Dynamic Programming

Slides from

1. 이웅원 외, 파이썬과 케라스로 배우는 강화학습, 주교재
2. 이웅원, 가깝고도 먼 DeepRL, PPT
3. David Silver, Reinforcement Learning, PPT

References

1. Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, MIT Press
2. 유튜브, 전민영, 노승은, 강화학습의 기초 이론, 팡요랩

Contents

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. 문제에 대한 수학적 정의 : Markov Decision Process
3. MDP를 계산으로 푸는 방법 : Dynamic Programming
4. MDP를 학습으로 푸는 방법 : Reinforcement Learning
5. 상태공간이 크고 차원이 높을 때 쓰는 방법 : Function Approximation
6. 바둑과 같은 복잡하고 어려운 문제를 푸는 방법 : Deep Reinforcement Learning

MDP 문제 풀이 방법

1. 강화학습이 풀고자 하는 문제 : Sequential Decision Problem
2. MDP : Sequential Decision Problem의 수학적 정의
3. MDP의 목표는 최대의 보상을 받는 것
-> 매 타임스텝마다 행동 선택의 기준 : 보상 -> 가치함수로 표현
-> 더 큰 보상을 얻을 수 있는 행동을 선택 : 정책

4. MDP 문제 풀이 방법
1) Dynamic Programming : 환경에 대한 모든 정보를 알고 가장 좋은 정책을 "계산"
2) Reinforcement Learning : 환경과의 상호작용을 통해 가장 좋은 정책을 "학습"

What is Dynamic Programming?

Dynamic sequential or temporal component to the problem

Programming optimising a "program", i.e. a policy

- c.f. linear programming

- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

David Silver, Reinforcement Learning, PPT

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
  - *Principle of optimality* applies
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused
- Markov decision processes satisfy both properties
  - Bellman equation gives recursive decomposition
  - Value function stores and reuses solutions

David Silver, Reinforcement Learning, PPT

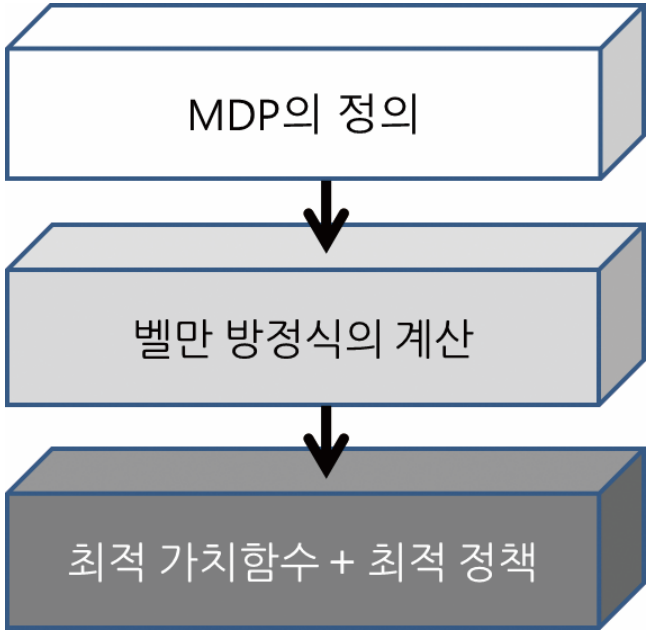
Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- 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_{*}

David Silver, Reinforcement Learning, PPT

Dynamic Programming in MDP

- 순차적 행동 결정 문제를 Dynamic Programming으로 풀어 나가는 과정
  1. 순차적 행동 문제를 MDP로 전환한다
  2. 가치함수를 벨만 방정식으로 반복적으로 계산한다
  3. 최적 가치함수와 최적 정책을 찾는다



- Dynamic Programming의 유래
  - ➢1953년 Richard E. Bellman, https://en.wikipedia.org/wiki/Richard_E._Bellman
  - ➢MDP를 벨만 방정식으로 풀기 위한 방법

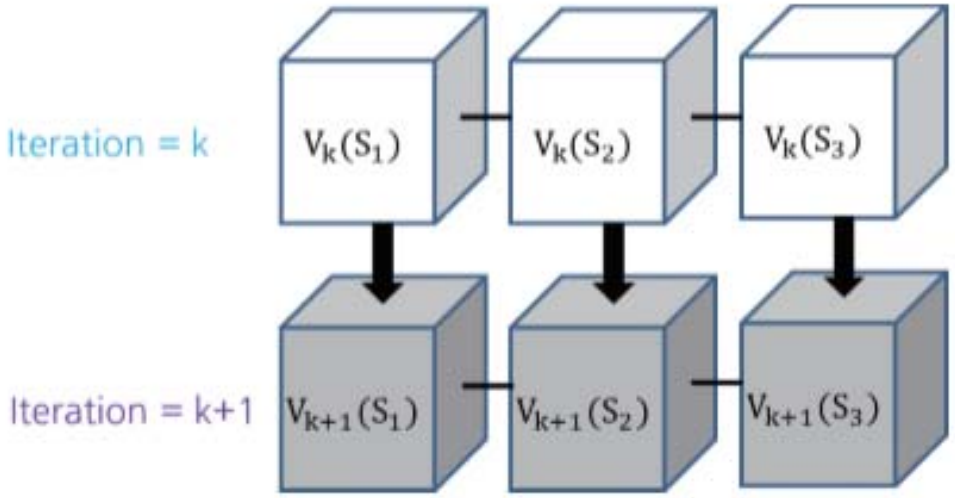
Dynamic Programming in MDP

1. MDP의 목표는 보상을 최대로 받는 정책을 구하기 : π^*
2. 큰 문제 : $\pi \rightarrow \pi^*$ (처음 π로부터 시작해서 π^*을 구하기)
   - 명시적 π : Policy Iteration
   - 내재적 π : Value Iteration
3. 작은 문제 : $\pi_k \rightarrow \pi_{k+1}$ (1 Iteration)
4. 반복되는 작은 문제를 푸는 방법 : Bellman Eq. (기대 or 최적)
5. 저장되는 값 : 가치함수

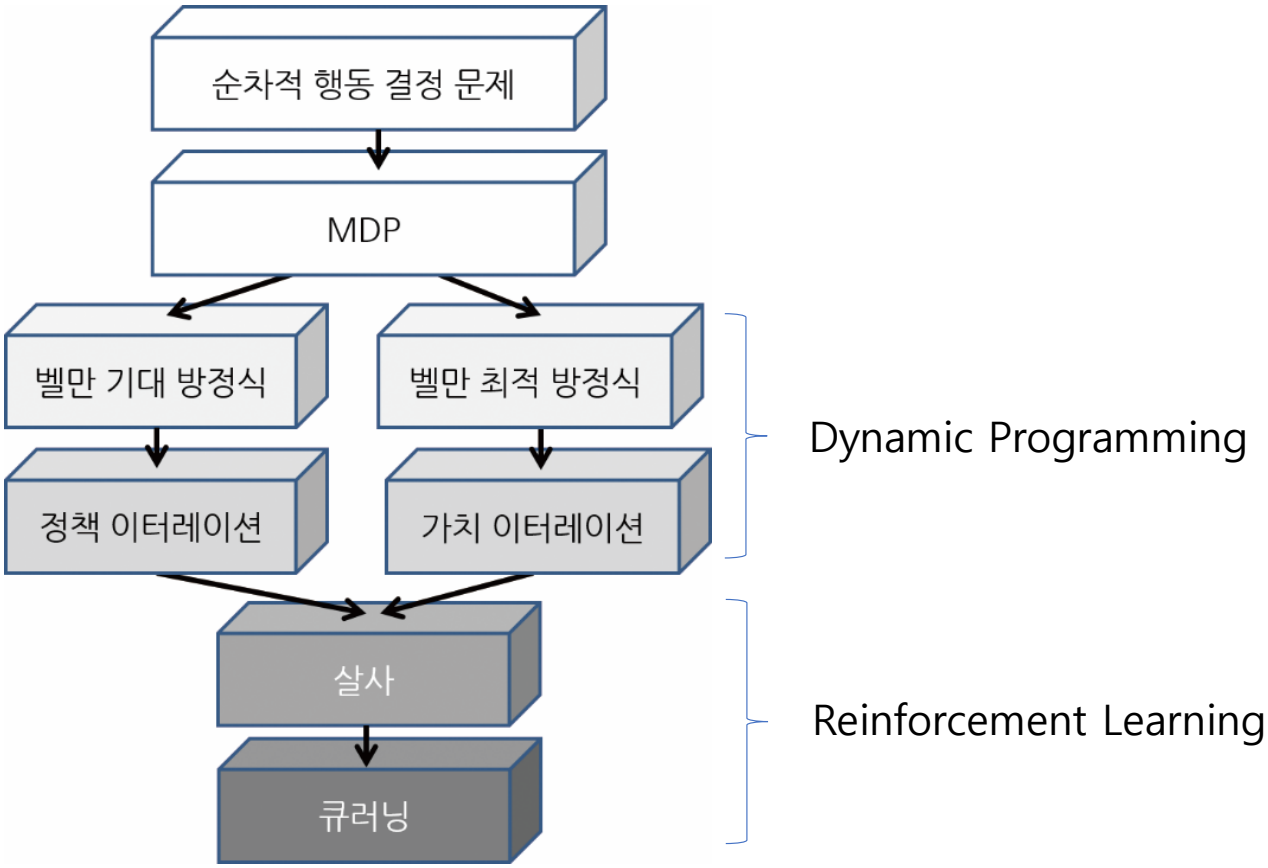
Dynamic Programming in MDP

1. 저장되는 값이 가치함수이므로 결국 가치함수의 업데이트
2. 벨만 방정식으로 가치함수 업데이트의 반복적 계산

$$v_0(s) \rightarrow v_1(s) \rightarrow v_2(s) \rightarrow v_3(s) \rightarrow \cdots \rightarrow v_\pi(s)$$



강화학습 알고리즘의 흐름



Policy Iteration

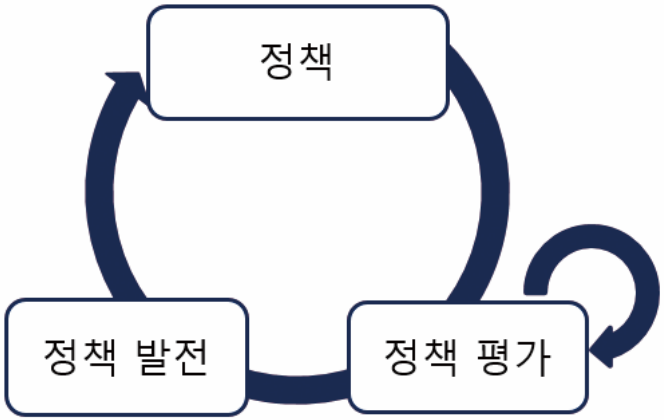
1) Policy Evaluation
2) Policy Improvement

Policy Iteration

- 벨만 기대방정식을 이용해 MDP로 정의되는 문제를 DP로 푸는 것

- 목적: Policy Iteration을 통해 가장 높은 보상을 얻게 하는 정책을 찾는 것

  => 정책 평가(Policy Evaluation)와 정책 발전(Policy Improvement)를 반복함 => 최적의 정책으로 수렴



Policy Evaluation

- 현재 정책에 대한 가치 함수를 평가함

$$v_\pi(s) = \boldsymbol{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \,|\, S_t = s]$$

- 벨만 기대방정식을 통한 가치함수 계산 -> DP

$$v_\pi(s) = \boldsymbol{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

Assume $P^a_{ss'} = 1$

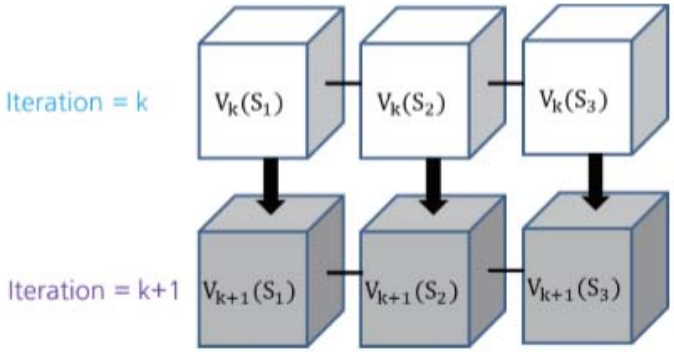
$$v_\pi(s) = \sum_{a \in A} \pi(a|s)(R_{t+1} + \gamma v_\pi(s'))$$

- 정책 평가 : 아래 가치함수를 k=1,2,3,...에 대해 반복적으로 계산

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R^a_s + \gamma v_k(s'))$$

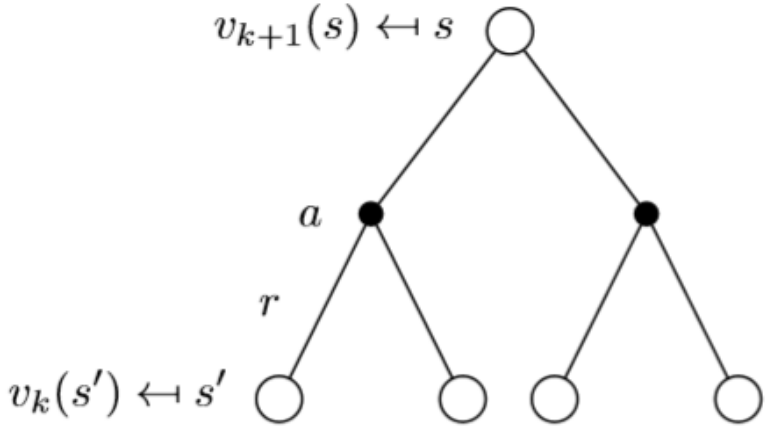
Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_\pi$
- Using *synchronous* backups,
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s



David Silver, Reinforcement Learning, PPT

Iterative Policy Evaluation



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

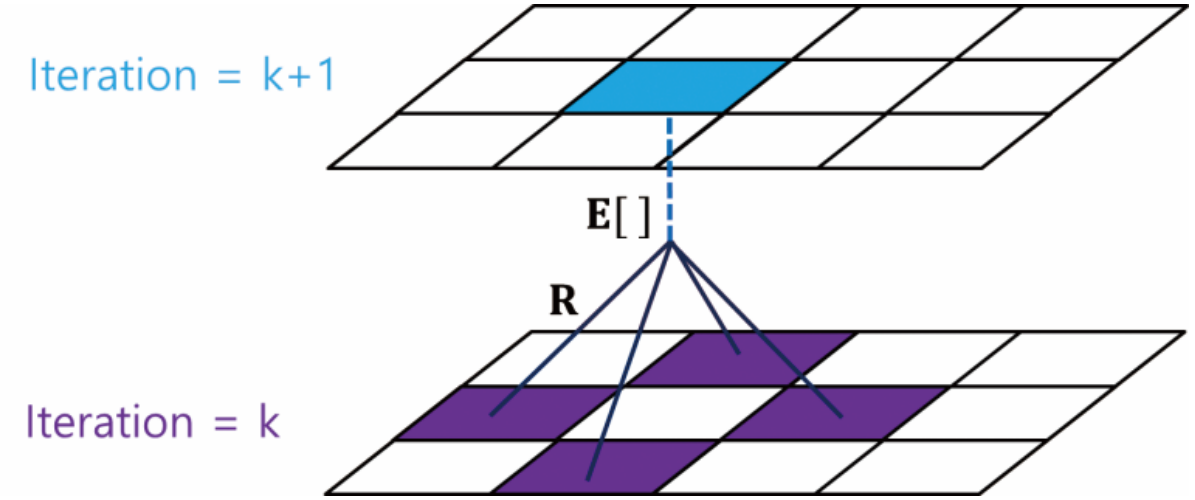
$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

David Silver, Reinforcement Learning, PPT

Iterative Policy Evaluation

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma v_k(s'))$$

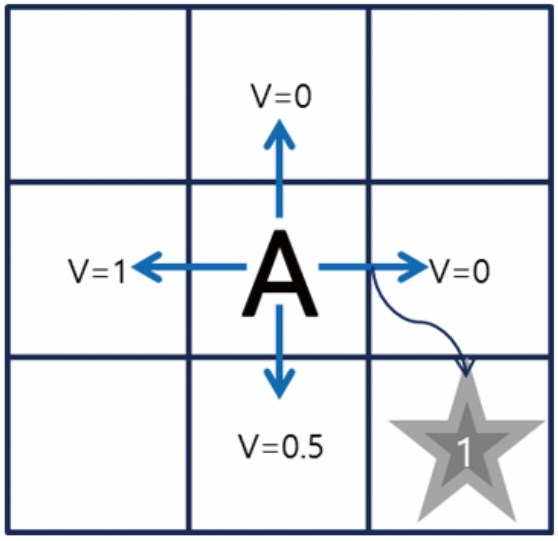
Assume $P_{ss'}^a = 1$



Ex1: Grid world에서 Iterative Policy Evaluation

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma v_k(s'))$$

Assume $P_{ss'}^a = 1$



상: $0.25 \times (0 + 0.9 \times 0) = 0$

하: $0.25 \times (0 + 0.9 \times 0.5) = 0.1125$

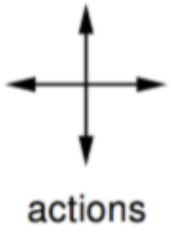
좌: $0.25 \times (0 + 0.9 \times 1) = 0.225$

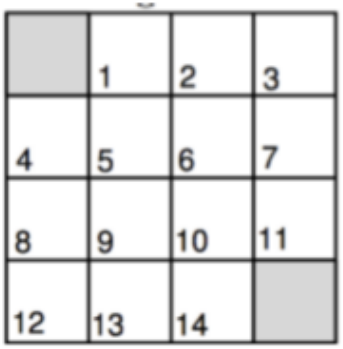
우: $0.25 \times (1 + 0.9 \times 0) = 0.25$

다음 가치함수 = 0 + 0.1125 + 0.225 + 0.25

$\pi(\text{상}|s) = \pi(\text{하}|s) = \pi(\text{좌}|s) = \pi(\text{우}|s) = 0.25, \gamma = 0.9$

Ex2: Evaluating a Random Policy in the Small Gridworld





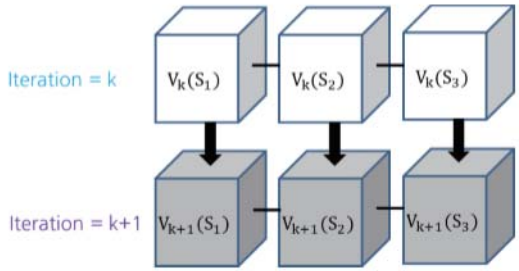
$r = -1$
on all transitions

- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- 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

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

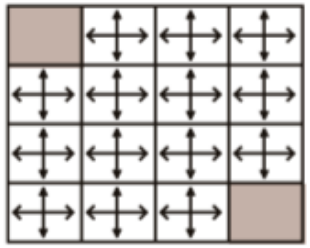
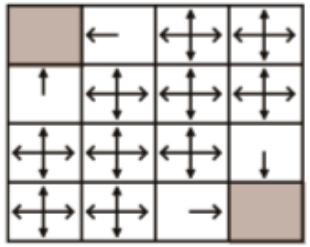
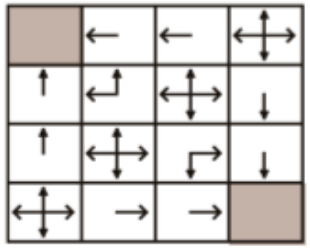
David Silver, Reinforcement Learning, PPT

Ex2: Iterative Policy Evaluation in Small Gridworld



$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma v_k(s'))$$

Assume γ=1

	v_k for the Random Policy	Greedy Policy w.r.t. v_k
$k = 0$	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	 ← random policy
$k = 1$	0.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 -1.0 0.0	
$k = 2$	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0	

$v_2(2,1)$ = 0.25*(-1+0) + 0.25*(-1-1) + 0.25*(-1-1) + 0.25*(-1-1) = -1.75

David Silver, Reinforcement Learning, PPT

Ex2: Iterative Policy Evaluation in Small Gridworld (2)

$k = 3$

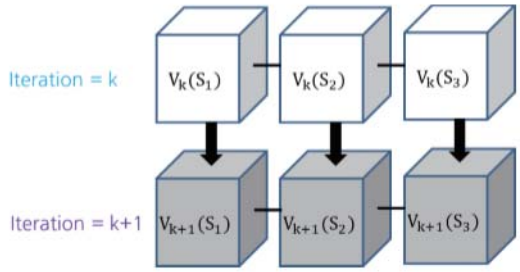
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

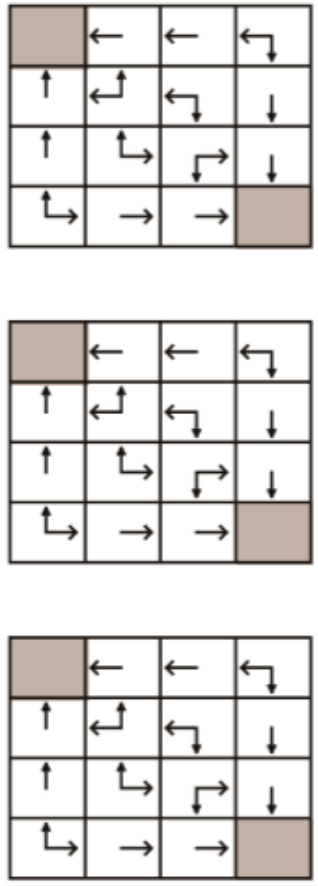
$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0





optimal policy

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma v_k(s'))$$

Assume γ=1

David Silver, Reinforcement Learning, PPT

How to Improve a Policy

- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

David Silver, Reinforcement Learning, PPT

Example: Grid world에서 Policy Improvement

- 처음에는 무작위로 행동을 하는 정책으로부터 시작함

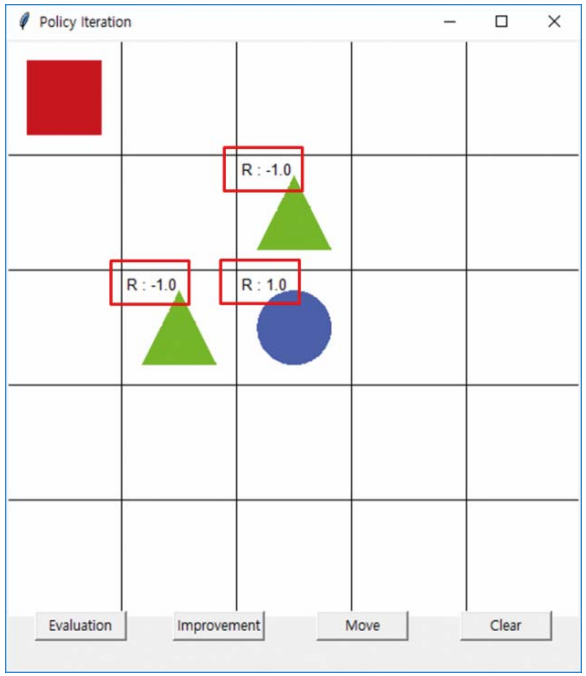
$\pi(\text{up}|s)$

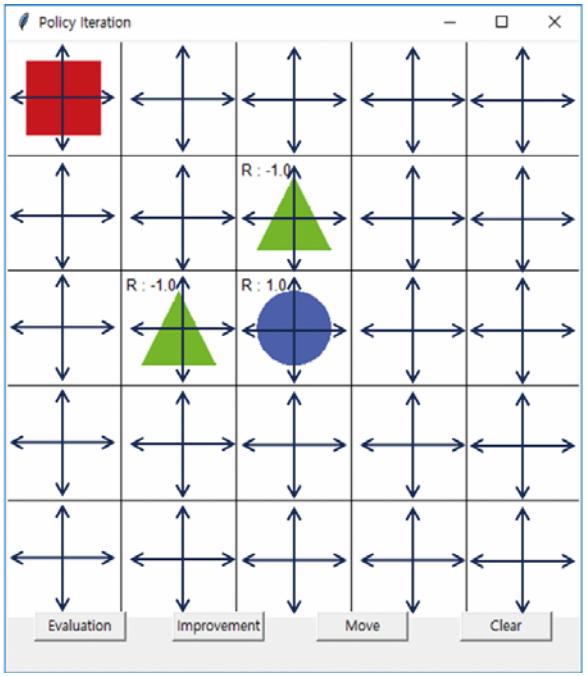
$= \pi(\text{down}|s)$

$= \pi(\text{left}|s)$

$= \pi(\text{right}|s)$

$= 0.25$





Example: Grid world에서 Policy Improvement

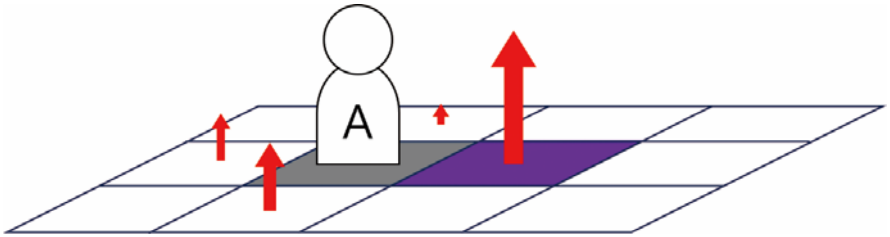
- 큐함수 : 어떤 행동이 좋은지를 알려 줌

$$q_\pi(s,a) = \boldsymbol{E}_{\boldsymbol{\pi}}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

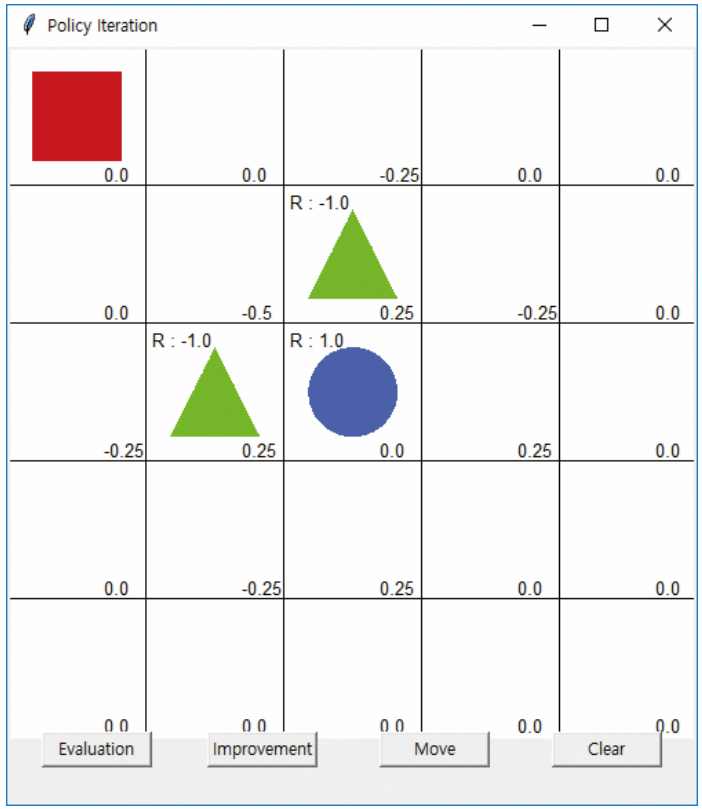
$$q_\pi(s,a) = R_s^a + \gamma v_k(s') \qquad \text{Assume } P_{ss'}^a = 1$$

- Greedy Policy Improvement : 상태 s에서 선택 가능한 행동의 $q_\pi(s,a)$를 비교하여 그 중에서 가장 큰 큐함수를 가지는 행동을 선택

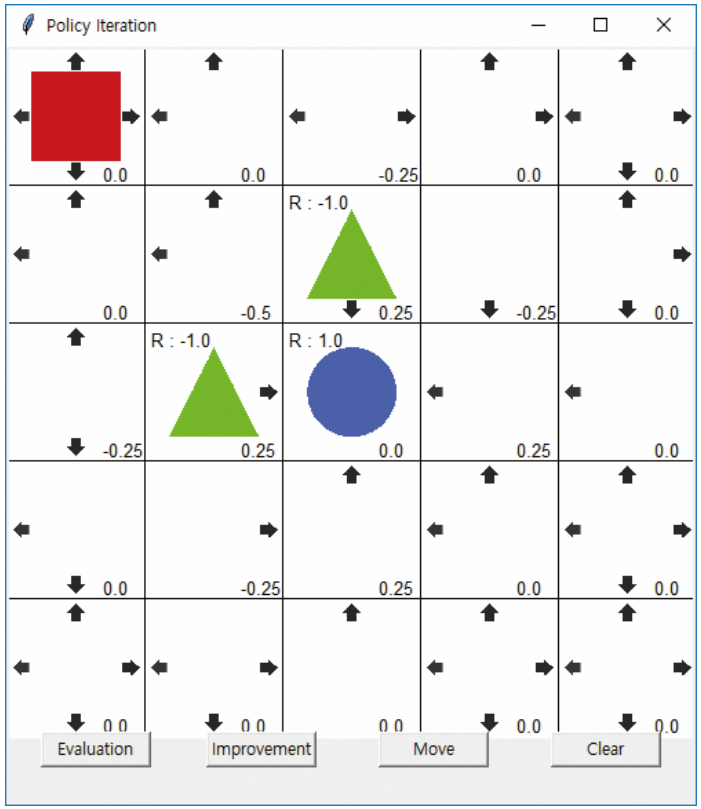
$$\pi'(s) = \text{argmax}_{a \in A}[q_\pi(s,a)]$$



Example: Grid world에서 Policy Iteration



한번 evaluation



한번 improvement

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

David Silver, Reinforcement Learning, PPT

Policy Improvement (2)

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

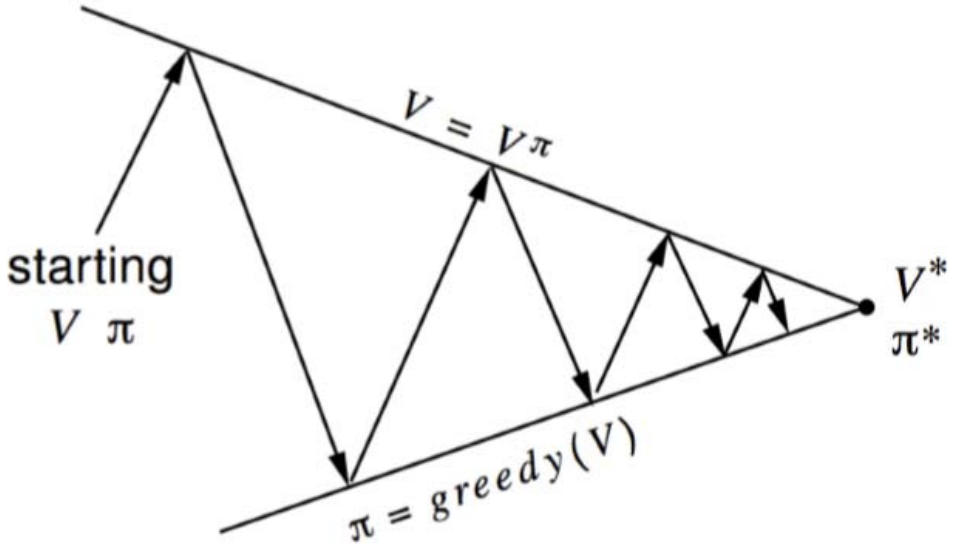
- Then the Bellman optimality equation has been satisfied

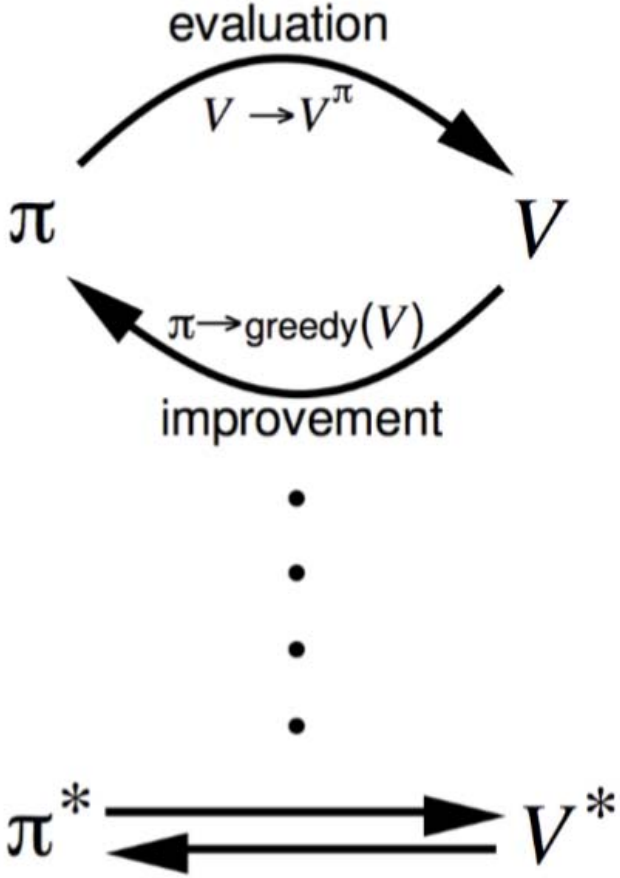
$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

David Silver, Reinforcement Learning, PPT

Summary of Policy Iteration





Policy evaluation Estimate v_π
Iterative policy evaluation

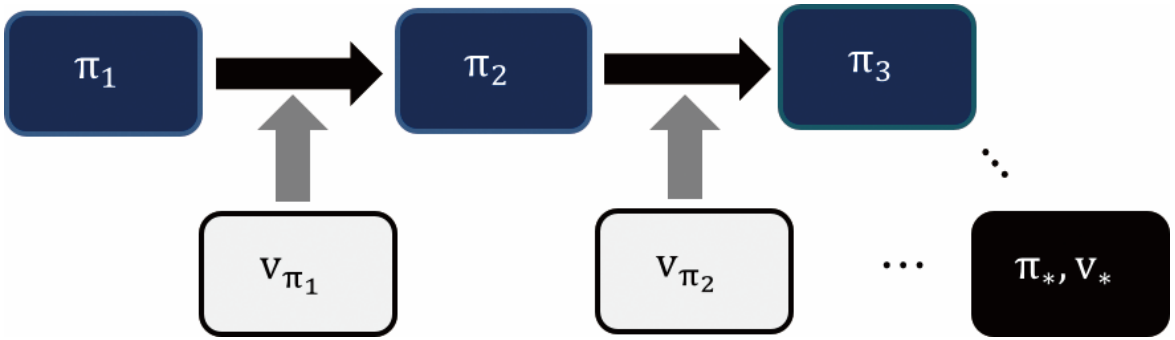
Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

David Silver, Reinforcement Learning, PPT

Value Iteration

Policy Iteration vs Value Iteration

1) Policy Iteration
   - 정책과 가치함수의 분리 -> Explicit Policy
   - 정책이 분리되어 있으므로 어떤 형태의 정책도 가능 -> Deterministic Policy or Stochastic Policy -> 벨만 기대 방정식



Policy Iteration vs Value Iteration

2) Value Iteration

- Optimal policy는 deterministic 함 (Deterministic Policy)
- 현재의 가치함수를 최적이라고 가정하고 그 가치함수에 결정적 정책을 적용하면?

  -> 벨만 최적방정식에 대해 DP를 반복적으로 수행하면?

  -> 진짜 최적 가치함수에 도달 -> 최적 정책

- Implicit Policy : 정책이 가치함수 안에 내재되어 있으므로 가치함수를 업데이트하면 자동으로 정책 또한 발전

Policy Iteration vs Value Iteration

1) Policy Iteration
   - 정책 평가: 벨만 기대방정식 -> 반복 계산 -> 참 가치함수

$$v_\pi(s) = \boldsymbol{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$$

   - 정책 발전 -> 최적 정책, 최적 가치함수

2) Value Iteration
   - 벨만 최적방정식 -> 반복 계산 -> 최적 가치함수 (최적 정책에 대한 참 가치 함수)

$$v^*(s) = \max_a \boldsymbol{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a]$$

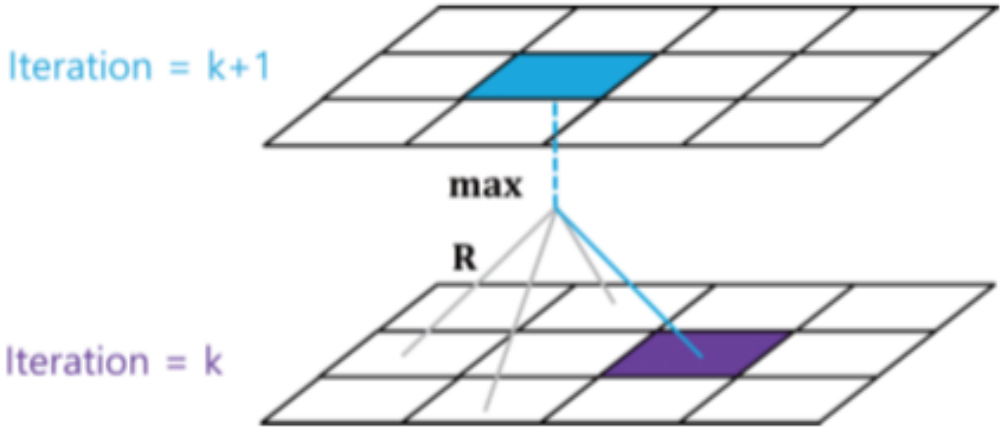
   -> 정책 발전이 필요 없음
   -> 시작부터 최적 정책을 가정했기 때문에 반복 계산을 통해 최적 가치함수 / 최적 정책이 구해짐 -> MDP가 풀렸음

Value Iteration

- 모든 상태에 대해서 벨만 최적 방정식을 통한 업데이트

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

Assume $P_{ss'}^a = 1$



Deterministic Value Iteration

- If we know the solution to subproblems $v_*(s')$
- Then solution $v_*(s)$ can be found by one-step lookahead

$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards

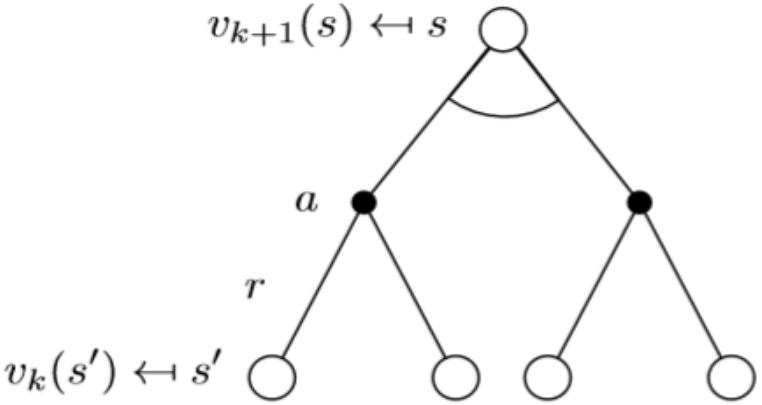
David Silver, Reinforcement Learning, PPT

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_*$
- Using synchronous backups
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_* will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

David Silver, Reinforcement Learning, PPT

Value Iteration (2)



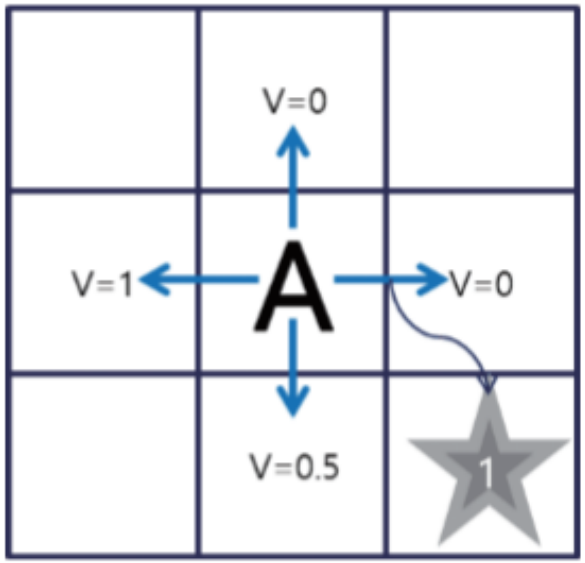
$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\boldsymbol{a}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{a}} \mathbf{v}_k$$

David Silver, Reinforcement Learning, PPT

Example: Grid world에서 Value Iteration

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

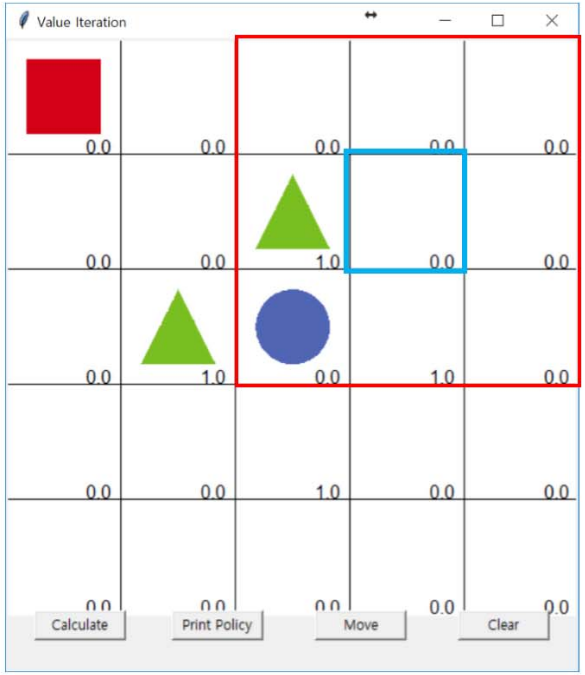


- '상' : $0 + 0.9 \times 0 = 0$
- '하' : $0 + 0.9 \times 0.5 = 0.45$
- '좌' : $0 + 0.9 \times 1 = 0.9$
- '우' : $1 + 0.9 \times 0 = 1$

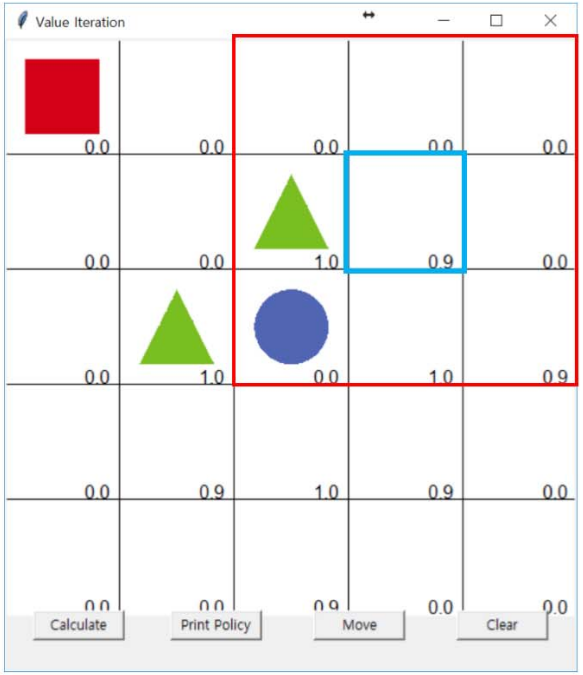
$$v_{k+1}(s) = \max[0, 0.45, 0.9, 1] = 1$$

Example: Grid world에서 Value Iteration

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$



K=1



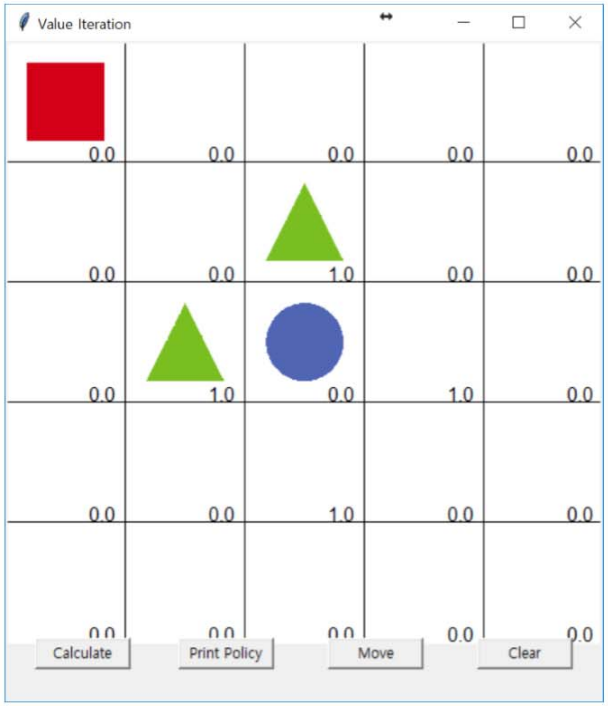
K=2

- '상' : $0 + 0.9 \times 0 = 0$
- '하' : $0 + 0.9 \times 1.0 = 0.9$
- '좌' : $-1 + 0.9 \times 1.0 = -0.1$
- '우' : $0 + 0.9 \times 0 = 0$

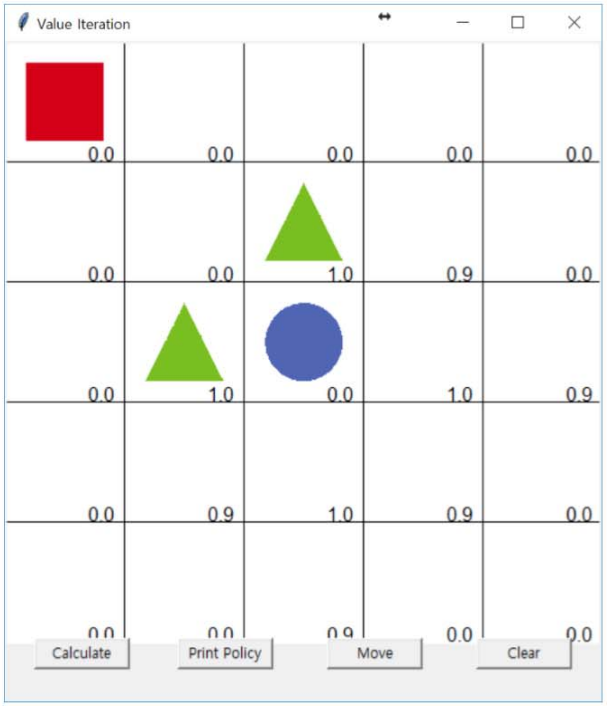
$$v_{k+1}(s) = \max[0, 0.9, -0.1, 0] = 0.9$$

Example: Grid world에서 Value Iteration

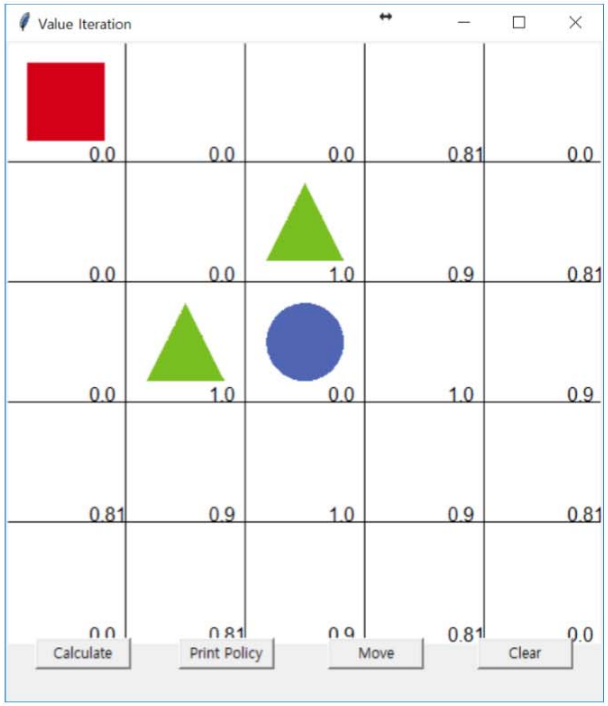
$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$



K=1



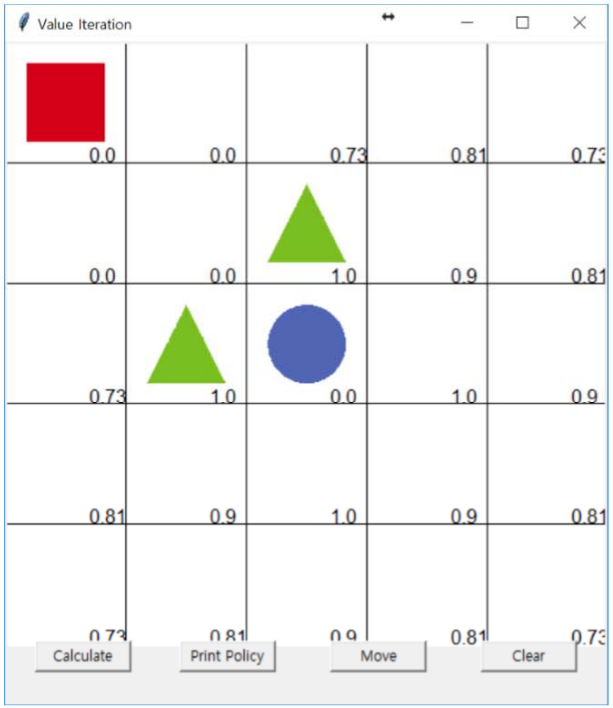
K=2



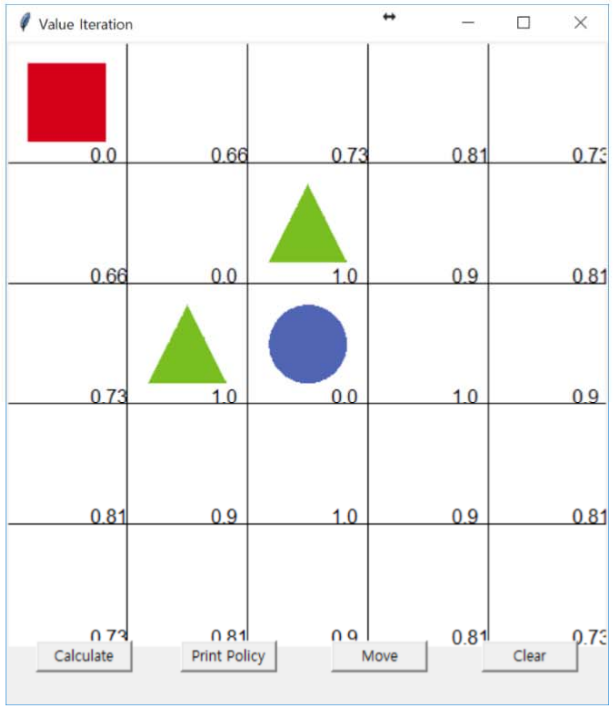
K=3

Example: Grid world에서 Value Iteration

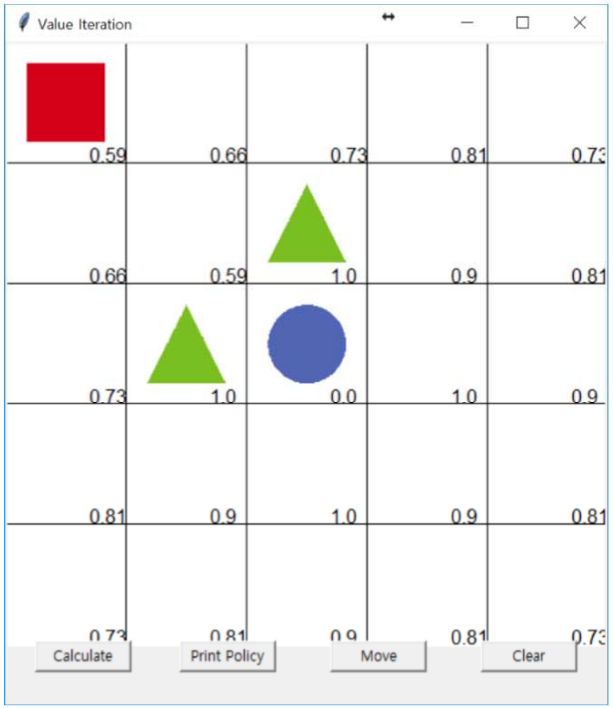
$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$



K=4



K=5

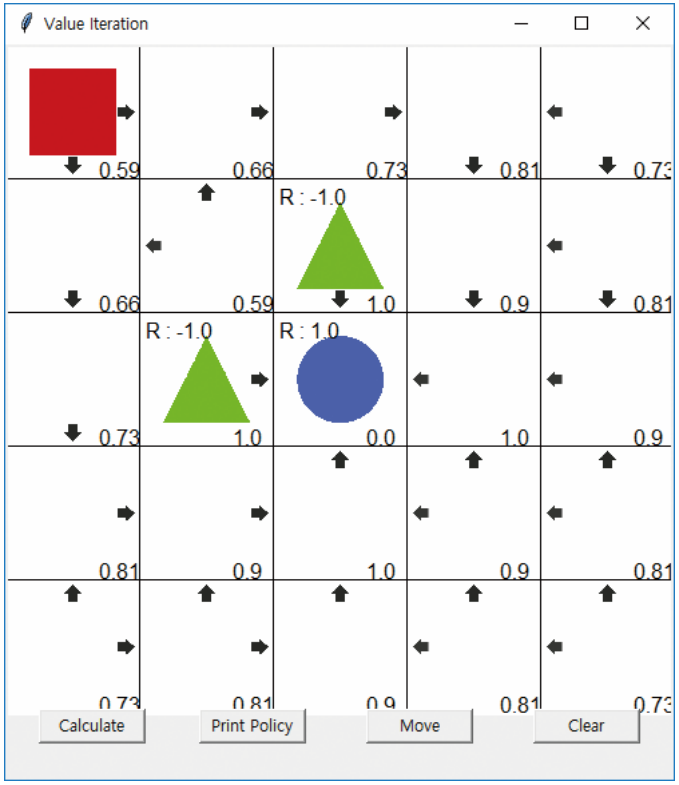


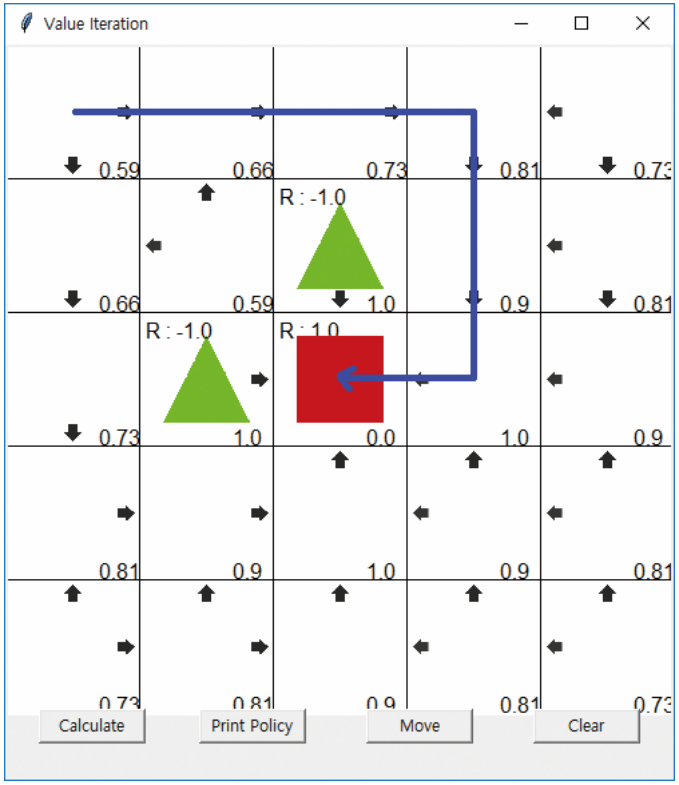
K=6

Example: Grid world에서 Value Iteration

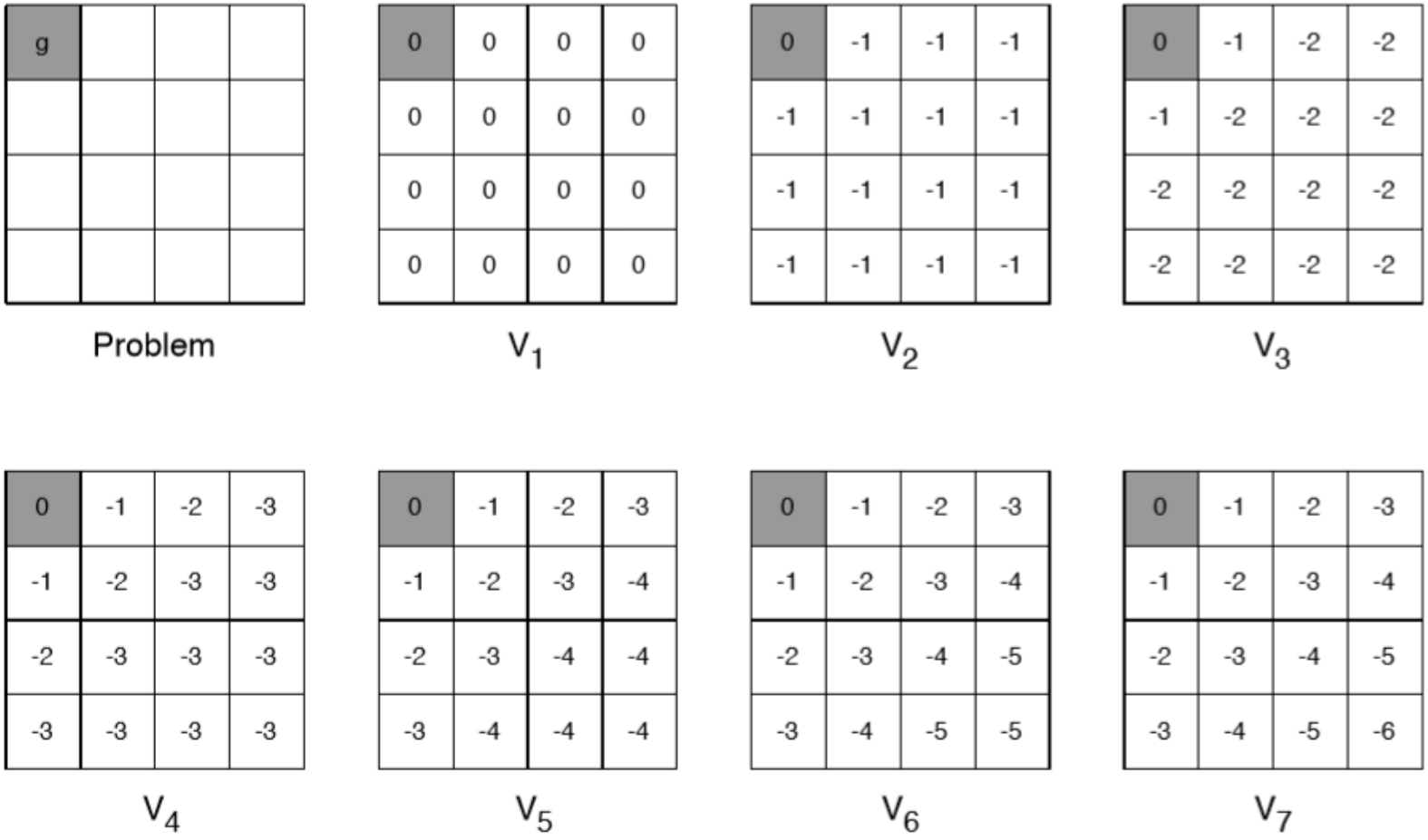
최적의 정책과 최적의 가치함수

$$\pi^*(s) = argmax_{a \in A} \boldsymbol{E}[R_s^a + \gamma v_k(s')]$$





Example: Shortest Path



David Silver, Reinforcement Learning, PPT

DP와 환경의 모델

- 벨만 최적 방정식을 통해 가치함수를 업데이트하려면 환경의 모델 (R_s^a 와 $P_{ss'}^a$)을 알아야함

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma \sum_{a \epsilon A} P_{ss'}^a v_k(s')]$$

-> $P_{ss'}^a$ 를 그 행동이 가려는 상태에 대해서 1, 나머지 0이라고 가정하면
(예, s'은 a가 right면 현재 상태에서 오른쪽에 있는 상태)

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

-> DP가 model-based 인 이유 -> learning이 아닌 Planning

- 모델을 정확히 알기 어려운 경우
-> Model-free -> Learning

Dynamic Programming Algorithms

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

David Silver, Reinforcement Learning, PPT

Summary

1. Dynamic Programming
   - 큰 문제를 작은 문제로, 반복되는 문제를 값을 저장하면서 해결
   - 큰 문제 : 최적 가치함수 계산 $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow \cdots \rightarrow v^*$
   - 작은 문제 : 현재의 가치함수를 더 좋은 가치함수로 업데이트 $v_k \rightarrow v_{k+1}$
   - Bellman Eq.를 이용해서 1-step 계산으로 optimal을 계산하기
2. Policy Iteration
   - 정책 평가(Policy Evaluation)와 정책 발전(Policy Improvement)을 반복함
     => 최적의 정책으로 수렴
   - 정책 평가(Policy Evaluation) - 벨만 기대방정식을 통한 가치함수 계산
   - 정책 발전(Policy Improvement) - Greedy Policy Improvement를 통한 최적의 정책 선택
3. Value Iteration
   - 가치함수가 최적이라고 가정하고 그 사이의 관계식인 벨만 최적 방정식 이용

$$v_{k+1}(s) \leftarrow \max_a [R_s^a + \gamma v_k(s')]$$

   - 수렴한 가치함수에 대해 greedy policy
   - Q-Learning으로 연결