김범수

MDP와 DP

MDP and DP

By POSCAT

Contents

강화학습에 본격적으로 들어가기 전에…

- 강화학습의 목표는 행동(action)을 취하여 보상(reward)을 최대화하는 것을 목표로 합니다.
- 행동을 하는 주체(agent)는 주어진 환경(environment)에서 관찰(observation)을 통해 행동을 결정합니다.
- observation -> action -> reward -> observation -> action -> reward -> … 의 순으로 agent는 environment와 상호작용합니다.
- Environment의 상태(state)는 observation을 통해 모두 관찰될 수도 있고(ex. 아타리 게임) 부분적으로 관찰(ex. 블랙잭)될 수도 있습니다.
- Environment의 다음 상태는 현재 상태에만 영향을 받고 과거의 상태에는 영향을 주지 않는다고 가정하고 이를 Markov state라 합니다.

$$\mathbb{P}(S_{t+1}|S_t) = \mathbb{P}(S_{t+1}|S_1, S_2, S_3, \dots, S_t)$$

강화학습에 본격적으로 들어가기 전에…

- Agent는 policy, value function, model 세 가지로 구성되어 있습니다.
- policy는 agent의 행동(behavior)를 의미합니다. state가 주어질 때, action을 선택합니다.
- state에서 하나의 행동만 하는 policy를 deterministic policy라 하고, state에서 여러 action중 하나를 확률적으로 선택하는 policy를 stochastic policy라 합니다.

$$deterministic : a = \pi(s), stochastic : \pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

- value function은 미래의 보상을 예측하는 함수입니다. policy가 주어질 때의 보상을 예측합니다. 이 때 γ는 discount factor라 하고, 이 값이 클수록 미래의 보상을 중시한다는 의미를 가집니다.

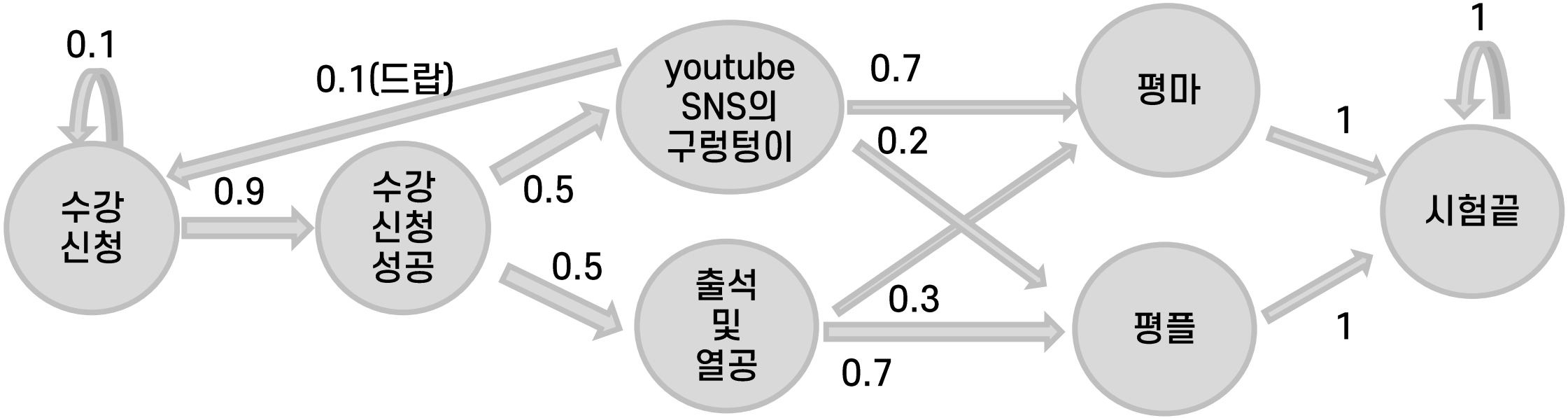
$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$$

- model은 action을 취한 후, 다음 state와 action을 통해 받는 reward를 추측합니다.

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \qquad \mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

Markov Process

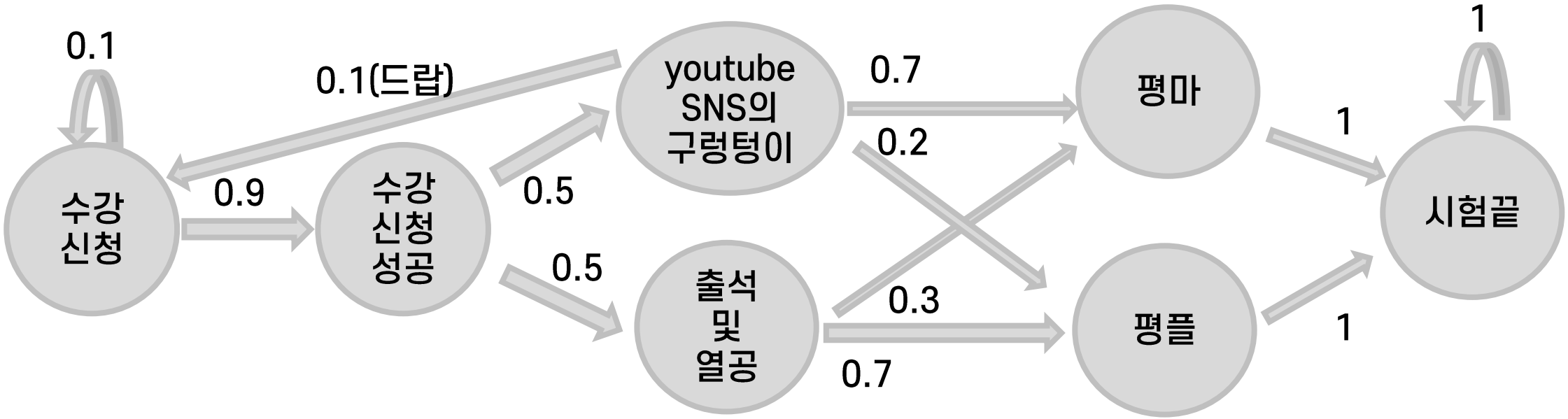
- 우리가 수업을 듣는 과정을 예로 들어 설명합니다.
- Markov Process(MP)는 state set과 state 전이 확률 튜플 $< \mathcal{S}, \mathcal{P} >$ 을 의미합니다.
- $\mathcal{S}$는 (finite) state를 모아둔 set을 의미합니다.
- $\mathcal{R}$는 state transition probability matrix를 의미합니다. $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$



Markov Process

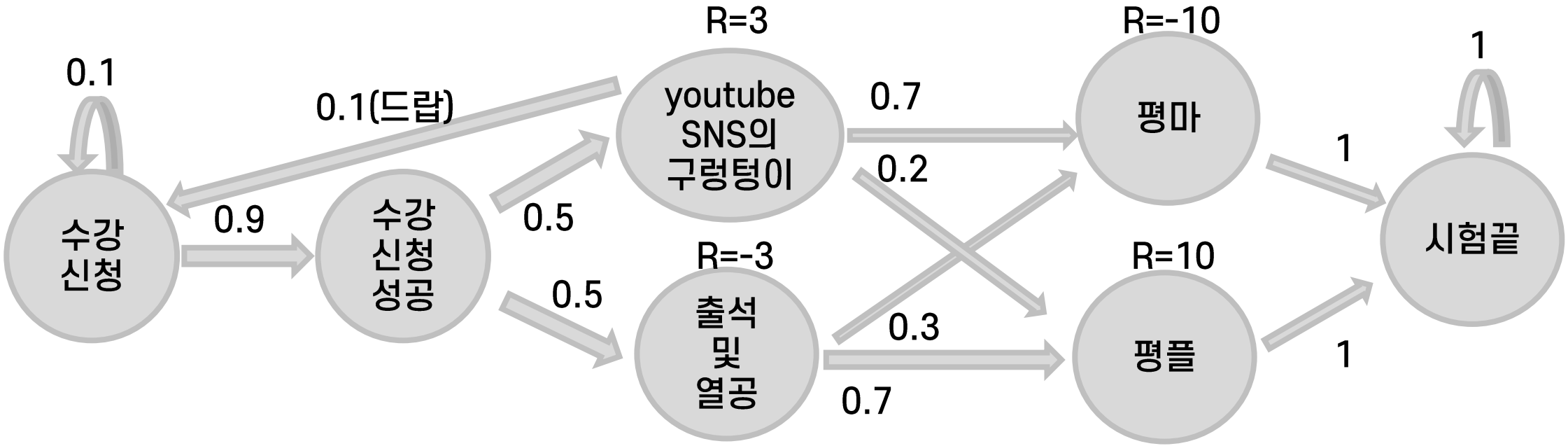
- $S = \{$수강신청, 수강신청성공, 구렁텅이, 열공, 평마, 평쁠, 시험끝$\}$

$$\mathcal{P} = \begin{array}{c} \\ \text{수강신청} \\ \text{신청성공} \\ \text{구렁텅이} \\ \text{열공} \\ \text{평마} \\ \text{평쁠} \\ \text{시험끝} \end{array} \begin{array}{c} \begin{array}{ccccccc} \text{수강신청} & \text{신청성공} & \text{구렁텅이} & \text{열공} & \text{평마} & \text{평쁠} & \text{시험끝} \end{array} \\ \begin{pmatrix} 0.1 & 0.9 & & & & & \\ & & 0.5 & 0.5 & & & \\ 0.1 & & & & 0.7 & 0.2 & \\ & & & & 0.3 & 0.7 & \\ & & & & & & 1 \\ & & & & & & 1 \\ & & & & & & 1 \end{pmatrix} \end{array}$$



Markov Reward Process

- Markov Reward Process(MRP)는 MP에서 reward function와 discount factor 개념을 추가한 튜플 $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$입니다.
- $\mathcal{R}_s = \mathbb{E}[R_{t+1}|S_t = s]$, 즉 $\mathcal{R}_s$는 state s에서 받을 수 있는 예상 보상을 의미합니다.
- γ는 0이상 1이하의 discount factor입니다.

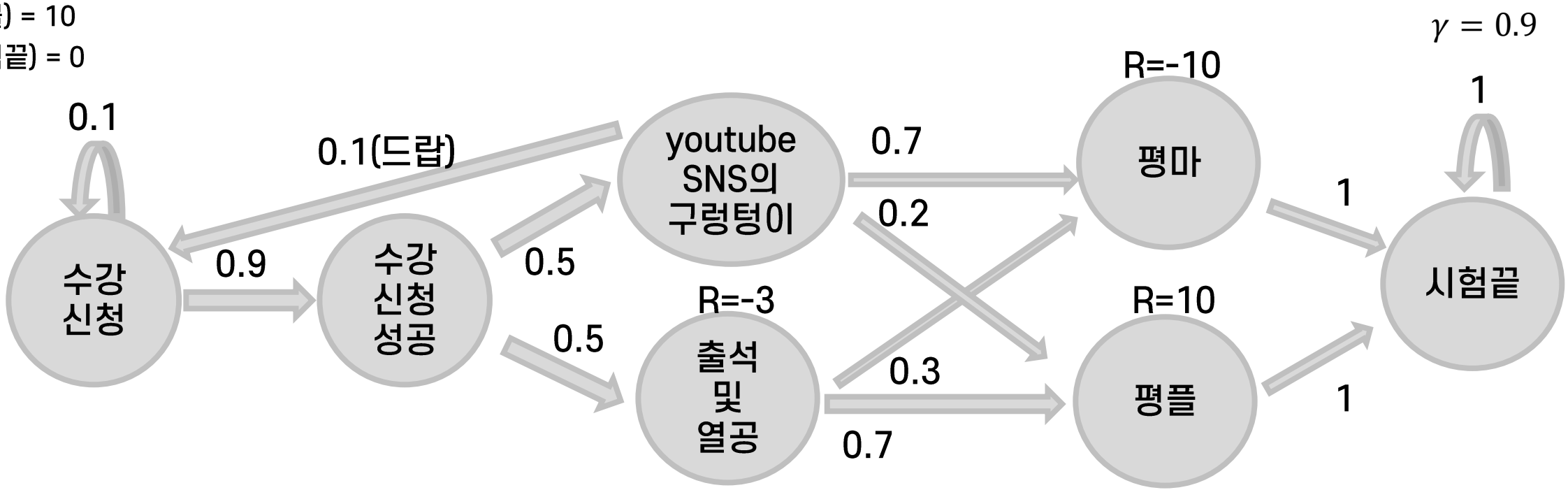


Markov Reward Process

- 현재 시간 t부터 “앞으로 받게 될 총 reward”를 $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$ 으로 정의할 수 있습니다. 이 때 우리는 value function을 아래와 같이 정의할 수 있습니다.
- $v(s) = \mathbb{E}[G_t | S_t = s]$, 즉 state s에서 받을 수 있는 총 reward의 기댓값으로 정의할 수 있습니다.
- $v(s) = \mathbb{E}[G_t | S = s] = \mathbb{E}[R_{t+1} + \gamma v(S_t) | S_t = s] = R_s + \gamma \sum_{s' \in \mathcal{S}} P_{ss'} v(s')$
- 즉, v(s), state s에서 받을 수 있는 총 reward의 기대 가치는 해당 state에서 받을 수 있는 reward R_s와 해당 state에서 미래에 받을 수 있는 reward $\sum_{s' \in \mathcal{S}} P_{ss'} v(s')$에 discounted count γ로 정의할 수 있습니다.

Markov Reward Process

v(수강신청) = -0.374
v(신청성공) = -0.420
v(구렁텅이) = -1.534
v(열공) = 0.6
v(평마) = -10
v(평쁠) = 10
v(시험끝) = 0



Markov Decision Process

- Markov Decision Process(MDP)는 이번 수업의 핵심으로 MRP에서 agent의 action을 추가한 튜플 $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ 입니다. 이 때, $\mathcal{P}, \mathcal{R}$의 정의가 약간 달라지게 됩니다. 다시 한 번 튜플의 각 원소의 정의를 정리해보면,
- $\mathcal{S}$ 는 state의 set을 의미합니다.
- $\mathcal{A}$ 는 action의 set을 의미합니다.
- $\mathcal{P}$는 state transition table matrix를 의미합니다. 이 때, $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$, 즉 state s에서 action a를 취했을 때, state s'으로 전이할 확률을 의미합니다.
- $\mathcal{R}$는 reward fuction을 의미합니다. 이 때, $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$, 즉 state s에서 action a를 취했을 때 받을 수 있는 보상의 평균을 의미합니다.
- γ 는 discount factor를 의미합니다. 이 값이 클수록 미래의 보상을 중요시하고, 작을수록 지금 당장의 보상을 중요시합니다.

Markov Decision Process

- MDP에는 action을 선택하는 정책, policy가 필요합니다. 이 때 policy π는 아래와 같이 정의합니다.
- $\pi(\mathrm{a}|\mathrm{s}) = \mathbb{P}[A_t = a|S_t = s]$, 즉 state s에서 action a를 선택할 확률로 정의합니다.
- MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$ 와 policy π가 주어졌다고 가정해봅시다. 그러면 우리는 이것을 MRP $< \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma >$ 라 할 수 있습니다. 이 때,
- $\mathcal{P}^{\pi}_{ss'} = (state\ s\text{에서 } state\ s'\text{으로 전이할 확률}) =$

$$(state\ s\text{에서 } action\ a\text{를 취할 확률})\ X(state\ s\text{에서 } action\ a\text{를 취했을 때 } state\ s'\text{으로 전이할 확률}) = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{P}^{a}_{ss'}$$

- $\mathcal{R}^{\pi}_{s} = (state\ s\text{에서 받는 } reward) =$

$$(state\ s\text{에서 } action\ a\text{를 취할 확률})\ X\ (state\ s\text{에서 } action\ a\text{를 취했을 때 받는 } reward) = \sum_{a \in \mathcal{A}} \pi(a|s)\mathcal{R}^{a}_{s}$$

Markov Decision Process

- 우리의 목표는 MDP $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$가 주어졌을 때 보상을 최대화하는 것입니다. 이를 위해서 우리는 optimal value function과 optimal policy를 구해야 합니다.
- optimal value function $v_*(s)$는 모든 policy 에 대해 $v(s)$의 최댓값, 즉 $v_*(s) = max_\pi \, v_\pi(s)$ 로 정의됩니다. 다시 말하면 $v_*(s)$는 total reward가 최대가 되도록 하는 action을 선택했을 때, 그 때의 total reward라 할 수 있습니다. 이를 수식으로 다시 적어보면

$$v_*(s) = max_a((state\ s\text{에서 } action\ a\text{를 취할 때의 보상}) +$$

$$\gamma \sum_{s' \in S} (state\ s\text{에서 } action\ a\text{를 취할 때 } state\ s'\text{으로 전이할 확률}) X (state\ s'\text{에서의 } total\ reward))$$

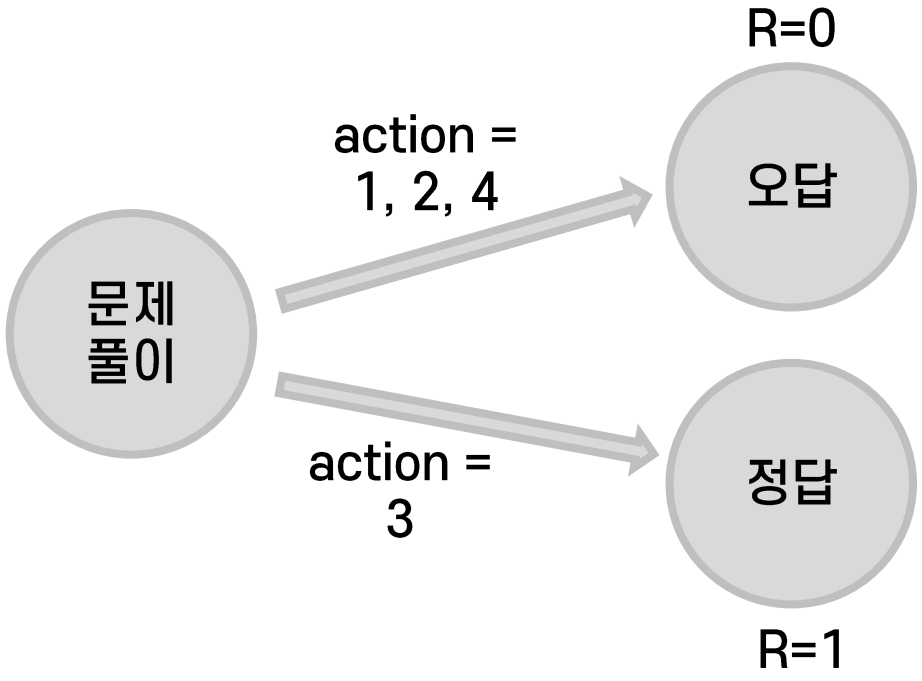
$$= max_a(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_*(s'))$$

Markov Decision Process

- optimal policy를 정의하기 위해서는 policy간의 ordering을 해야 합니다.
- 모든 state s에 대해 $v_\pi(s) \geq v_{\pi'}(s)$일 때, $\pi \geq \pi'$이라 할 수 있습니다. 쉽게 말하면 2개의 전략이 있고, 어떤 상황에서도 한 전략이 다른 전략보다 더 많은 total reward를 받을 때, 한 쪽의 전략이 더 좋다고 할 수 있습니다.
- 그렇다면 optimal policy는 최고의 전략이 됩니다. 수식으로 하면, 모든 policy π에 대해 $v_{\pi*}(s) \geq v_\pi(s)$일 때 π_*는 optimal policy라 할 수 있습니다.
- MDP에서 optimal value function을 알고 있을 때는 optimal value가 최대가 되도록 하는 state로 전이하는 action을 선택하는 것이 optimal policy가 된다는 것을 알 수 있습니다. 즉 가장 보상이 많은 action을 취하는 것이 optimal policy입니다. 따라서 MDP에서의 policy는 항상 deterministic policy가 존재합니다.
- 지금까지의 내용을 4지선다 문제를 푸는 예제로 정리해보겠습니다. 총 4개의 action(1번을 선택, 2번을 선택…)이 존재하고, 정답을 맞출 경우, 1의 보상을 받고 틀리면 0의 보상을 받는 간단한 MDP를 생각해보겠습니다. (3번이 정답이라 가정합니다.)

Markov Decision Process

- optimal value function를 알고 있다고 가정해보죠. 그 말은 3번을 체크하면 1점을 얻고, 다른 번호를 체크하면 0점을 얻는다는 것을 알고 있습니다. 그렇다면, 문제풀이 state에서의 optimal policy는 3을 취하고 1점을 받는 action을 취하는 것임을 알 수 있습니다.
- 마찬가지로, optimal policy가 3임을 알고 있다고 가정하면, 문제풀이 state에서의 optimal value는 3번을 체크했을 때의 점수인 1임을 알 수 있습니다.
- 하지만 우리는 optimal value function과 optimal policy를 모두 모르는 경우가 대부분입니다. 이런 경우에는 어떤 방법으로 optimal policy를 구해야 할까요?



잠깐 복습 타임

- 지금까지 우리는 MDP를 정의해보았습니다. MDP는 상태를 모아놓은 state set, 내가 할 수 있는 행동을 모아둔 action set, 내가 특정 state에서 특정 action을 취할 때 어떤 state로 이동할 지에 대한 확률을 모아둔 state transition matrix, 특정 state에서 받는 reward지에 대한 reward function, 그리고 미래의 reward를 얼마나 중요시할 지를 결정할 gamma로 구성됩니다.
- value function은 특정 state에서 내가 미래 가치를 포함해 받을 수 있는 total reward입니다.
- policy는 특정 state에서 내가 어떤 action을 할 지를 정의한 것입니다.
- optimal value function은 특정 state에서 내가 최대로 받을 수 있는 total reward입니다.
- optimal policy는 내가 total reward를 최대로 받을 수 있도록 하는 action입니다.
- MDP에서는 항상 deterministic optimal policy가 존재합니다.

Dynamic Programming

- DP는 policy evaluation과 policy improvement의 두 과정으로 나눠집니다.
- policy evaluation은 policy π가 주어졌을 때, value function을 구하는 과정입니다. 우리의 목표는 v_π를 구하는 것입니다.
- 먼저 모든 state에서의 value를 0으로 초기화합니다. 이 때의 value function을 v_1이라 하면, 모든 state s에 대해 $v_1(s) = 0$ 라 할 수 있습니다.
- v_k를 알고 있다고 가정하고, v_k값을 이용하여 v_{k+1}을 구합니다. 이 때 v_{k+1}을 update하는 식은 아래와 같습니다.
- $v_{k+1}(s) = \sum_{a\in\mathcal{A}}$(state s에서 action a를 할 확률) X (state s에서 받는 보상 $+ \gamma$ X 미래에 받을 보상) , 이 때 미래에 받을 보상은v_k로 표현하면, (미래에 받을 보상)

  = (state s에서 action a를 취했을 때 state s′으로 이동할 확률) X (state s′에서의 total reward)

  $= \sum_{s'\in\mathcal{S}} P^a_{ss'} v_k(s')$로 표현할 수 있습니다.
- k가 커질수록 v_π로 수렴합니다. 다음 페이지에서 미로 예제를 통해 policy evaluation을 해봅니다.

Dynamic Programming

- 4X4의 미로가 주어져있습니다. 이 때, (0, 0)과 (3, 3)은 도착지점으로 terminal state이고, 그 이외의 14개의 state가 존재합니다.
- 총 4개의 action(위쪽, 아래쪽, 오른쪽, 왼쪽)이 가능합니다.
- terminal state에 도착할 때까지 한 걸음 이동할 때마다 -1의 보상이 주어집니다.
- 만약 미로 밖으로 이동하면, 위치는 변하지 않습니다.
- policy는 랜덤 policy, 즉 $\pi(N|\cdot) = \pi(S|\cdot) = \pi(E|\cdot) = \pi(S|\cdot) =$ 0.25라고 가정합니다.
- discount factor $\gamma = 1$로 가정합니다.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Dynamic Programming

- v2 value function 값을 계산해보면, 어느 칸으로 이동하든 -1의 보상을 받고, 미래에 받을 보상은 v1이 모두 0으로 초기화되어있으므로, 0입니다. 따라서 terminal state를 제외한 모든 state의 value function 값은 -1로 update됩니다.

v1

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

v2

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

Dynamic Programming

- (0, 1)에서의 v3 value function 값, v3(0,1)을 구해봅시다.
- 먼저 왼쪽으로 이동할 경우, -1의 보상을 받고, 미래의 예상 보상은 0임을 알 수 있습니다. 그 이외의 방향으로 이동할 경우, value function값이 -1인 칸에 도착하므로, 미래의 예상 보상이 -1임을 알 수 있습니다.
- 따라서 v3(0,1)
  = (왼쪽으로 이동할 확률) x -1 + (그 이외의 방향으로 이동할 확률) x -2
  = 0.25 x -1 + 0.75 x -2 = -1.75가 됩니다. (오른쪽 표에는 소수점 1자리까지 표기합니다.)

v2

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

v3

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

Dynamic Programming

- 마찬가지로 v4(0, 1)을 계산해보겠습니다.
- 왼쪽으로 이동할 경우, -1의 보상을 받고 예상 미래 보상은 0으로 총 -1의 보상을 받게 됩니다.
- 오른쪽으로 이동할 경우, -1의 보상을 받고 예상 미래 보상은 -2.0으로 총 -3의 보상을 받게 됩니다.
- 아래쪽으로 이동할 경우, -1의 보상을 받고 예상 미래 보상은 -2.0으로 총 -3의 보상을 받게 됩니다.
- 위쪽으로 이동할 경우, -1의 보상을 받고 예상 미래 보상은 -1.7으로 총 -2.7의 보상을 받게 됩니다.
- v4(0, 1) = 0.25 x -1 + 0.25 x -3 + 0.25 x -3 + 0.25 x -2.7 = -2.4

v3

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

v4

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

Dynamic Programming

- 이 과정을 무한히 반복하면, 아래와 같은 값으로 수렴하게 되고, 이 값은 곧 v_π가 됩니다.
- 이처럼 무한히 반복하여 v_π구함으로써 policy evaluation은 종료됩니다.

v4

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

v∞

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

Dynamic Programming

- v_π를 구했다면, policy improvement를 진행합니다. policy iteration은 매우 간단합니다.
- 각 state에서 value function값이 최대가 되도록 하는 state로 이동하는 action을 optimal policy로 채택합니다. 그렇다면 policy improvement 은 종료됩니다.
- 실제로 optimal policy를 보면 처음의 랜덤 policy보다 확실히 개선되었음을 알 수 있습니다.

v_∞

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

optimal policy

	W	W	SW
N	NW	SW	S
N	NE	SE	S
NE	E	E	0

Dynamic Programming

- 이처럼 policy evaluation과 policy improvement 한 과정을 거치는 것을 policy iteration이라 합니다.
- policy iteration으로 도출된 optimized policy로 다시 policy iteration을 진행하는 것을 반복합니다.
- policy iteration을 무한정 반복하면, policy는 optimal policy로 수렴됨이 증명되어 있습니다.
- policy evaluation에서 value function을 찾기 위해서는 무한 번 반복해야 하지만, 실제로는 무한 번 반복이 불가능하기 때문에, k번만 반복하거나, 변화하는 정도가 작아질 때까지만 반복하는 등의 방법을 이용합니다.

연습문제

- 이번 연습문제에서는 DP를 이용하여 임의로 주어지는 미로 문제를 해결해봅니다. 미로 문제를 해결할 optimal pi를 찾는 것이 이번 연습문제의 목표입니다.
- 미로의 도착지점이 랜덤이고, 미로의 벽이 존재하여, 벽 방향으로 이동하면 위치가 변하지 않습니다.
- 총 4방향으로 이동이 가능하고, 1칸 이동할 때마다 -1의 reward가 주어집니다.
- 도착지점에서 action을 취할 경우, 아무 방향으로도 이동하지 않고, 0의 reward가 주어집니다.
- 벽이 아닌 모든 지점에서 도작 지점으로 도달이 가능함이 보장되어 있습니다.
- https://github.com/qjatn0120/2020_winter_AI/blob/master/source_code/Lecture3/maze.py
- 위 주소에서 미로 환경을 제공합니다. 복사 붙여넣기나 다운로드를 하여 자신의 컴퓨터에 코드를 저장하세요.

연습문제

- 먼저 maze 모듈에서 미로가 구현된 Maze class를 가져옵니다.

```
from maze import Maze
```

- Maze(가로 길이, 세로 길이, 벽의 출현 빈도)로 미로를 생성합니다.
- env.possible_state() 함수를 이용해 벽이 존재하지 않는 모든 좌표 리스트를 얻을 수 있습니다.
  [(x0, y0), (x1, y1), …]의 형태로 반환합니다.

```
env = Maze(size_x, size_y, wall_rate)
state = env.possible_state()
```

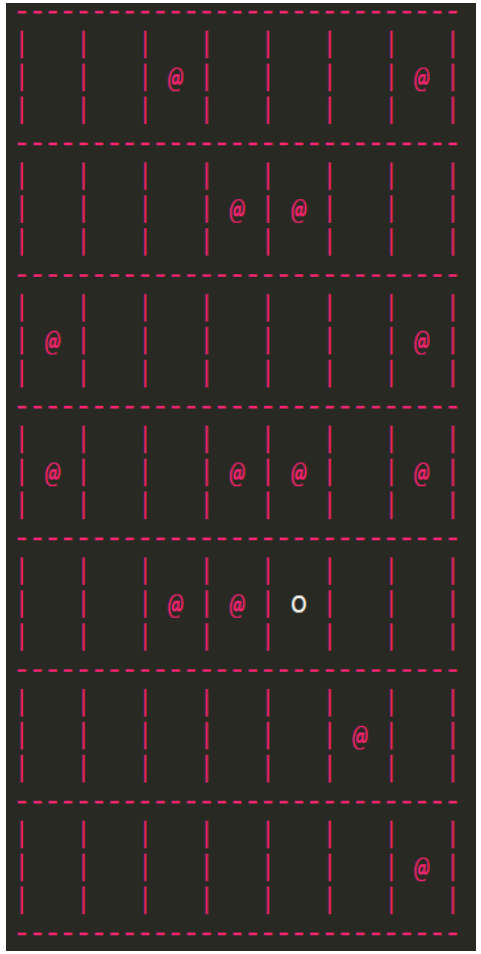
- env.step(x좌표, y좌표, action) 함수는 (x좌표, y좌표)에서 action을 취하는 함수입니다.
  ((action을 취한 후의 x좌표, y좌표), reward, 도착점 도착 여부)를 반환합니다.
  action 0 = 북쪽, 1 = 남쪽, 2 = 서쪽, 3 = 동쪽

```
(px, py), reward, done = env.step(x, y, action)
```

연습문제

- env.display_maze() 함수를 이용하여 미로의 모양을 출력 해볼 수 있습니다.
- 오른쪽은 Maze(7, 7, 0.3)으로 생성한 미로입니다.
- @는 벽을 O는 도착점을 의미합니다.

```
env.display_maze()
```



연습문제

```
env.display_pi(pi)
```

- env.display_pi(pi) 함수를 전달하여 계산한 policy가 optimal policy인지 확인할 수 있습니다. pi[y][x][action]이 양의 확률인 경우, (x, y) 좌표에서 action 방향으로의 화살표가 출력됩니다.
- 오른쪽 그림처럼 정답이 출력된다면 연습문제를 해결한 것입니다.