  $Y_{t+1}$  모두에서 발생

Control algorithms에서도:

- Fitted value iteration (Munos & Szepesvári, 2008)
- Fitted actor-critic (Antos et al., 2007)
- Approximate policy iteration (Antos et al., 2008)
- Finite-sample performance bounds 존재

결론: Approximation-Estimation tradeoff는 RL의 근본적인 문제

- Function space 너무 크면: Overfitting (high variance)
- Function space 너무 작으면: Underfitting (high bias)

## Function Space 선택 (8/9)

### 자동 Feature Selection & Regularization:

Function space 선택의 중요성과 어려움 인식 ⇒ 자동화 연구 증가

### Parsimonious basis functions 구성:

- Gaussian RBF 파라미터 튜닝 (Menache et al., 2005)
- Nonparametric techniques (Keller et al., 2006; Parr et al., 2007)
- Numerical analysis + nonparametric (Mahadevan, 2009)

### Supervised learning 기법 활용:

- Regression trees (Ernst et al., 2005)
- Kernel methods (Rasmussen & Kuss, 2004; Engel et al., 2005)
- $\ell_1$ -regularization (LASSO-inspired, Kolter & Ng, 2009)
- Regularization with statistical guarantees (Farahmand et al., 2009)

# Function Space 선택 (9/9)

개사기 Neural Network의 등장:

개례전드 Neural Networks가 제공하는 것:

- **Universal Approximation:** 충분한 크기의 네트워크는 임의의 연속 함수 근사 가능
- **Automatic Feature Learning:** Hand-crafted features 대신 데이터로부터 representation 학습
- **Scalability:** 고차원 state/action spaces 처리 가능
- **End-to-End Learning:** Raw sensory input부터 action까지 직접 학습

마침내. Deep Reinforcement Learning의 시작:

- DQN (Mnih et al., 2015): Atari 게임을 raw pixels로부터 학습
- 이전 방법들과 달리 feature engineering 불필요
- TD learning + Deep Neural Networks의 결합
- Experience replay & Target networks로 안정성 확보

현대 RL의 방향: Function approximation의 선택이 linear에서 deep neural networks로 확장

# Control

---

## **For Further Exploration**

---

# 참고문헌 i

주요 참고 자료:

1. Szepesvári, C. (2009).  
*Algorithms for Reinforcement Learning*  
Synthesis Lectures on Artificial Intelligence and Machine Learning,  
Morgan & Claypool Publishers  
<https://sites.ualberta.ca/~szepesva/rlbook.html>
2. Sutton, R. S., & Barto, A. G. (2018).  
*Reinforcement Learning: An Introduction (2nd ed.)*  
MIT Press
3. Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996).  
*Reinforcement Learning: A Survey*  
Journal of Artificial Intelligence Research (JAIR), 4, 237-285
4. Silver, D.  
*UCL Course on Reinforcement Learning - Lecture 1: Introduction to RL*

5. Zhou, Y. (2019).

*IE 498: Online Learning and Decision Making*

Fall 2019, University of Illinois at Urbana-Champaign

Teaching Assistant: Tanmay Gangwani

6. Bertsekas, D. P. (2019).

*Reinforcement Learning and Optimal Control*

Athena Scientific

7. Ng, A.

*CS229 Lecture Notes - Reinforcement Learning and Control*

Stanford University

8. Dayan, P., & Niv, Y. (2008).

*Reinforcement learning: The Good, The Bad and The Ugly*

Current Opinion in Neurobiology, 18(2), 185-196

## 참고문헌 iii

9. Mnih, V., et al. (2015).  
*Human-level control through deep reinforcement learning*  
Nature, 518(7540), 529-533
10. Pineau, J.  
*Beyond Bellman's Legacy: Rethinking What we Value*
11. Weng, L. (2018).  
*A (Long) Peek into Reinforcement Learning*  
Lil'Log Blog  
<https://lilianweng.github.io/posts/2018-02-19-rl-overview/>
12. Stack Exchange Discussion (2018).  
*When are Monte Carlo methods preferred over temporal difference ones?*  
Cross Validated  
<https://stats.stackexchange.com/questions/336974>

본 발표 자료는 위 교재와 논문들을 바탕으로 작성되었습니다.