

강화학습(Reinforcement Learning)



강화학습은 Reinforcement Learning이라고 함.

Reinforcement 강화, 증강이라는 뜻.

강화학습의 핵심은 일단 해보는 것.

지도학습

배움

강화학습

경험

비유하자면 지도학습이 배움을 통해서 실력을 키우는 것이라면,

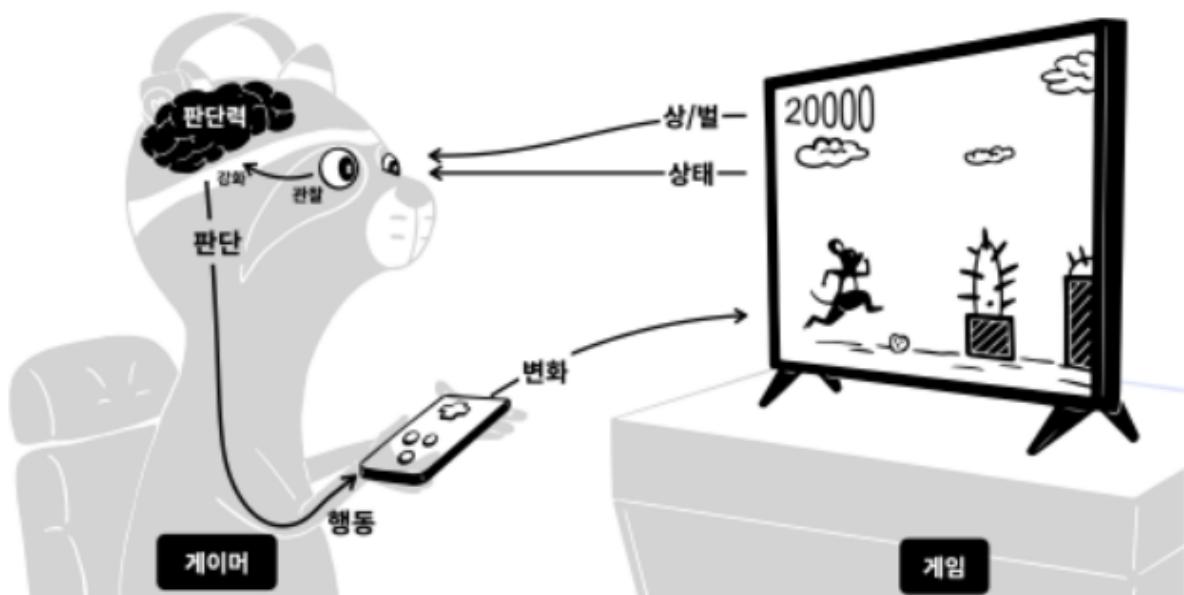
강화학습은 일단 해보면서 경험을 통해서 실력을 키워가는 것.

그 행동의 결과가 자신에게 유리한 것이었다면 상을 받고, 불리한 것이었다면 벌을 받게 됨.



이 과정을 매우 많이 반복하면 더 많은 보상을 받을 수 있는 더 좋은 답을 찾아낼 수 있다는 것이 강화학습의 기본 아이디어.

게임 실력을 키워가는 것도 강화학습이라고 볼 수 있음.



이런 상태에서 게임의 실력을 키워가는 과정을 따져보자.

1. 우선 게임은 게이머에게 현재의 상태를 보여줌. 캐릭터는 어디에 있고, 장애물은 어디에 있는지 알려줌.
2. 동시에 현재의 점수도 알려줌. 게이머는 이 값이 높아지는 것이 상이고, 장애물에 부딪히는 것이 벌.

3. 관찰의 결과에 따라서 어떤 상태에서 어떻게 행동해야 더 많은 상을 받고, 더 적은 벌을 받을 수 있는지를 알게 됨.
4. 즉, 판단력이 강화.
5. 판단에 따라서 행동.
6. 그 행동은 게임에 변화를 주게 됨.

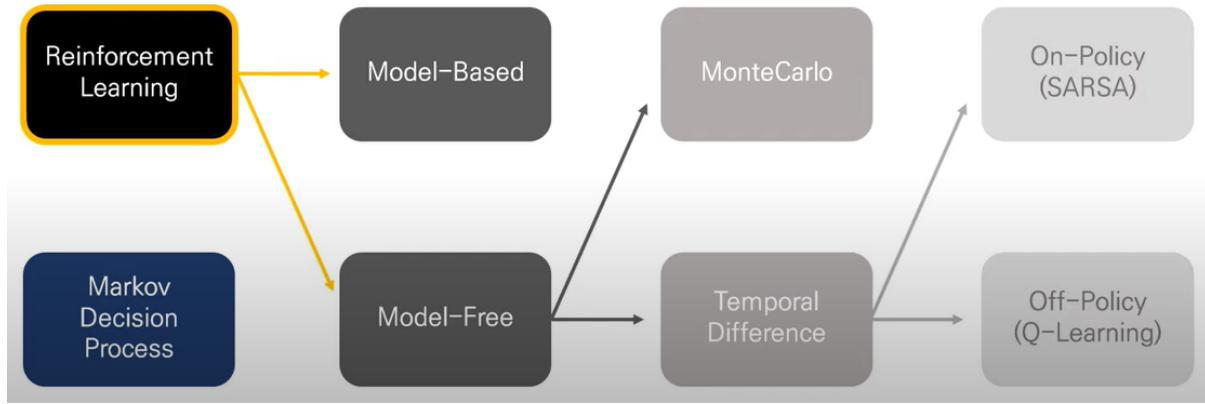
이런 과정을 반복하면 판단력이 점점 강화됨. 이것이 현실에서 게임의 실력자가 되는 과정임. 생각해보면 배우지 않고도 결국에 잘하게 되는 많은 일들이 이런 과정을 통해서 이루어짐. 강화학습은 이러한 과정을 모방해서 기계를 학습시키는 것. 이 과정을 강화학습에서 사용하는 용어로만 바꿔보자구



- 게임 → 환경(environment)
- 게이머 → 에이전트(agent)
- 게임화면 → 상태(state)
- 게이머의 조작 → 행동(action)
- 상과 벌 → 보상(reward)
- 게이머의 판단력 → 정책(policy)

강화학습에서는 **더 많은 보상을 받을 수 있는 정책**을 만드는 것이 핵심.

주어진 environment에서 agent가 더 높은 reward를 위해 최적의 action을 취해나가는 학습과정



강화학습 : 순차적인 의사결정 문제에서 누적 보상을 최대화하기 위해 시행착오를 거쳐서 상황에 따른 행동 정책을 학습

▼ 문제 범위 설정

❖ Markov Property

- 미래의 상태(s_{t+1})는 현재의 상태(s_t)에만 영향을 받으며 과거의 상태($s_1 \dots s_{t-1}$)의 영향을 받지 않음
- $\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, S_2, \dots, S_t]$



1. Markov Process(MP)

MP는 미래에 가능한 이벤트들의 순서를 표현하기 위한 모델. 즉 앞으로 발생할 수 있는 이벤트들에 대한 순서를 확률로 모델링하겠다는 것. 이러한 MP는 현재 발생한 이벤트는 이전의 이벤트에만 영향을 받는 가정인 Markov Property를 기반으로 함.

ex) 오늘 날씨가 맑은 건 아무래도 어제 날씨가 맑았기 때문임, 한달 전에 날씨가 영향을 미쳤을 리 없음

이러한 성질을 Memoryless property라고도 함.

$$P(S_{t+1} = s' | S_0, S_1, \dots, S_t) = P(S_{t+1} = s' | S_t) = P_{ss'}$$

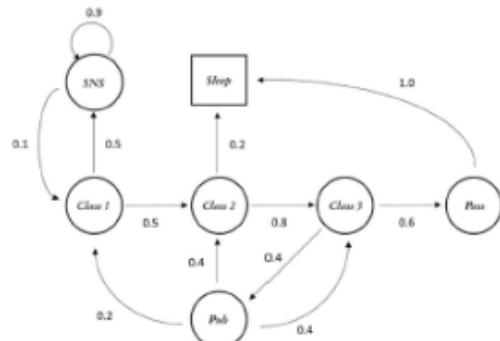
Markov Property

$\langle S, P_{tran} \rangle$

S: set of states

P_{tran} : state transition probability matrix

	Class 1	Class 2	Class 3	SNS	Bus	Pub	Sleep
Class 1	0.5	0.5	0.5	0.1	0.2	0.4	
Class 2		0.8	0.6	0.4	0.2	0.4	
Class 3			0.4	0.6	0.8	0.4	
SNS	0.1		0.9				
Pub	0.2	0.4	0.4				
Sleep				1			



MP로 모델링한 예시

2. Markov Reward Process(MRP)

- Reward r_t
 - Discounting factor $\gamma \in [0,1]$
 - Return: total discounted reward $G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
 - State-value function
- $$V(s) = \mathbb{E}[G_t | S_t = s]$$

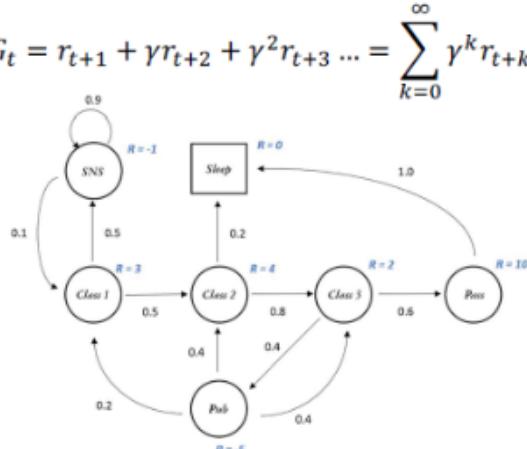
$\langle S, P_{tran}, R, \gamma \rangle$

S: set of states

P_{tran} : state transition probability matrix

R: Reward function

γ : Discounting factor



MRP에 대한 설명

$$\text{MRP} = \text{MP} + \text{Reward}$$

MRP는 사실상 MP에서 각 State마다 좋고 나쁨을 나타낼 수 있는 Reward가 추가된 개념.

Reward는 계속 쌓임. 앞으로 받을 모든 Reward를 계산한게 Return이 됨.

여기서 미래의 받을 보상과 지금 받을 보상의 가치를 다르게 평가하기 위해서 Discounting Factor라는 개념 필요.

ex) 지금 10000원 받을래 아니면 1년 뒤에 10000원 받을래? 중 뭘 선택하겠니. 닥전.

지금의 Reward와 미래의 Reward는 다름

이렇게 현재 State에서 Return의 기댓값을 정의한 게 바로 **State-Value function**

3. Markov Decision Process(MDP)

- Action A_t



- Policy $\pi(a|s) = P(A_t = a|S_t = s)$

- Action-value function $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

$\langle S, A, P_{tran}, R, \gamma \rangle$

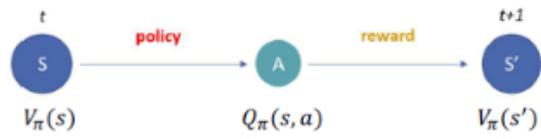
S: set of states

A: set of actions

P_{tran} : state transition probability matrix

R: Reward function

γ : Discounting factor



MDP에 대한 설명

$$MDP = MRP + Decision = MP + Reward + Decision$$

MDP는 MRP에서 Decision(Action)이 추가된 개념. 강화학습에서는 이 MDP를 통해 모델링.

MP와 MRP에서는 현재 State에서 다음 State로 바로 이어졌지만, MDP에서는 다음 State로 넘어갈 때, 어떤 Action을 수행하느냐에 따라서 바뀜. 즉 어떤 Action을 하느냐는 확률에 따라서 결정됨.

현재 State에서 어떤 Action을 수행할 것인지를 Policy라고 함.

앞선 State-value function은 현재 State에 따라서 받을 수 있는 Return의 기댓값이었다면,

Action-value function은 현재 State에서 특정 Action에 따라서 받을 수 있는 Return의 기댓값이라고 볼 수 있음. 여기서의 핵심은 Action.

▼ Bellman Equation

앞서 봤던 가치함수들을 재귀적으로 표현해서 각 시점의 상태, 상태-행동 가치 계산할 수 있게 함

벨만 방정식

- 현재 시점의 가치와 다음 시점의 가치 사이의 관계를 다루는 방정식
- 벨만 방정식을 통해서 상태 및 행동의 가치를 책정할 수 있음

벨만 방정식의 두 종류

- 벨만 기대방정식(Bellman Expectation equation) : 주어진 정책을 평가할 때 사용
- 벨만 최적방정식(Bellman Optimality equation) : 최적의 상태 가치를 찾을 때 사용

▼ 벨만 기대방정식

❖ Bellman Expectation Equation

- 벨만 기대방정식 (bellman expectation equation)
 - ✓ 정책이 고정되어 있는 상황에서 상태 또는 상태-행동의 가치를 책정하는 방법으로서 주어진 정책에 대해 평가
- 보상함수와 상태전이확률을 알고 있는지 여부에 따라 표현하는 방법이나됨
 - ** 보상함수와 상태전이확률을 있다고 할 때 MDP를 있다고 표현함

	MDP를 모르는 경우	MDP를 아는 경우
State Value	$v_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]$	$v_\pi(s) = \sum_{a \in A} \pi(a s) \left(\mathbf{r}_s^a + \gamma \sum_{s' \in S} \mathbf{P}_{ss'}^a v_\pi(s') \right)$
State-Action Value	$q_\pi(s_t, a_t) = \mathbb{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})]$	$q_\pi(s, a) = \mathbf{r}_s^a + \gamma \sum_{s' \in S} \mathbf{P}_{ss'}^a \sum_{a' \in A} \pi(a' s') q_\pi(s', a')$

1. MDP를 모르는 경우 + State Value

- 주어진 환경에서 샘플링 된 에피소드의 리턴 값에 평균을 취하여 해당 상태의 가치 평가

$$\begin{aligned}
 v_\pi(s_t) &= \mathbb{E}_\pi[G_t] \\
 &= \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots] \\
 &= \mathbb{E}_\pi[r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \dots)] \\
 &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1}] \\
 &= \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]
 \end{aligned}$$

2. MDP를 아는 경우 + State Value

$$v_{\pi}(s) = \sum_{a \in A} \pi(a|s) \left(r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \right)$$

현재 상태에서 특정 행동을 선택할 확률
 ↑
 현재 상태에서 특정 행동을 통해 얻는 보상
 ↑
 현재 상태에서 특정 행동을 하였을 때 특정 다음 상태로 넘어갈 확률
 ↑
 특정 다음 상태의 가치

MDP를 모르는 상황에서 상태가치함수에 상태전이확률 및 보상함수를 고려할 경우
MDP를 아는 상황의 상태가치함수로 풀어쓸 수 있음

$$\begin{aligned}
 v_{\pi}(s) &= \sum_{a \in A} \pi(a|s) \left(r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_{\pi}(s') \right) \\
 &= \sum_{a \in A} \pi(a|s) r_s^a + \gamma \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P_{ss'}^a v_{\pi}(s')
 \end{aligned}$$

현재 상태에서 특정 행동을 선택할 확률 X 현재 상태에서 특정 행동을 통해 얻는 보상
 ↑
 현재 상태에서 특정 행동을 하였을 때 특정 다음 상태로 넘어갈 확률 X 특정 다음 상태의 가치
 ↑
 현재 상태에서 특정 행동을 할 확률 X 특정 행동을 할 때 다음 상태의 평균 가치

$$\begin{aligned}
 &= \mathbb{E}_{\pi}[r_{t+1}] + \mathbb{E}_{\pi}[\gamma v_{\pi}(s_{t+1})] \\
 &= \mathbb{E}_{\pi}[r_{t+1} + \gamma v_{\pi}(s_{t+1})]
 \end{aligned}$$

3. MDP 모르는 경우 + State-Action Value Function(Q)

- 주어진 환경에서 샘플링 된 에피소드의 리턴 값에 평균을 취하여 해당 상태-행동의 가치 평가

$$v_{\pi}(s_t) = \mathbb{E}_{\pi}[r_{t+1} + \gamma v_{\pi}(s_{t+1})] \quad \rightarrow \quad q_{\pi}(s_t, a_t) = \mathbb{E}_{\pi}[r_{t+1} + \gamma q_{\pi}(s_{t+1}, a_{t+1})]$$

4. MDP 아는 경우 + State-Action Value Function(Q)

$$q_{\pi}(s, a) = \mathbf{r}_s^a + \gamma \sum_{s' \in S} \mathbf{P}_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_{\pi}(s', a')$$

현재 상태에서 특정 행동을 하였을 때 특정 다음 상태로 넘어갈 확률 X 특정 다음 상태의 가치

$$= \mathbb{E}_{\pi}[r_{t+1}] + \mathbb{E}_{\pi}[\gamma q_{\pi}(s_{t+1}, a_{t+1})]$$

$$= \mathbb{E}_{\pi}[r_{t+1} + \gamma q_{\pi}(s_{t+1}, a_{t+1})]$$

▼ 벨만 최적방정식

❖ Bellman Optimality Equation

- 벨만 최적방정식 (bellman optimality equation)
 - ✓ 주어진 MDP에서 가장 좋은 정책(최적정책)을 선택하여 최적의 상태 혹은 상태-행동 가치를 평가함
- 보상함수와 상태전이확률을 알고 있는지 여부에 따라 표현하는 방법이나됨
 - 〃 보상함수와 상태전이확률을 있다고 할 때 MDP를 있다고 표현함

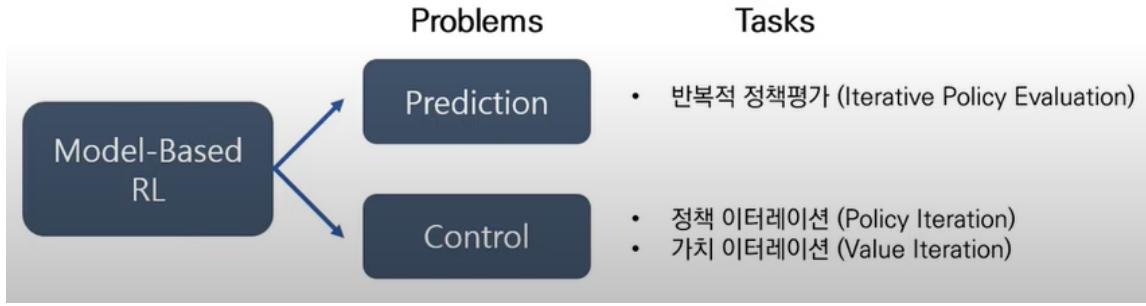
	MDP를 모르는 경우	MDP를 아는 경우
State Value	$v_*(s_t) = \max_a \mathbb{E} [r_{t+1} + \gamma v_*(s_{t+1})]$	$v_*(s) = \max_a \left[\mathbf{r}_s^a + \gamma \sum_{s' \in S} \mathbf{P}_{ss'}^a v_*(s') \right]$
Action-State Value	$q_*(s_t, a_t) = \mathbb{E} \left[r_{t+1} + \gamma \max_{a'} q_*(s_{t+1}, a') \right]$	$q_*(s, a) = \mathbf{r}_s^a + \gamma \sum_{s' \in S} \mathbf{P}_{ss'}^a \max_{a'} q_*(s', a')$

- 벨만 최적방정식은 행동을 선택할 때 확률적으로 선택하는 것이 아니라 최댓값 연산자를 통해 제일 좋은 액션을 선택함
- 따라서 확률적 요소 중 상태전이확률은 유지되지만 행동 선택 확률은 사용되지 않음

▼ Model-Based Reinforcement Learning

MDP에 대한 모든 정보를 알 때 이를 이용하여 정책을 평가 및 개선해 나가는 과정

- Prediction problem : 정책이 주어졌을 때 각 상태의 가치를 평가하는 문제
- Control problem : 최적의 정책 함수를 찾는 문제



Example

❖ Example Environment) Grid World 1

- Action : Left(25%), Right(25%), Up(25%), Down(25%)
- Reward : -1 per step
- State : Total 16 states including terminal states
- Decay Ratio : 1
- State Transition Probability : 1

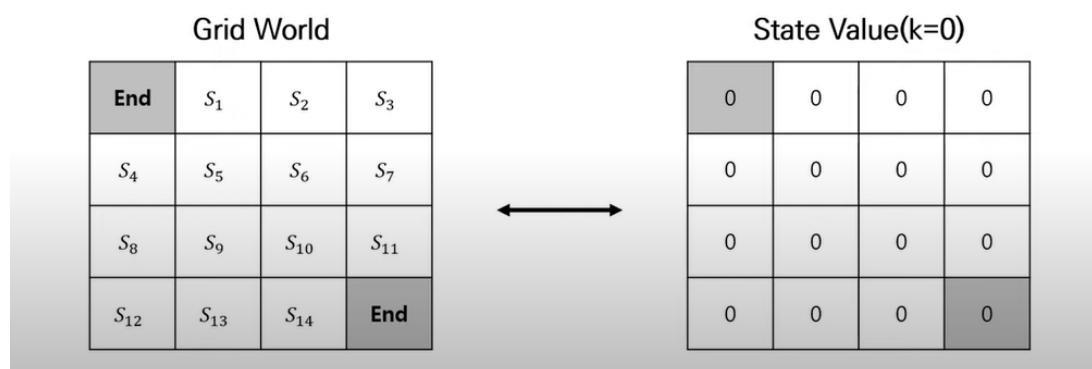
Grid World

End	S_1	S_2	S_3
S_4	S_5	S_6	S_7
S_8	S_9	S_{10}	S_{11}
S_{12}	S_{13}	S_{14}	End

▼ Prediction - 반복적 정책평가(Iterative Policy Evaluation)

Step 1. 상태 가치 초기화

- ✓ 상태 가치의 초기값을 0으로 설정함



Step 2. 상태 가치 업데이트 (반복)

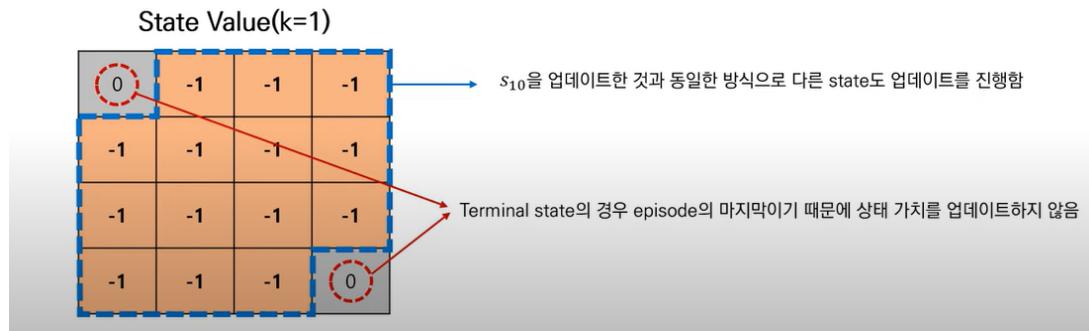
- ✓ 벨만기대방정식을 사용하여 상태의 가치를 업데이트

State Value(k=1)			
0	0	0	0
0	0	s_6 0	0
0	s_9 0	s_{10} -1	s_{11} 0
0	0	s_{14} 0	0

$$\begin{aligned}
 v_\pi(s_{10}) &= \sum_{a \in A} \pi(a|s_{10}) \left(r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right) \\
 &= 0.25 * (-1 + 1 * (1 * 0 + 0 * 0 + 0 * 0 + 0 * 0)) \quad \# \text{Left} \\
 &\quad + 0.25 * (-1 + 1 * (0 * 0 + 1 * 0 + 0 * 0 + 0 * 0)) \quad \# \text{Right} \\
 &\quad + 0.25 * (-1 + 1 * (0 * 0 + 0 * 0 + 1 * 0 + 0 * 0)) \quad \# \text{Up} \\
 &\quad + 0.25 * (-1 + 1 * (0 * 0 + 0 * 0 + 0 * 0 + 1 * 0)) \quad \# \text{Down} \\
 &= -1.0
 \end{aligned}$$

Step 2. 상태 가치 업데이트 (반복)

- ✓ 벨만기대방정식을 사용하여 상태의 가치를 업데이트



Step 2. 상태 가치 업데이트 (반복)

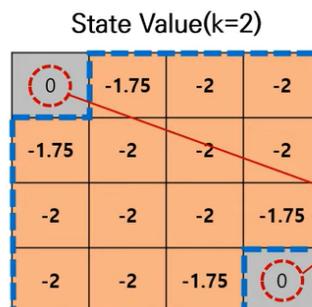
- ✓ 벨만기대방정식을 사용하여 상태의 가치를 업데이트

State Value(k=2)			
0	-1	-1	-1
-1	-1	s_6 -1	-1
-1	s_9 -1	s_{10} -2	s_{11} -1
-1	-1	s_{14} -1	0

$$\begin{aligned}
 v_\pi(s_{10}) &= \sum_{a \in A} \pi(a|s_{10}) \left(r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right) \\
 &= 0.25 * (-1 + 1 * (1 * -1 + 0 * -1 + 0 * -1 + 0 * -1)) \\
 &\quad + 0.25 * (-1 + 1 * (0 * -1 + 1 * -1 + 0 * -1 + 0 * -1)) \\
 &\quad + 0.25 * (-1 + 1 * (0 * -1 + 0 * -1 + 1 * -1 + 0 * -1)) \\
 &\quad + 0.25 * (-1 + 1 * (0 * -1 + 0 * -1 + 0 * -1 + 1 * -1)) \\
 &= -2.0
 \end{aligned}$$

Step 2. 상태 가치 업데이트 (반복)

- ✓ 벨만기대방정식을 사용하여 상태의 가치를 업데이트



Step 2. 상태 가치 업데이트 (반복)

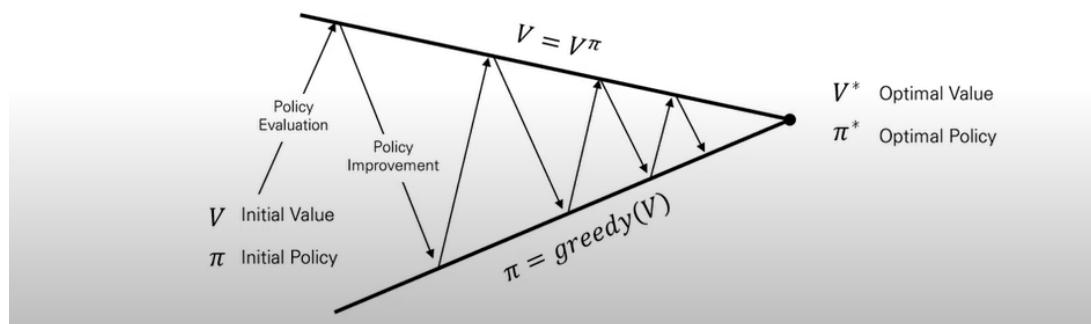
- ✓ 벨만기대방정식을 사용하여 상태의 가치를 업데이트

State Value(k=2)				State Value(k=3)				State Value(k=∞)			
0	-1.75	-2	-2	0	-2.4	-2.9	-3	0	-14	-20	-22
-1.75	-2	-2	-2	-2.4	-2.9	-3	-2.9	-14	-18	-20	-20
-2	-2	-2	-1.75	-2.9	-3	-2.9	-2.4	-20	-20	-18	-14
-2	-2	-1.75	0	-3	-2.9	-2.4	0	-22	-20	-14	0

▼ Control - 정책 이터레이션(Policy Iteration)

❖ Policy Iteration

- 정책 이터레이션 (Policy Iteration)
 - ✓ 정책 평가와 정책 개선을 번갈아 수행하여 정책이 수렴할 때까지 반복하는 방법론
- 정책 평가: 반복적 정책 평가
- 정책 개선: 그리디 정책 개선



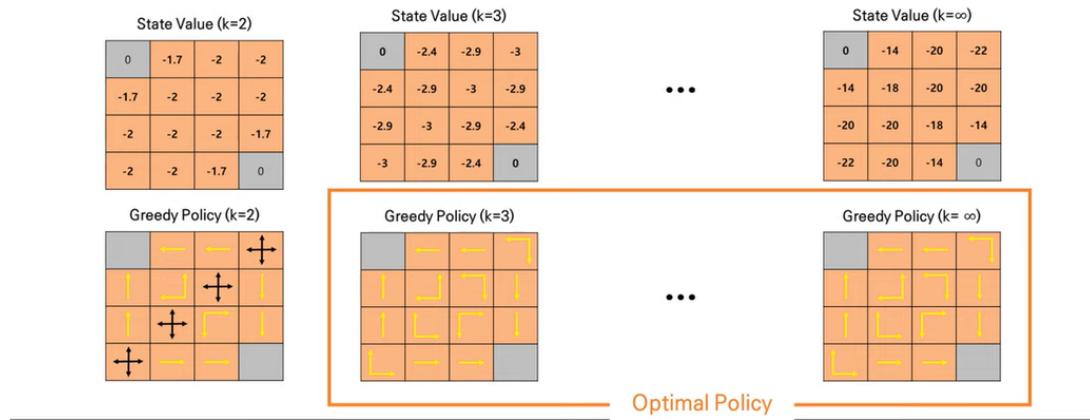
❖ Greedy Policy Improvement

- 현재 상태에서 갈 수 있는 다음 상태 중 가치가 가장 높은 쪽으로만 행동을 취하도록 하는 정책

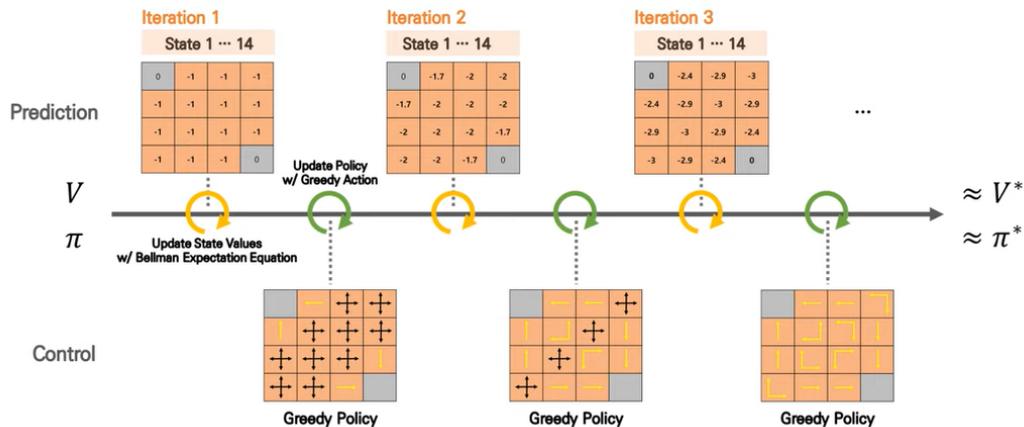
State Value (k=1)				Random Policy (k=0)				Greedy Policy (k=1)			
0	-1	-1	-1	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓
-1	-1	-1	-1	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓
-1	-1	-1	-1	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓
-1	-1	-1	0	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓	↑↓

❖ How many iterations are needed to finish the policy evaluation?

- 매 정책 평가 때마다 상태 가치가 수렴할 필요는 없으며, 정책 선언이 발생할 수 있을 만큼 루프를 진행해도 충분함
- 최적 정책을 찾는 것이 정책 이터레이션의 목적이기 때문에 정책이 수렴한 경우에는 업데이트 진행 필요 없음



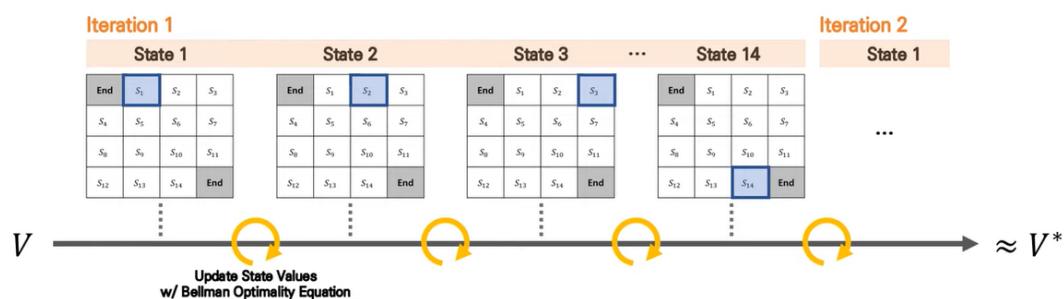
❖ Overall Process Visualization



▼ Control - 가치 이터레이션(Value Iteration)

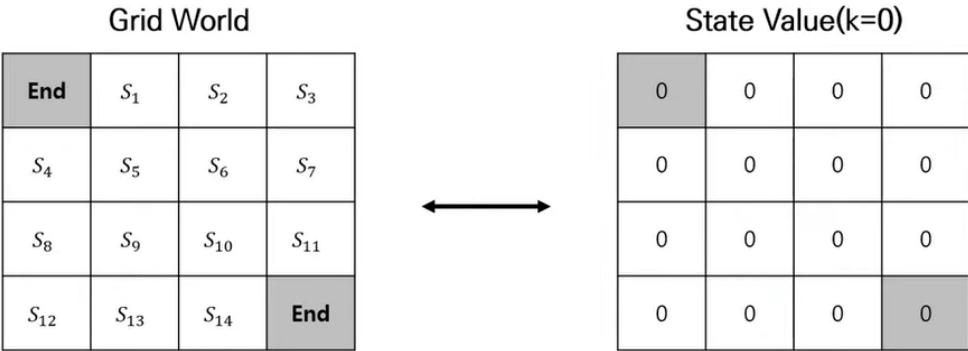
❖ Value Iteration

- 가치 이터레이션 (Value Iteration)
 - 환경에 의존적인 정책 평가를 진행하여 최적 가치를 구함으로써 최적 정책을 도출하는 방법론
- 벨만최적방정식을 사용하여 상태가치를 반복적으로 업데이트 함 $\rightarrow v_*(s) = \max_a [r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')]$



Step 1. 상태 가치 초기화

- 상태 가치의 초기값을 0으로 설정함



Step 2. 상태 가치 업데이트 (반복)

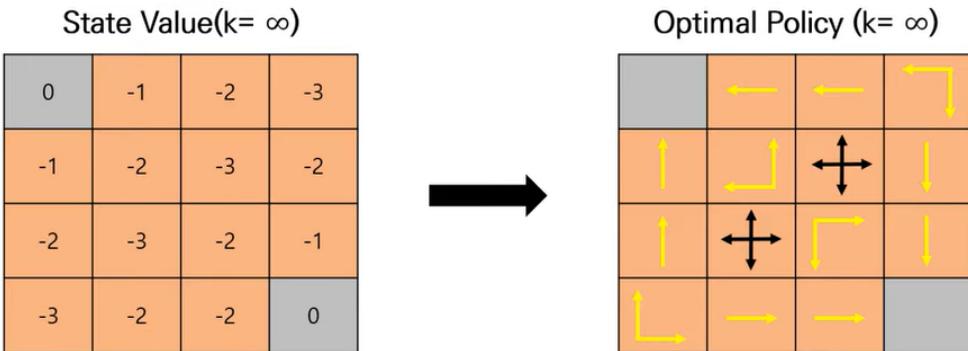
- 벨만최적방정식을 사용하여 상태의 가치를 업데이트

State Value(k=1)			
0	-1	-1	-1
-1	-1	S_6 -1	-1
-1	S_9 -1	S_{10} -1	S_{11} -1
-1	-1	S_{14} -1	0

$$\begin{aligned}
 v_*(s_{10}) &= \max_a \left[r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \right] \\
 &= \max(-1 + 1 * (1 * 0 + 0 * 0 + 0 * 0 + 0 * 0) \quad \# \text{Left} \\
 &\quad -1 + 1 * (0 * 0 + 1 * 0 + 0 * 0 + 0 * 0) \quad \# \text{Right} \\
 &\quad -1 + 1 * (0 * 0 + 0 * 0 + 1 * 0 + 0 * 0) \quad \# \text{Up} \\
 &\quad -1 + 1 * (0 * 0 + 0 * 0 + 0 * 0 + 1 * 0) \quad \# \text{Down}) \\
 &= -1.0
 \end{aligned}$$

Step 2. 상태 가치 업데이트 (반복)

- 벨만최적방정식을 사용하여 상태의 가치를 업데이트
- 업데이트를 반복하여 최적 가치를 함으로써 최적 정책을 가짐



Summary

Prediction : 정책이 주어졌을 때 각 상태의 가치를 평가하는 문제

1. 반복적 정책 평가

- 벨만기대방정식 사용
- 주어진 정책 고정되어 있음

- 주어진 정책으로 각 상태 가치 평가

Control : 최적의 정책 함수를 찾는 문제

1. 정책 이터레이션

- 정책 평가와 정책 개선이 번갈아 가며 시행
- 정책 평가는 prediction 문제로 접근하며 마찬가지로 반복적 정책 평가 사용(**벨만 기대방정식**)
- 정책 개선은 정책 평가를 통해 업데이트 된 상태가치에 따라서 그리디 행동을 취하도록 정책 변경

2. 가치 이터레이션

- Prediction 문제를 수행할 때 **벨만최적방정식**을 사용
- 벨만최적방정식에서 이미 그리디 행동을 취하기 때문에 최적 가치를 구함으로써 최적 정책함수를 가지게 됨

▼ Model-Free Reinforcement Learning

MDP를 알 수 없을 때 정책을 평가 및 개선해 나가는 과정



Example

❖ Example Environment) Grid World 2

- Action : Left(25%), Right(25%), Up(25%), Down(25%)
- Reward : $\text{?} = -1 \text{ per step}$
- State : Total 16 states including terminal states
- Decay Ratio : 1
- State Transition Probability : $\text{?} \downarrow$

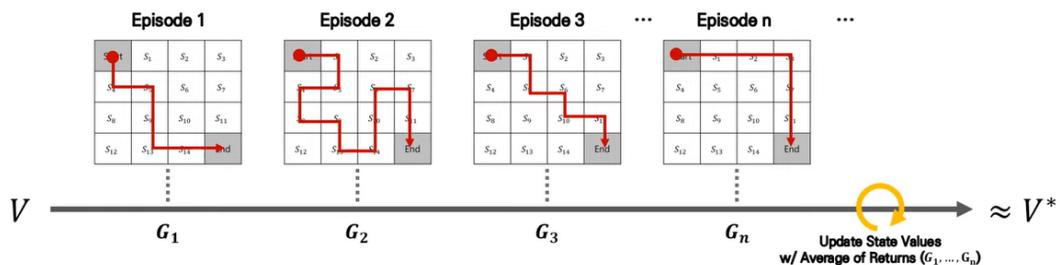
Grid World

Start	S_1	S_2	S_3
S_4	S_5	S_6	S_7
S_8	S_9	S_{10}	S_{11}
S_{12}	S_{13}	S_{14}	End

▼ Prediction - Monte-Carlo(MC)

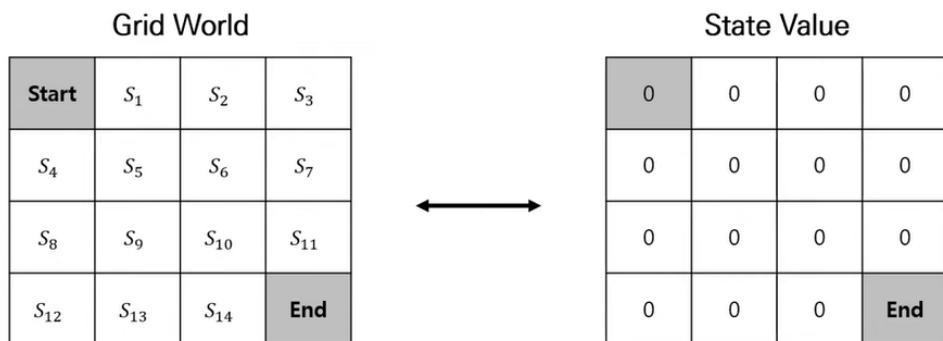
❖ Monte-Carlo(MC)

- Monte-Carlo
 - ✓ 반복된 무작위 추출을 이용하여 함수의 값을 수리적으로 근사하는 알고리즘
 - ✓ 주어진 정책을 반복적으로 실행하여 얻은 리턴의 값으로 상태 가치를 근사하는 알고리즘, 이때 대수의 법칙에 의해 예측치가 점점 정확해짐
- 종료된 에피소드에서만 학습할 수 있음, 따라서 Non-terminating Episode에서는 사용할 수 없음
- 각 상태의 가치를 평가할 때 Return 값을 사용함 $V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$



Step 1. 상태 가치 초기화

- ✓ 상태 가치의 초기값을 0으로 설정함



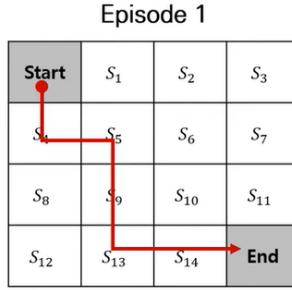
❖ Monte-Carlo(MC)

Step 2. 상태 가치 업데이트 (반복)

- ✓ 획득한 리턴의 값에 따라 상태의 가치를 업데이트 (수식 참고)

MC / update equation

$$V(s_t) \leftarrow V(s_t) + \alpha * (G_t - V(s_t))$$



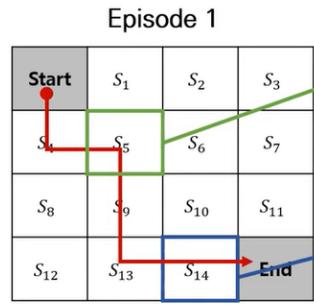
총 6 step으로 구성된 episode

Time Step(t)	State(s _t)	Action(a _t)	Reward(r _t)	Next State(s _{t+1})	Return(G _t)
1	Start	Down	-1	State 4	-6
2	State 1	Right	-1	State 5	-5
3	State 5	Down	-1	State 9	-4
4	State 9	Down	-1	State 13	-3
5	State 13	Right	-1	State 14	-2
6	State 14	Right	-1	End	-1

$$\begin{aligned} G_1 &= r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \gamma^4 r_5 + \gamma^5 r_6 \\ &= -1 + 1 * -1 + 1 * -1 + 1 * -1 + 1 * -1 + 1 * -1 \\ &= -6 \quad \# \text{Return of Start State} \end{aligned}$$

Step 2. 상태 가치 업데이트 (반복)

- ✓ 획득한 리턴의 값에 따라 상태의 가치를 업데이트 (수식 참고)



-4 리턴(G) : -4

$$V(s_3) \leftarrow V(s_3) + \alpha(G_3 - V(s_3)) = 0 + 0.001(-4 - 0) = -0.004$$

** α 가 0.001 일 때

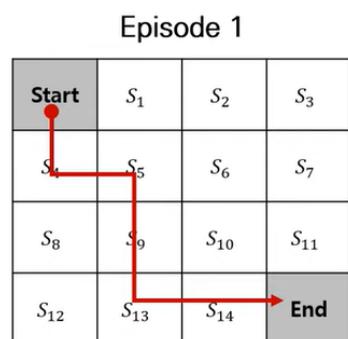
-1 리턴(G) : -1

$$V(s_6) \leftarrow V(s_6) + \alpha(G_6 - V(s_6)) = 0 + 0.001(-1 - 0) = -0.001$$

** α 가 0.001 일 때

Step 2. 상태 가치 업데이트 (반복)

- ✓ 획득한 리턴의 값에 따라 상태의 가치를 업데이트 (수식 참고)



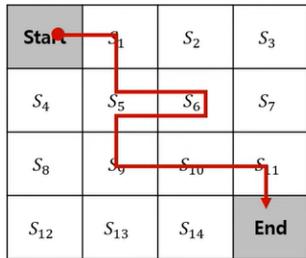
State Value

-0.006	0	0	0
-0.005	-0.004	0	0
0	-0.003	0	0
0	-0.002	-0.001	End

Step 2. 상태 가치 업데이트 (반복)

- 만일 같은 상태를 두 번 이상 방문하게 될 경우, 리턴 값을 계산하는 방법은 사용자가 지정함
ex) First-Visit / Every-Visit / Incremental

Episode 2



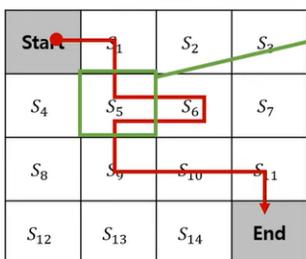
총 8 step으로 구성된 episode

Time Step(t)	State(s_t)	Action(a_t)	Reward(r_t)	Next State(s_{t+1})	Return(G_t)
1	Start	Right	-1	State 1	-8
2	State 1	Down	-1	State 5	-7
3	State 5	Right	-1	State 6	-6
4	State 6	Left	-1	State 5	-5
5	State 5	Down	-1	State 9	-4
6	State 9	Right	-1	State 10	-3
7	State 10	Right	-1	State 11	-2
8	State 11	Down	-1	End	-1

Step 2. 상태 가치 업데이트 (반복)

- 획득한 리턴의 값에 따라 상태의 가치를 업데이트 (수식 참고)

Episode 2



-4, -6 리턴(G) : -4, -6

$$V(s_5) \leftarrow V(s_5) + \alpha(G_5 - V(s_5)) \\ = -0.004 + 0.001(-4 + 0.004) = -0.008$$

** α 가 0.001 일 때

$$V(s_3) \leftarrow V(s_3) + \alpha(G_3 - V(s_3)) \\ = -0.008 + 0.001(-6 + 0.008) = -0.014$$

** α 가 0.001 일 때

Step 2. 상태 가치 업데이트 (반복)

- 획득한 리턴의 값에 따라 상태의 가치를 업데이트 (수식 참고)

State Value(Episode 1)

-0.006	0	0	0
-0.005	-0.004	0	0
0	-0.003	0	0
0	-0.002	-0.001	End

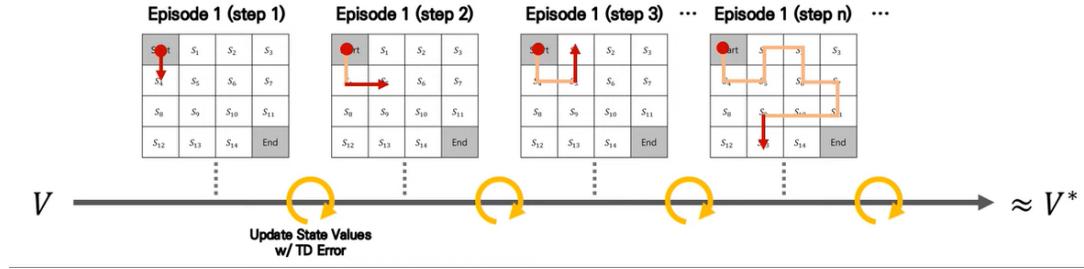
State Value(Episode 2)

-0.006	-0.007	0	0
-0.005	-0.014	-0.005	0
0	-0.006	-0.002	-0.001
0	-0.002	-0.001	End

▼ Prediction - Temporal Difference(TD)

❖ Temporal Difference(TD)

- Temporal Difference
 - ✓ 미래의 추측으로 현재의 추측을 업데이트하는 방법론
 - ✓ 조금이라도 시간이 흐르면 좀 더 정확한 추측을 할 수 있다고 가정하여 미래의 추측을 일종의 정답지로 활용하는 것
- 에피소드가 진행되는 중간에도 학습할 수 있음, 따라서 Non-terminating Episode에서도 사용할 수 있음
- 각 상태의 가치를 평가할 때 TD error 값을 사용함



$$V(s_t) \leftarrow V(s_t) + \alpha * (r_{t+1} + \gamma V(s_{t+1}) - V(s_t))$$

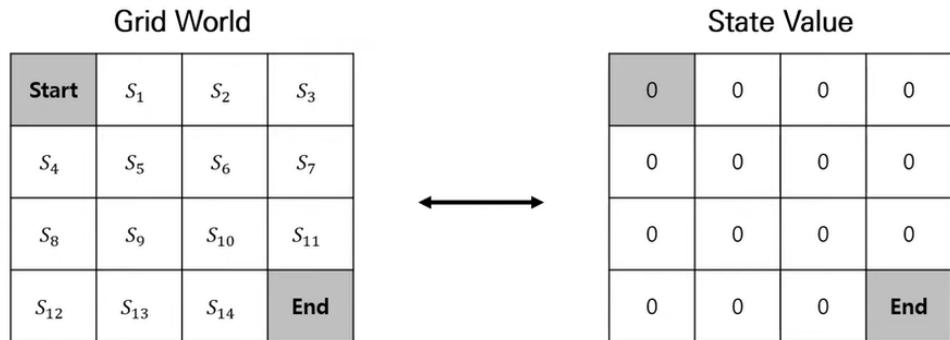
TD Target

TD Error

(실제 보상 + 다음 상태 가치) - 현재 상태 가치

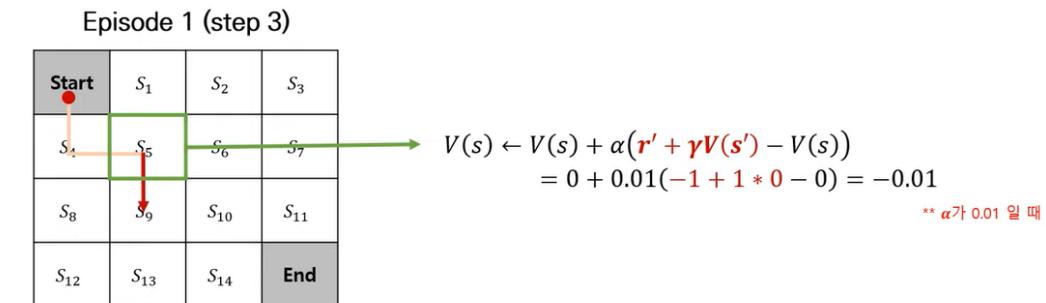
Step 1. 상태 가치 초기화

- ✓ 상태 가치의 초기값을 0으로 설정함



Step 2. 상태 가치 업데이트 (반복)

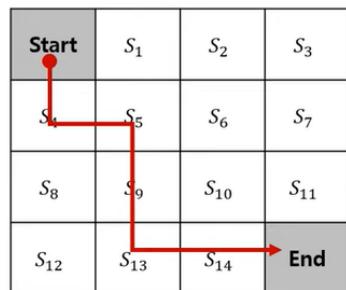
- ✓ 획득한 TD error에 따라 상태의 가치를 업데이트 (수식 참고)



Step 2. 상태 가치 업데이트 (반복)

✓ 획득한 TD error에 따라 상태의 가치를 업데이트 (수식 참고)

Episode 1 (steps 1 ~ 6)



State Value

-0.01	0	0	0
-0.01	-0.01	0	0
0	-0.01	0	0
0	-0.01	-0.01	End



▼ MC, TD 방법론 간의 비교

1. 학습시점

- MC 방법론은 에피소드 단위로 상태 가치를 업데이트함. 에피소드가 **종료되어야 학습 가능**
- TD 방법론은 스텝 단위로 상태 가치를 업데이트함. 에피소드가 **종료되지 않아도 학습 가능**
- 따라서 학습 시점 측면에서는 **TD 방법론이 더 효과적**

2. 편향성

- MC 방법론은 여러 번의 업데이트로 얻은 **리턴의 평균은 실제 가치에 수렴하게 됨**
- TD 방법론은 **실제 가치 함수를 알 수 없어서** 주어진 상태의 가치를 사용하기 때문에 **수렴이 보장 안됨**
- 따라서 학습 편향성 측면에서는 **MC 방법론이 더 효과적**

Updating Equation

MC method

$$V(s_t) \leftarrow V(s_t) + \alpha * (G_t - V(s_t))$$



$$v_\pi(s_t) = \mathbb{E}[G_t]$$

가치 함수의 정의

TD method

$$V(s_t) \leftarrow V(s_t) + \alpha * (r_t + \gamma V(s_{t+1}) - V(s_t))$$



$$v_\pi(s_t) = \mathbb{E}[r_t + \gamma v_\pi(s_{t+1})]$$

벨만 기대 방정식

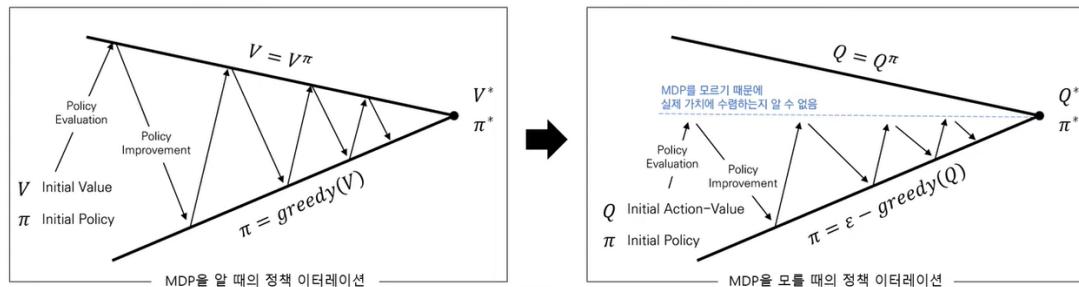
3. 분산

- 업데이트 주기가 유동적이므로 학습의 분산이 큼
- 업데이트 주기가 고정되어 있으므로 학습의 분산이 작음
- 따라서 학습의 분산 측면에서는 TD 방법론이 더 효과적

▼ Control - 개요

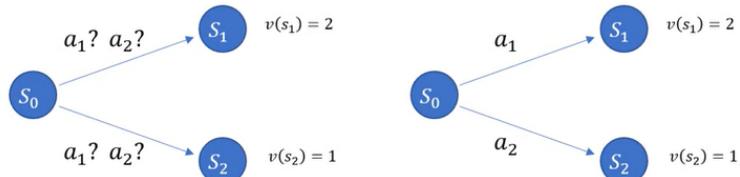
❖ Improving Policy in Model-Free RL

- MDP를 모르는 상황에서 기존의 빈ぼ적 정책 평가 방법론은 사용할 수 없음
→ MDP를 모르는 상황에서 상태 또는 상태-행동의 가치를 업데이트하는 방법론인 Monte-Carlo와 Temporal Difference를 정책 평가에 사용함
- 상태 가치만으로는 상태 전이 확률을 알 수 없기 때문에 어떤 행동으로 어떤 상태에 도달하는지 알 수 없기 때문에 정책 개선을 할 수 없음
→ 상태-행동 가치를 통해서 더 높은 상태-행동 가치를 가진 행동을 선택하도록 정책 개선을 함



❖ Greedy Action in Model Free Control

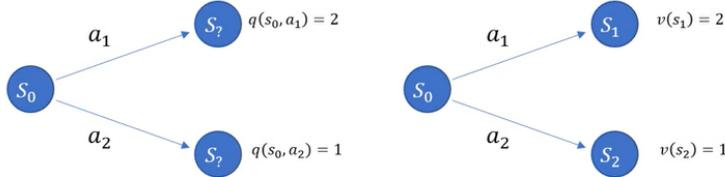
- MDP를 모르는 상황에서는 상태 전이 확률을 모르기 때문에 각 행동을 선택했을 때 다음 상태가 어디가 될지 알 수 없음
- 행동에 따라 도착하는 상태를 비교할 수 없기 때문에 정책을 개선할 수 없음



	MDP를 모르는 경우	MDP를 아는 경우
State Value	$v_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]$	$v_\pi(s) = \sum_{a \in A} \pi(a s) \left(r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$
State-Action Value	$q_\pi(s_t, a_t) = \mathbb{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})]$	$q_\pi(s, a) = r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' s') q_\pi(s', a')$

❖ Greedy Action in Model Free Control

- 가치 함수 대신에 행동-가치 함수를 쓸 경우 MDP 정보를 몰라도 그리디 행동을 취할 수 있음
- 상태전이 확률을 모르기 때문에 선택한 액션이 어느 상태로 데려가는지는 모르지만 가장 높은 가치에 도달하는 것은 알 수 있음

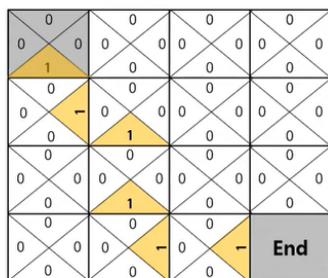


	MDP를 모르는 경우	MDP를 아는 경우
State Value	$v_\pi(s_t) = \mathbb{E}_\pi[r_{t+1} + \gamma v_\pi(s_{t+1})]$	$v_\pi(s) = \sum_{a \in A} \pi(a s) \left(r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$
State-Action Value	$q_\pi(s_t, a_t) = \mathbb{E}_\pi[r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})]$	$q_\pi(s, a) = r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a' s') q_\pi(s', a')$

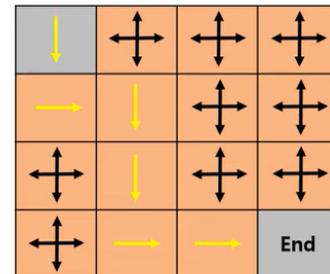
❖ ϵ -Greedy Policy

- Model-Free의 경우 경험한 상태-행동 가치에 대해서만 업데이트 됨
- 따라서 정책 개선 후에 특정 상태에서 특정 행동만 선택하도록 패턴이 고착될 수 있음
- 이를 타파하기 위해서 일정 확률로 무작위 행동을 선택하도록 하는 장치가 필요함

Action-State Value (Episode 1)



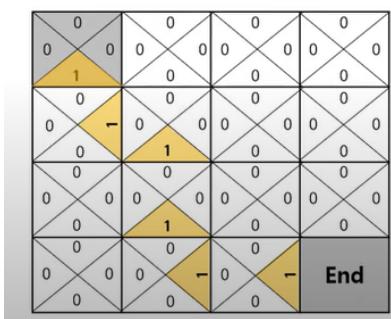
Greedy Policy



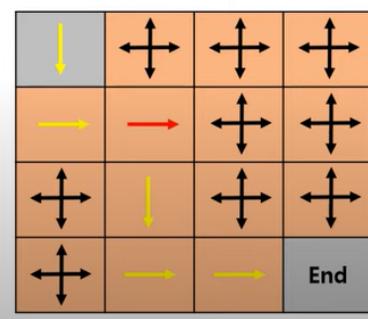
❖ ϵ -Greedy Policy

- 작은 확률(ϵ)로 랜덤한 액션을 선택하는 정책
- 그리디 행동을 취하면서도 다른 환경에 대한 정보를 탐색할 수 있음
- 시작할 때 ϵ 값을 높게 준 뒤 점점 줄이는 방식으로 초반에 환경에 대한 탐색에 중심을 두어 정보를 효과적으로 탐색하는 전략도 있음

Action-State Value (Episode 1)



ϵ -Greedy Policy



Control - On-policy vs Off-policy

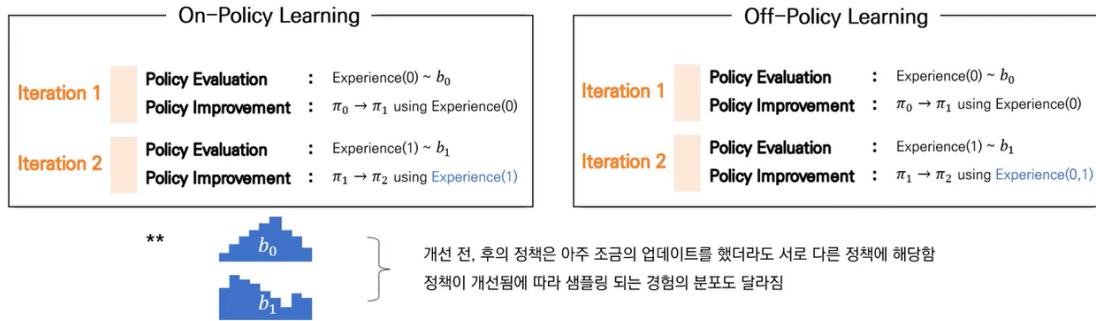
❖ On-Policy, Off-Policy 방법론 간의 비교

- Behavior Policy : 실제로 환경과 상호 작용하며 경험을 쌓는 정책, 정책 평가 단계에서 사용됨
- Target Policy : 강화하고자 하는 목표가 되는 정책, 정책 개선 단계에서 사용됨
- Behavior Policy와 Target Policy가 같은 경우 → On-Policy Learning
- Behavior Policy와 Target Policy가 다른 경우 → Off-Policy Learning

Category 1	Monte-Carlo	Temporal Difference	
Category 2	On-Policy	On-Policy	Off-Policy
Learning Method	On-Policy MC	SARSA	Q-Learning
Behavior Policy	ϵ -Greedy	ϵ -Greedy	ϵ -Greedy
Target Policy	ϵ -Greedy	ϵ -Greedy	Greedy

❖ On-Policy, Off-Policy 방법론 간의 비교 (경험 재사용)

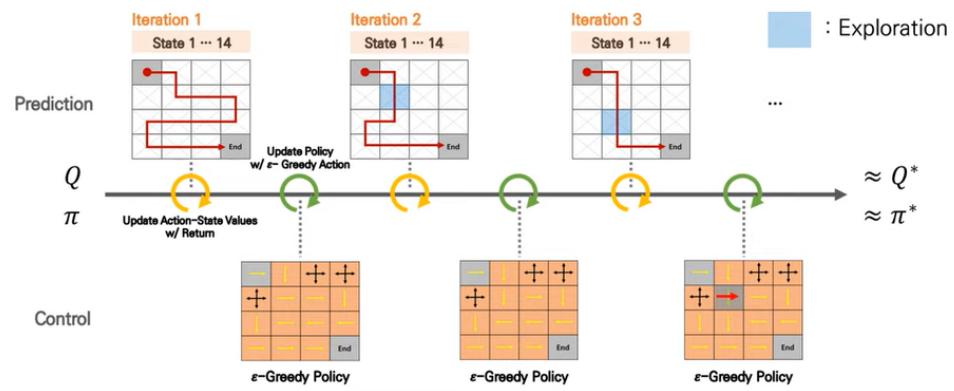
- On-Policy의 경우 현재 정책으로 쌓은 경험으로만 현재 정책을 개선할 수 있음
- Off-Policy의 경우 현재 정책 뿐만 아니라 과거에 쌓은 경험으로도 현재 정책을 개선할 수 있음
- 따라서 Off-Policy가 On-Policy보다 데이터 효율적인 학습이 가능함



▼ Control - On-policy Monte-Carlo

❖ On-Policy Monte-Carlo

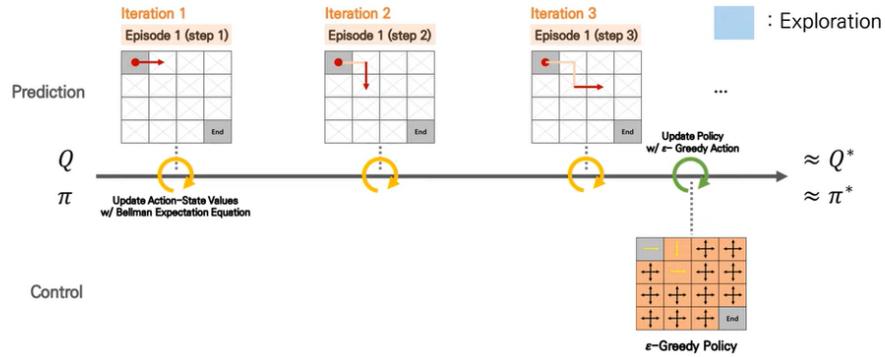
- 정책 평가(Behavior Policy) : ϵ -Greedy (MC Prediction)
- 정책 개선(Target Policy) : ϵ -Greedy
- 상태-행동 가치 함수 및 리턴을 사용함 : $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * G_t$



▼ Control - SARSA

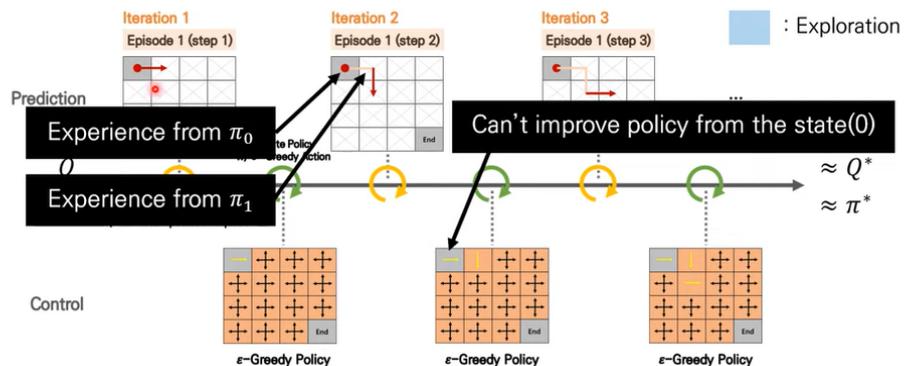
❖ SARSA

- 정책 평가(Behavior Policy) : ϵ -Greedy, TD Prediction
- 정책 개선(Target Policy) : ϵ -Greedy
- 상태-행동 가치 함수 및 벨만 기대 방정식을 사용함: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$



❖ SARSA

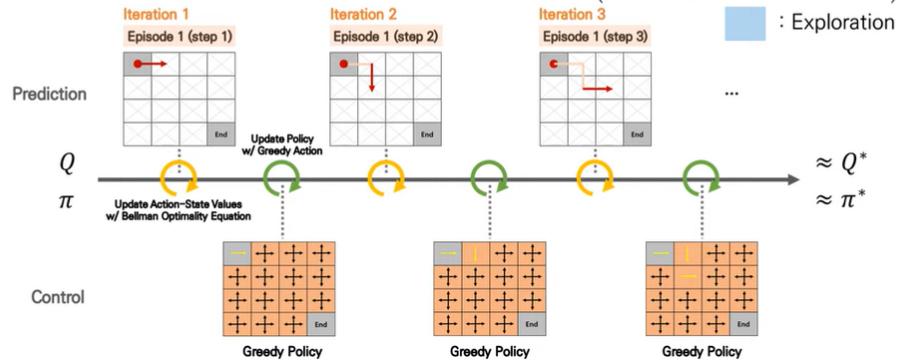
- 정책 평가(Behavior Policy) : ϵ -Greedy, TD Prediction
- 정책 개선(Target Policy) : ϵ -Greedy
- 상태-행동 가치 함수 및 벨만 기대 방정식을 사용함: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$



▼ Control - Q-Learning

❖ Q-Learning

- 정책 평가(Behavior Policy) : ϵ -Greedy, TD Prediction
- 정책 개선(Target Policy) : Greedy
- 상태-행동 가치 함수 및 벨만 최적 방정식을 사용함: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$



❖ SARSA가 서로 다른 정책을 사용하여 학습할 수 없는 이유

- SARSA는 행동-상태 가치를 계산할 때 벨만 기대 방정식을 사용함
- 벨만 기대 방정식은 주어진 정책 함수의 확률적인 요소에 따라서 행동을 취하고 이에 대한 리턴을 계산하도록 함

$$q_{\pi}(s, a) = r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_{\pi}(s', a')$$
- 특정 정책의 확률이 고려된 리턴이기 때문에 향후 개선된 정책에게는 다른 확률분포에서 나온 데이터에 해당하여 정책 개선에 사용될 수 없음

❖ Q-Learning이 서로 다른 정책을 사용하여 학습할 수 있는 이유

- Q-Learning은 행동-상태 가치를 계산할 때 벨만 최적 방정식을 사용함
- 벨만 최적 방정식은 특정된 정책 함수가 없으며 정해진 환경에 따라서 최적 정책이 부여되어 이에 대한 리턴을 계산하도록 함

$$q_*(s, a) = r_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s', a') \quad \leftarrow \text{별도의 } \pi \text{가 명시되지 않음}$$

- 즉, 리턴을 계산하는 구조 상 특정 정책에 제한되지 않으므로 Behavior Policy와 Target Policy가 달라도 학습이 가능함