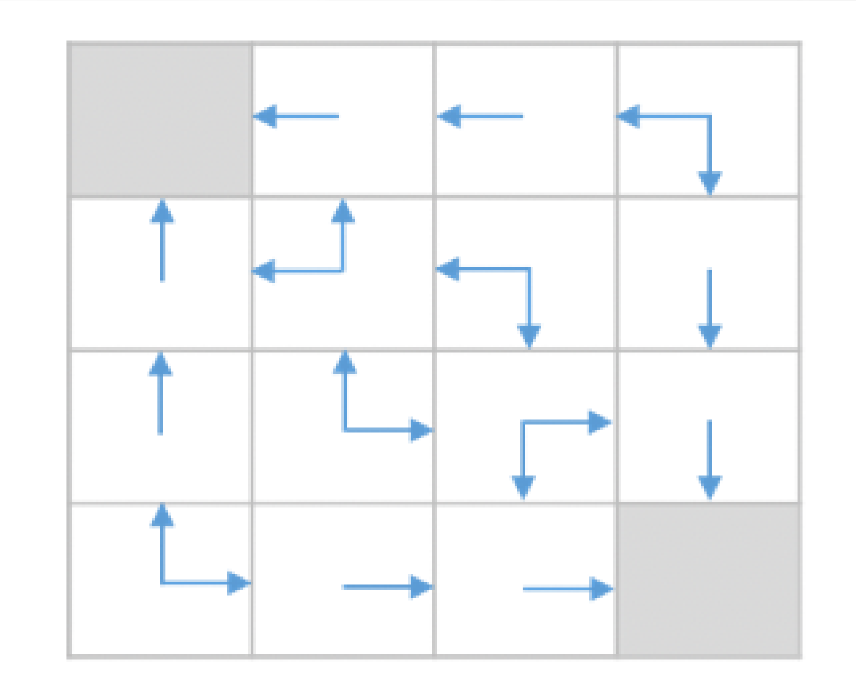
Chapter 06. 스스로 전략을 짜는 강화학습 (Reinforcement Learning)

Dynamic Programming



Bellman Equation

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

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

Markov Decision Process

Optimal state-value function and optimal action-value function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_\pi(s)$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Markov Decision Process

optimal policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy* π_* *that is better than or equal to all other policies,* $\pi_* \geq \pi, \forall \pi$
- *All optimal policies achieve the optimal value function,* $v_{\pi_*}(s) = v_*(s)$
- *All optimal policies achieve the optimal action-value function,* $q_{\pi_*}(s,a) = q_*(s,a)$

Planning & Learning

•Planning

앞서 배운 environment에 대한 model 을 가지고 있는 경우, Markov Decision Process 에 대한 full knowlege 를 가지고 있게 된다. 이를 planning 이라고 하며 MDP 의 정보를 기반한다.

•Learning

Learning이란 environment의 mod

을 말합니다.

•Process of Planning

- For prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function v_π
- Or for control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function v_*
  - and: optimal policy π_*

Dynamic Programming

https://zzsza.github.io/data/2019/01/06/dynamic-programming/

•Dynamic Programming의 조건

1) Optimal substructure
- 작은 문제로 나뉠 수 있어야 함

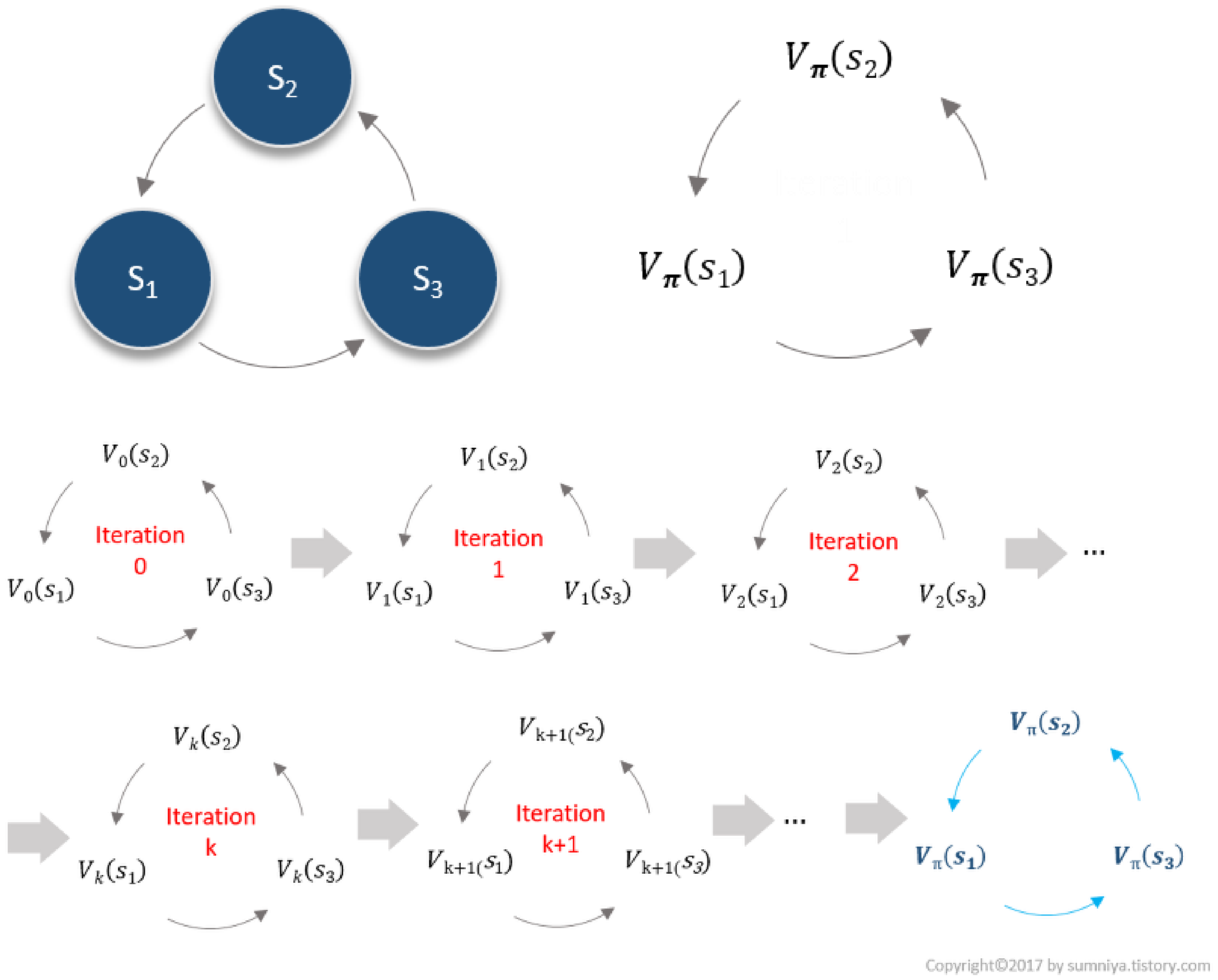
2) Overlapping subproblems
- 한 서브 문제를 풀고 나온 솔루션을 저장해(cached) 다시 사용할 수 있음

•MDP는 이 조건을 만족함
- Bellman 방정식이 recursive
- value fur ... 이 작은 문제들의 해

$$v(s) := \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

Dynamic Programming

https://sumniya.tistory.com/10



Dynamic Programming

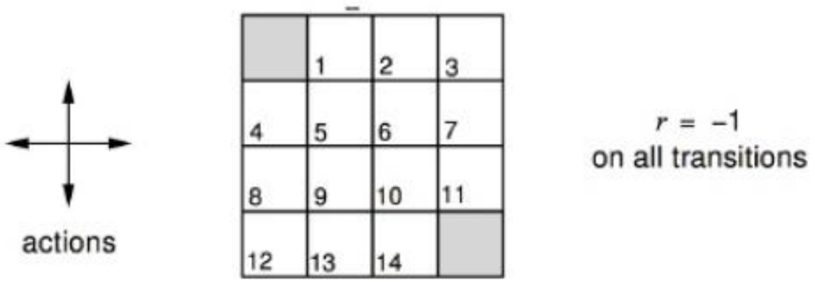
•Policy Iteration

1. Initialize π randomly
2. Repeat until converge
   - Let V=Vπ.
   - For each state s, let $\pi(s) = \arg\max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$.

value function VV에 대해 greedy한 policy update rule이라고 부른다. Policy iteration 역시 polynomial time 안에 optimal policy로 수렴하게 된다.

Dynamic Programming

•Policy Evaluation



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, ..., 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

fast campus

Dynamic Programming

https://sumniya.tistory.com/10

•Policy Evaluation

k = 0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

at (1,2) state

Up $V_1(s) = 0.25 \times (-1 + 0)$

Down $V_1(s) = 0.25 \times (-1 + 0)$

Left $V_1(s) = 0.25 \times (-1 + 0)$

Right $V_1(s) = 0.25 \times (-1 + 0)$

$\therefore V_1(s) = 4 \times 0.25 \times (-1) = \mathbf{-1}$

https://sumniya.tistory.com/10

Dynamic Programming

•Policy Evaluation

k = 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,2) state

Up $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Down $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Left $V_2(s) = 0.25 \text{ X } (-1 + 0)$
Right $V_2(s) = 0.25 \text{ X } (-1 + -1)$

$\therefore V_2(s) = 3 \text{ X } 0.25 \text{ X } (-2) + 0.25 \text{ X } (-1) = \mathbf{-1.75}$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,3) state

Up $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Down $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Left $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Right $V_2(s) = 0.25 \text{ X } (-1 + -1)$

$\therefore V_2(s) = 4 \text{ X } 0.25 \text{ X } (-2) = \mathbf{-2}$

https://sumniya.tistory.com/10

Dynamic Programming

•Policy Evaluation

0	-1.75	-2	-2
-1.75	-2	-2	-1
-1	-2	-2	-1.75
-1	-1	-1.75	0

k = 2

0	-2.438	-2.938	-3
-2.438	-2.938	-3	-2.938
-2.938	-3	-2.938	-2.438
-3	-2.938	-2.438	0

k = 3

...

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

k = ∞

https://sumniya.tistory.com/10

Dynamic Programming

•Policy Improvement

at state 1

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

True value func.

Up $q_\pi(1, 0) = -1 + (-14)$
Down $q_\pi(1, 1) = -1 + (-18)$
Left $q_\pi(1, 2) = -1 + (\ 0\ )$
Right $q_\pi(1, 3) = -1 + (-20)$

$\therefore\ max\ q_\pi(1, a) = q_\pi(1, \textbf{\textit{Left}})$

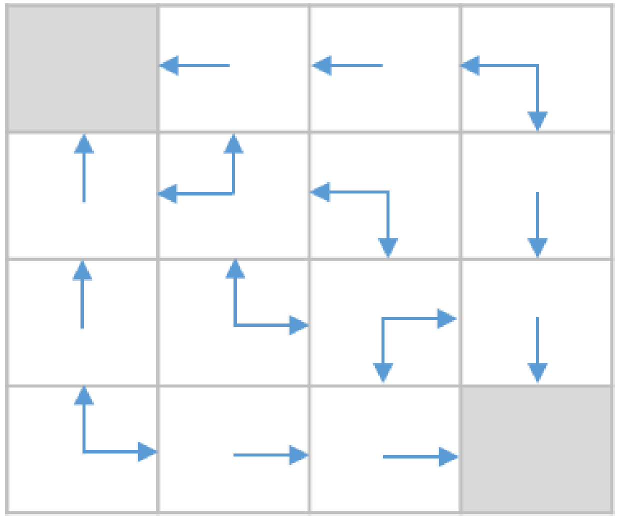
at state 5

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

True value func.

Up $q_\pi(5, 0) = -1 + (-14)$
Down $q_\pi(5, 1) = -1 + (-20)$
Left $q_\pi(5, 2) = -1 + (-14)$
Right $q_\pi(5, 3) = -1 + (-20)$

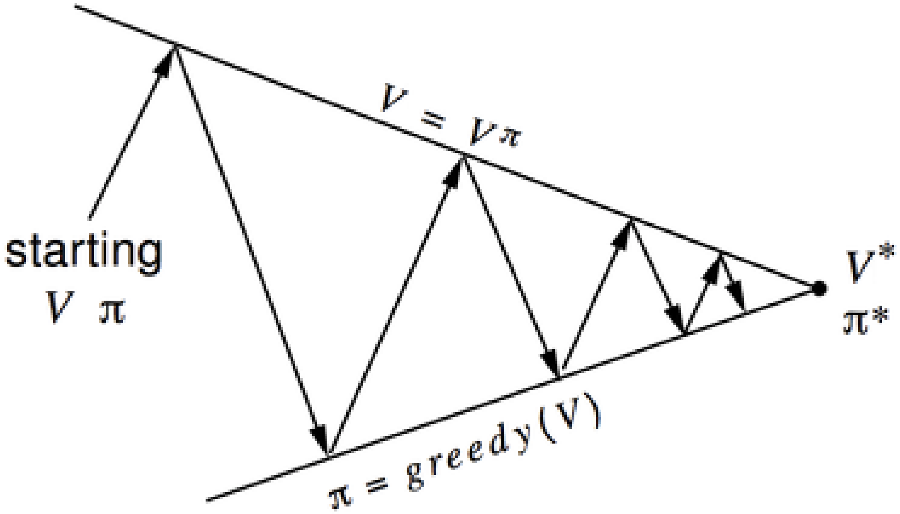
$\therefore\ max\ q_\pi(5, a) = q_\pi(5, \textbf{\textit{Up}})\ or\ q_\pi(5, \textbf{\textit{Left}})$

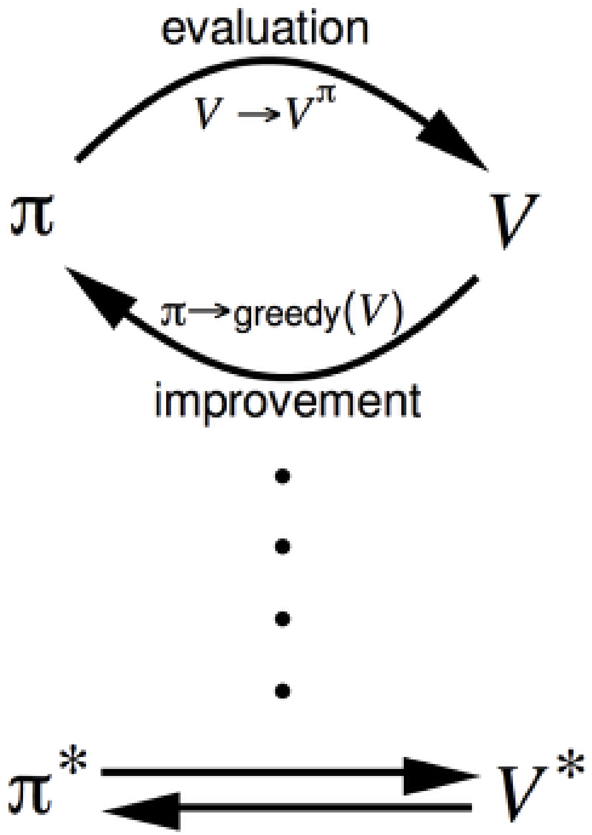


The result of GPI

Dynamic Programming

- **Policy Iteration**





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Dynamic Programming

•Value Iteration

Value iteration 알고리즘은 다음과 같다.

1.Initialize V(s)=0, for all s.

2.Repeat until converge

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{a'} P_{sa}(s')V(s'), \quad \text{for all } s.$$

Bellman Optimality Eqn.을 evaluation을 한번만 진행.

우리는 evaluation과정에서 이동가능한 state s'들에 대해 모든 value func.들을 더하여 도출했지만, 이중에 max값을 취해서 greedy하게 value func.을 구해서 improve 버리자는게 Value Iteration의 아이디어입니다. 그래서 우리는 action을 취할 확률을 곱해서 summation하는 대신에 max값을 취하는 아래의 optimal value func.식을 사용합니다.

Dynamic Programming

- **Value Iteration**

k = 0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

at (1,2) state : state 1

Up $V_1(s) = -1 + 0$
Down $V_1(s) = -1 + 0$
Left $V_1(s) = -1 + 0$
Right $V_1(s) = -1 + 0$

$\therefore V_1(1) = \max V_1(s) = \mathbf{-1}$

k = 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,2) state : state 1

Up $V_2(s) = -1 + (-1)$
Down $V_2(s) = -1 + (-1)$
Left $V_2(s) = -1 + (0)$
Right $V_2(s) = -1 + (-1)$

$\therefore V_2(s) = \max V_2(s) = \mathbf{-1}$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,3) state : state 2

Up $V_2(s) = -1 + (-1)$
Down $V_2(s) = -1 + (-1)$
Left $V_2(s) = -1 + (-1)$
Right $V_2(s) = -1 + (-1)$

$\therefore V_2(s) = \max V_2(s) = \mathbf{-2}$

Dynamic Programming

•Value Iteration

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-1
-2	-2	-1	0

k = 2

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

k = 3

…

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

k = ∞

Dynamic Programming

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

- *Thank you*