Reinforce Learning

Dynamic Programming in RL

소프트웨어 꼰대 강의

노기섭 교수

(kafa46@hongik.ac.kr)

DP in RL

DP solves MDPs using Bellman equations.

MDP

↓

Dynamic Programming

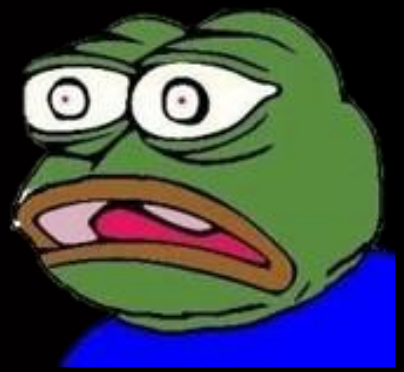
↓

Bellman Equations

↙ ↘

Policy Iteration
Iterative Evaluation & Improvement

Value Iteration
Direct Value Updates



Recap: Dynamic Programming (DP)

DP solves sequential decision problems by exploiting

- ✓ Optimal Substructure
- ✓ Overlapping Subproblems

Assumption: the environment of model is known

- ✓ $P(s'|s,a)$
- ✓ $R(s,a,s')$

Connection to RL:

Provides theoretical foundation
for many RL algorithms

- Value Iteration
- Policy Iteration
- Temporal Difference Methods
  (bootstrapping idea)

Bellman Expectation Equation

Dynamic Programming solves sequential decision problems by exploiting

- ✓ Optimal Substructure
- ✓ Overlapping Subproblems

The value of a state under policy π:

$$V^{\pi} = \mathbb{E}_{\pi}[G_t | s_t = s]$$

, where $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$

Bellman Equation in RL

DP works because of recursive structure
↓
Bellman equation provides this recursion
↓
Therefore, DP can solve RL problems

Recursive Form (Bellman Expectation Equation)

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

$V^{\pi}(s)$ ← Expected return from next states

Recursive structure enables iterative updates!

Bellman Expectation Equation: Recursive Value Definition

Recursive Form (Bellman Expectation Equation)

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Recursive structure
enables iterative updates!

Goal: Evaluate the value function of a given policy π

We want to compute: $V^{\pi}(s)$ which represent expected return starting from s & a following policy π

Bellman Expectation Update:

$$V_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

(omitted the π for simplicity)

New value
= expected immediate reward
+ discounted value of next states

Iterative Policy Evaluation via Bellman Updates

Iterative Policy Evaluation Algorithm

1) Initialize $V(s)$ arbitrarily
2) Repeat until convergence
3) Update value using the Bellman expectation equation

} V ← Bellman update

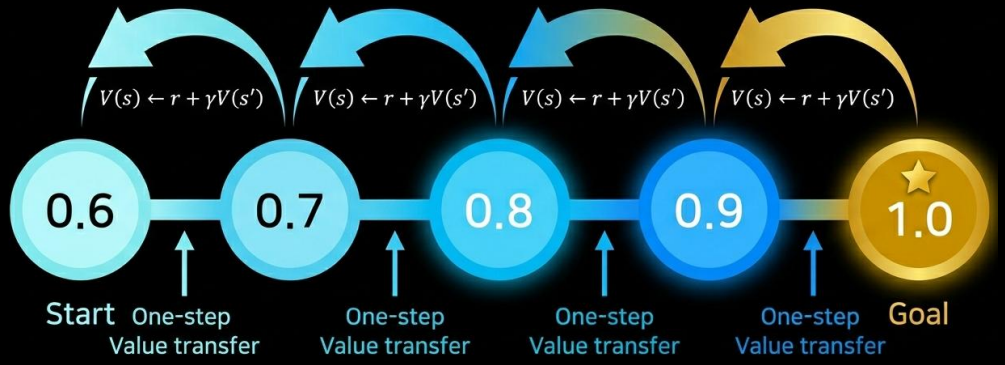
Value propagates:

Goal state value ⇨ Neighbor state value ⇨ Entire environment

Start → 0.6 → 0.7 → 0.8 → 0.9 → 1.0 (Goal)

Value Propagation: Learning Flows Backward

Value Propagation: Future → Current



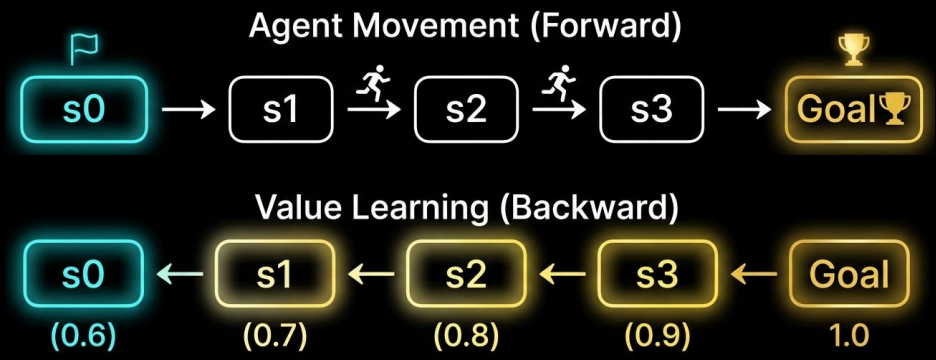
Iteration 1: Values appear near the goal

Iteration 2: Values propagate farther

Iteration 3: Values spread across the entire state space

...

Converged



The environment gives local rewards, but RL learns global consequences.

Value Propagation: Model vs Bellman Recursion

DP can propagate values exactly because it has the model,

but the propagation itself comes from Bellman recursion.

- DP: exact propagation
- RL: sample-based propagation

Policy Improvement (1/2)

After computing the value function of a policy,
we improve the policy.

Key Idea:

- Use the evaluated value function to improve the policy.
- The agent selects the action that leads to the highest expected value.

Policy Improvement Rule:

"Which action is best in that state."

$$\pi_{new}(s) = \arg\max_a Q^{\pi}(s,a)$$

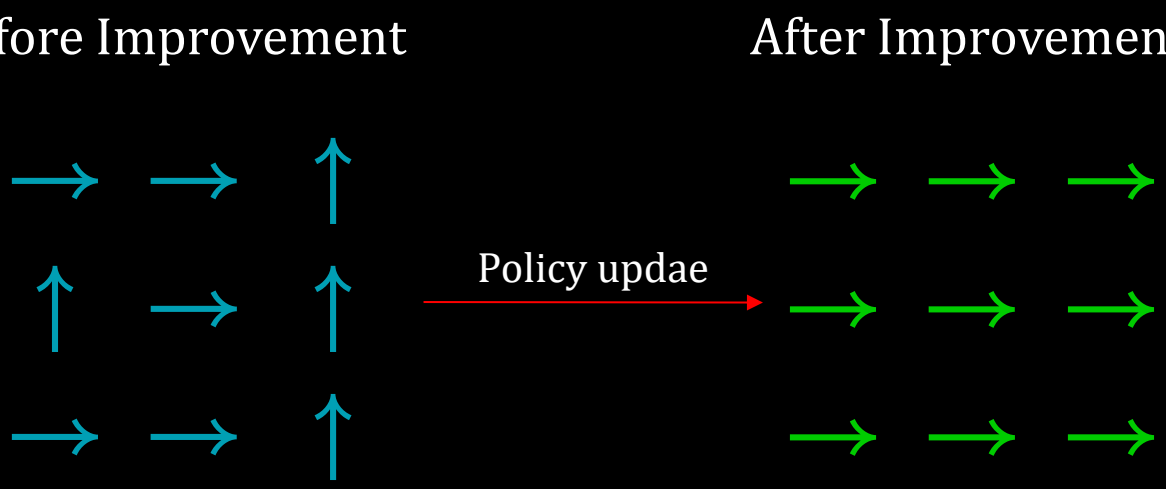
, where $Q^{\pi}(s,a) = \mathbb{E}[r + \gamma V^{\pi}(s')]$

"A new policy that is equal or better than the previous policy."

Policy Improvement (2/2)

Concept:

Greedy policy improvement



Agent chooses the best direction toward the goal.

Policy Iteration

Key Idea:

Policy Iteration alternates between **policy evaluation** and **policy improvement** until the policy converges to the **optimal policy**.

Algorithm:

Initialize a policy π

Repeat:

Policy Evaluation: $V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi\right]$

How good each state is

Policy Improvement: $\pi_{new}(s) = \arg\max_a Q^{\pi}(s, a)$

Which action is best

until policy no longer changes (converged)

Return the optimal policy π^* if exist, else return "Fail"

Policy Iteration은 MDP 문제를 해결하기 위한 "Dynamic Programming 방법" 중 하나임

Value Iteration: Concept

Key Observation (Disadvantage of Policy Iteration)
Full **policy evaluation** in Policy Iteration can be **computationally expensive**.

Core Idea:

- ✓ Value Iteration **combines policy evaluation and policy improvement into a single update step**.
- ✓ Instead of evaluating a policy fully, the algorithm **directly updates the value using the best action**.

Bellman Optimality Equation

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

Value Iteration Update Rule

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$$

After Convergence $\pi^*(s) = \arg\max_a Q^*(s,a)$

Value Iteration: Algorithm

Algorithm:

Initialize $V(s)$ arbitrarily

Repeat:

Value Update:

How good each state is

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$$

(Implicit) Policy Improvement:

$$\pi_{new}(s) = \arg\max_a \sum P(s'|s,a)[r + \gamma V(s')]$$

Which action is best

until $V(s)$ no longer changes (converged)

Return the optimal policy π^* if exist, else return "Fail"

Policy vs. Value Iteration

Policy Iteration vs Value Iteration

Fully evaluate before improving

Skip full evaluation, improve during update

Asynchronous Dynamic Programming

Key idea:

Focus updates on states with the largest expected improvement.

Instead of updating all states simultaneously, asynchronous methods update selected states.

Examples:

- Asynchronous Value Iteration
- Prioritized Sweeping

Limitations of Dynamic Programming in RL

1. Model Requirement: Not easy to apply

DP assumes that the **environment model is known**.

$$P(s'|s,a) \qquad R(s,a)$$

In many real-world problems,
the transition probabilities and reward functions are **unknown**.

2. State Explosion: curse of dimensionality

The number of state–action pairs grows rapidly:

$$|S| \times |A|$$

As the environment becomes more complex,
the computational cost becomes **prohibitively large**.

3. Continuous State or Action Spaces: Not real world-based

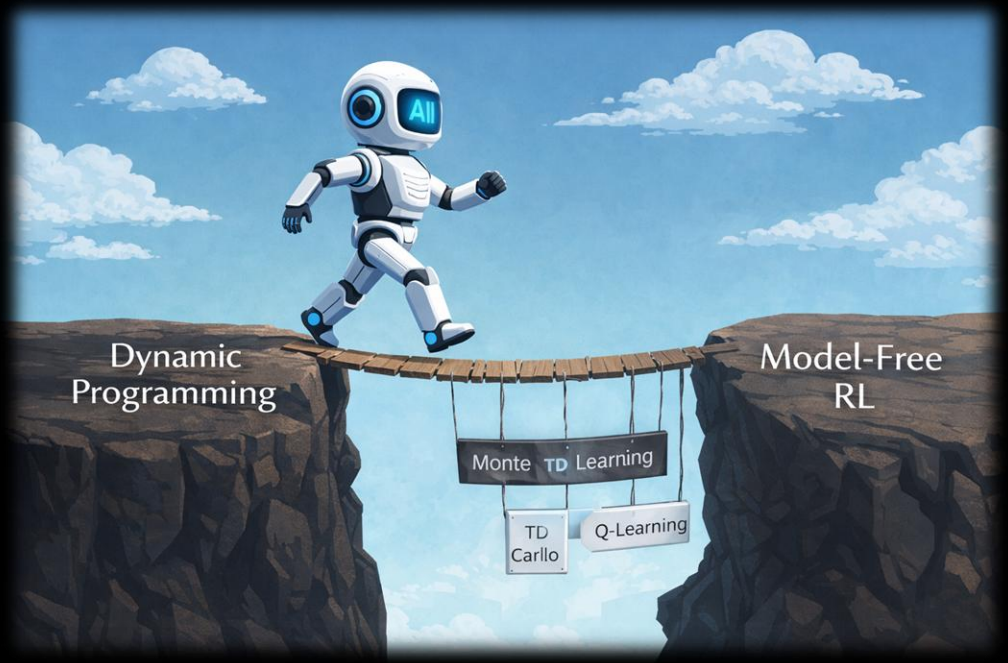
DP typically assumes **discrete states and actions**.

Bridge to Next RL

DL limitations motivate model-free reinforcement learning methods,

Such as:

- ✓ Monte Carlo (MC) methods
- ✓ Temporal Difference (TD) learning
- ✓ Q-learning



Role of DP

Dynamic Programming motivates many RL algorithms.

Dynamic Programming Concept	Reinforcement Learning Equivalent
Policy Evaluation	Temporal Difference (TD) Learning
Policy Improvement	Greedy / ε-greedy Policy
Policy Iteration	Actor–Critic Methods
Value Iteration	Q-learning

Foundations of Reinforcement Learning

Three fundamental learning paradigms:

Method	Key Characteristics
Dynamic Programming (DP)	Model-based, bootstrapping
Monte Carlo Methods (MC)	Model-free, uses full return
Temporal Difference (TD)	Model-free, bootstrapping

Evolution of Reinforcement Learning Methods

DP → MC → TD

Therefore, TD = MC + DP

TD provides efficient online learning without requiring a model.

- ✓ From MC → learning from sampled experience
- ✓ From DP → bootstrapping with value estimates

수고하셨습니다 ..^^..