Markov Decision Process

Younghoon Kim

nongaussian@hanyang.ac.kr

Keywords

- Markov process ★
- Markov decision process ★★★
- Return ★★★★★
- Policy ★★★★★
- Value function ★★★★★
- Bellman equation ★★
- Policy Evaluation ★
- Policy Improvement ★

Today’s Practice: Pathfinding In Gridworld

- 문제
  - 터미널이 아닌 셀에서 시작
  - 정책에 따라 회색으로 음영 처리된 터미널 셀 중 하나에 도달합니다.
- Actions = {up, down, left, right}
  - 보상은 터미널을 제외한 모든 상태에 대해 -1
- Goal
  - 목표에 도달하기 위한 정책 찾기

15	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

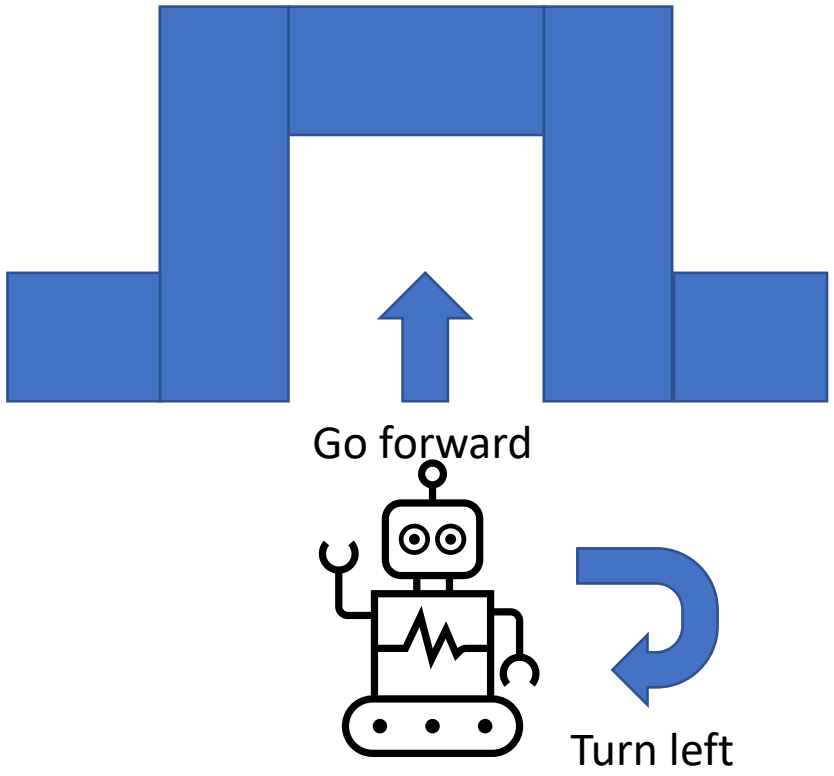
MDPs for Reinforcement Learning

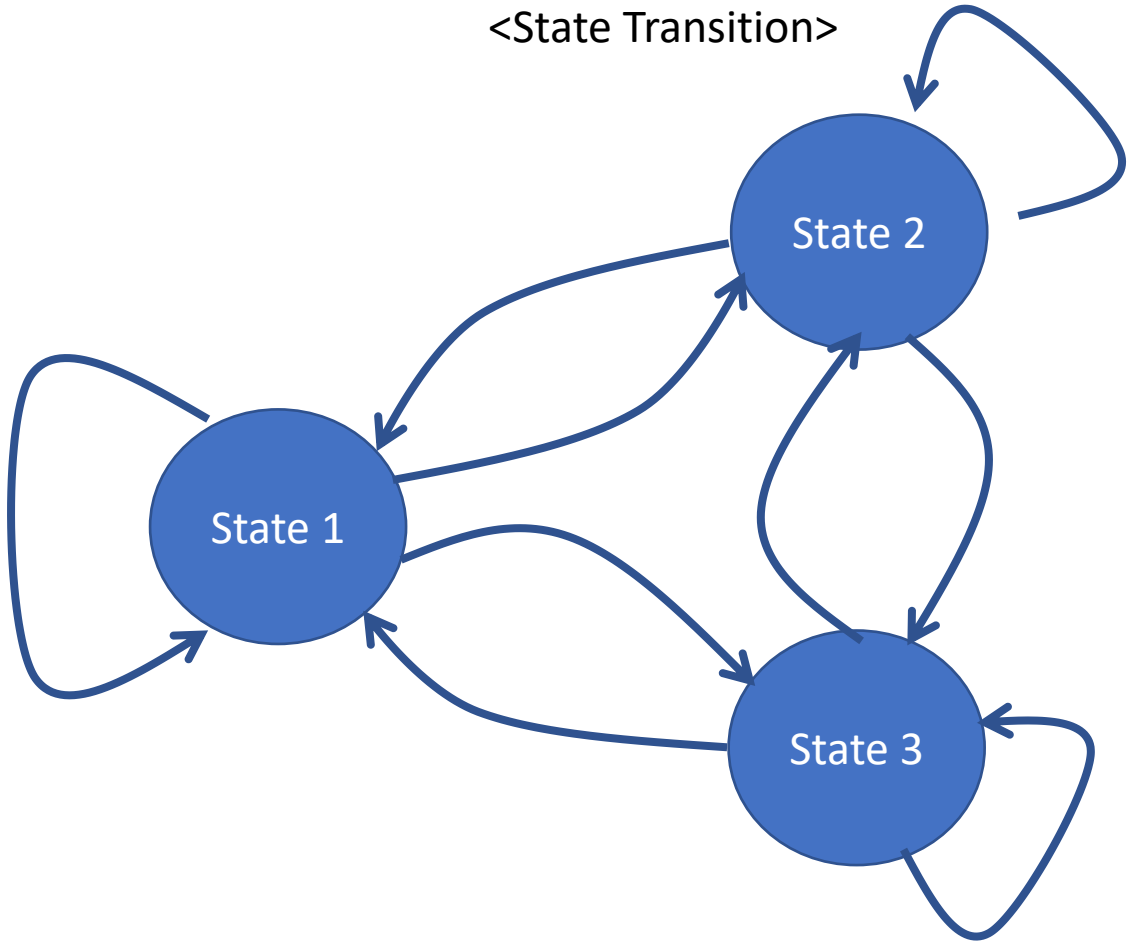
- 환경을 완전히 관찰할 수 있다는 가정이 필요
  - 거의 모든 퍼즐 문제들은 MDP를 사용하여 모델링 가능
- Markov property
  - A state S_t is called Markov if and only if,

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

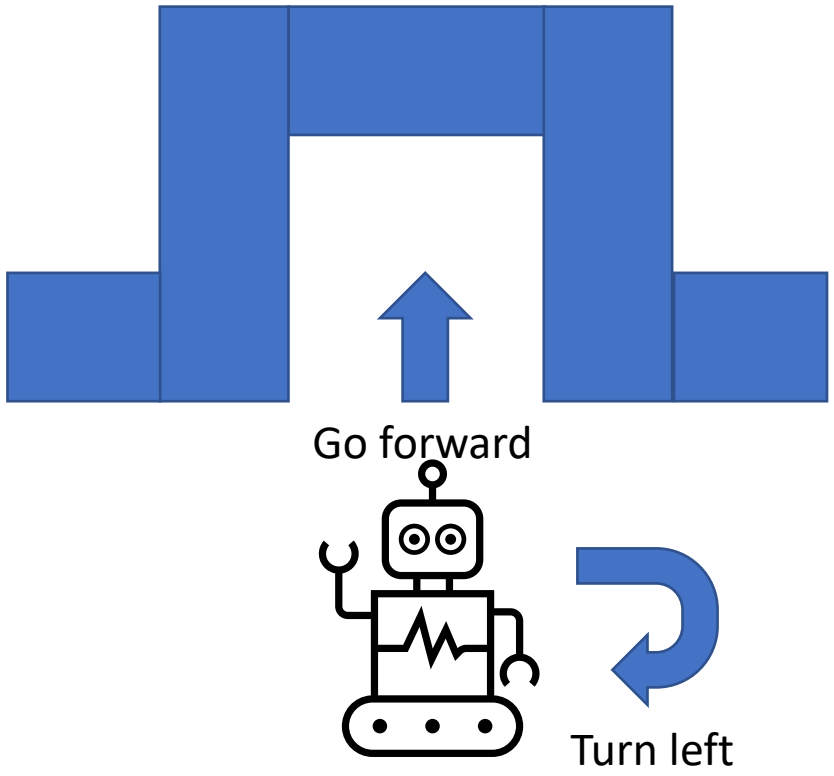
인덱스 t가 있는 대문자 S_t는 t의 확률 변수이고
인덱스 i가 있는 소문자 s_i는 i 번째 상태

A Running Example: Robot Motion

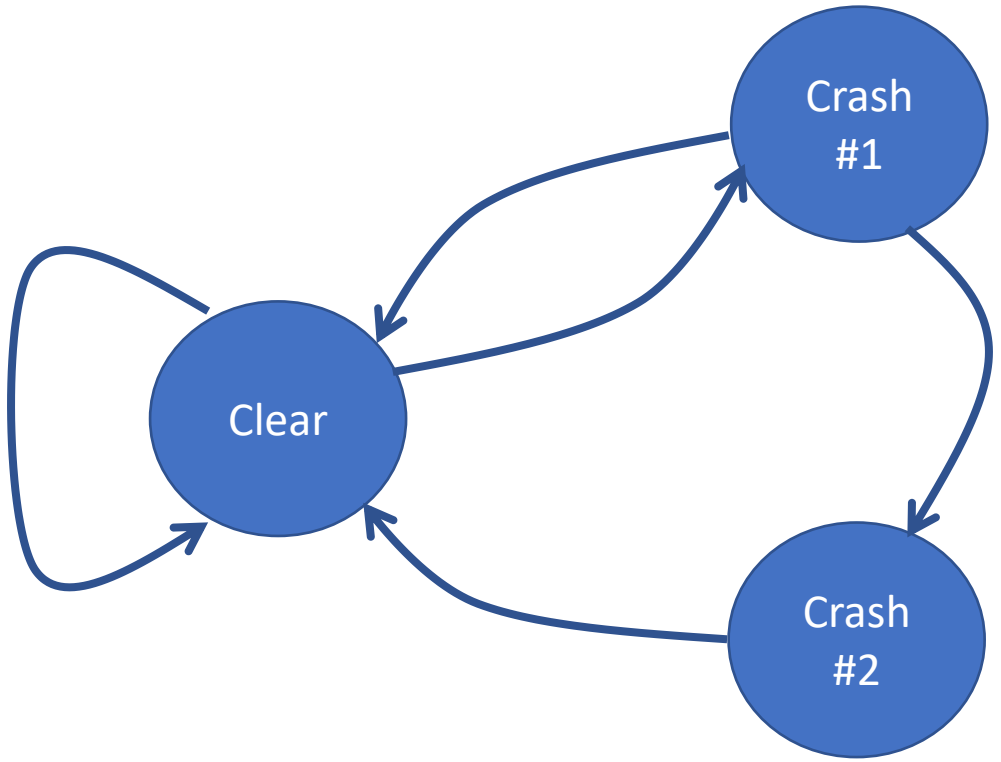




A Running Example: Robot Motion



<State Transition>



State Transition Matrix

- State transition matrix P = { p_{ij} } where p_{ij} is the probability $\Pr(S_{t+1} = s_j | S_t = s_i)$ that the s_i is changed to s_j

$$P = \begin{Bmatrix} p_{11} & \cdots & p_{1n} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nn} \end{Bmatrix}$$

- Then, the probability to reach the state s_j at t + 1 can be calculated as

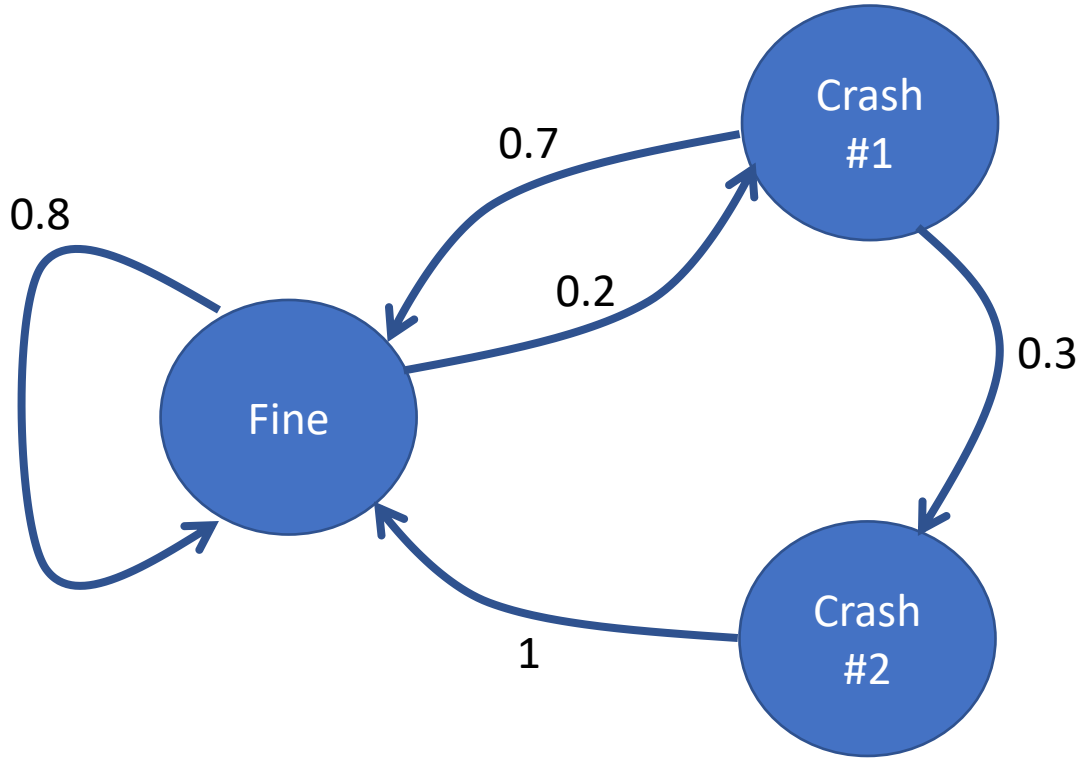
$$\Pr(S_{t+1} = s_j) = \sum_{i=1}^{n} \Pr(S_t = s_i) \cdot p_{ij}$$

Markov Process

- A Markov process is a memoryless random process
- (S, P) is called a Markov process, where

  - S is a finite set of n states $\{s_1, s_2, \ldots, s_n\}$
  - P is a transition probability matrix such that
    $p_{ij} = \Pr(S_{t+1} = s_j | S_t = s_i)$

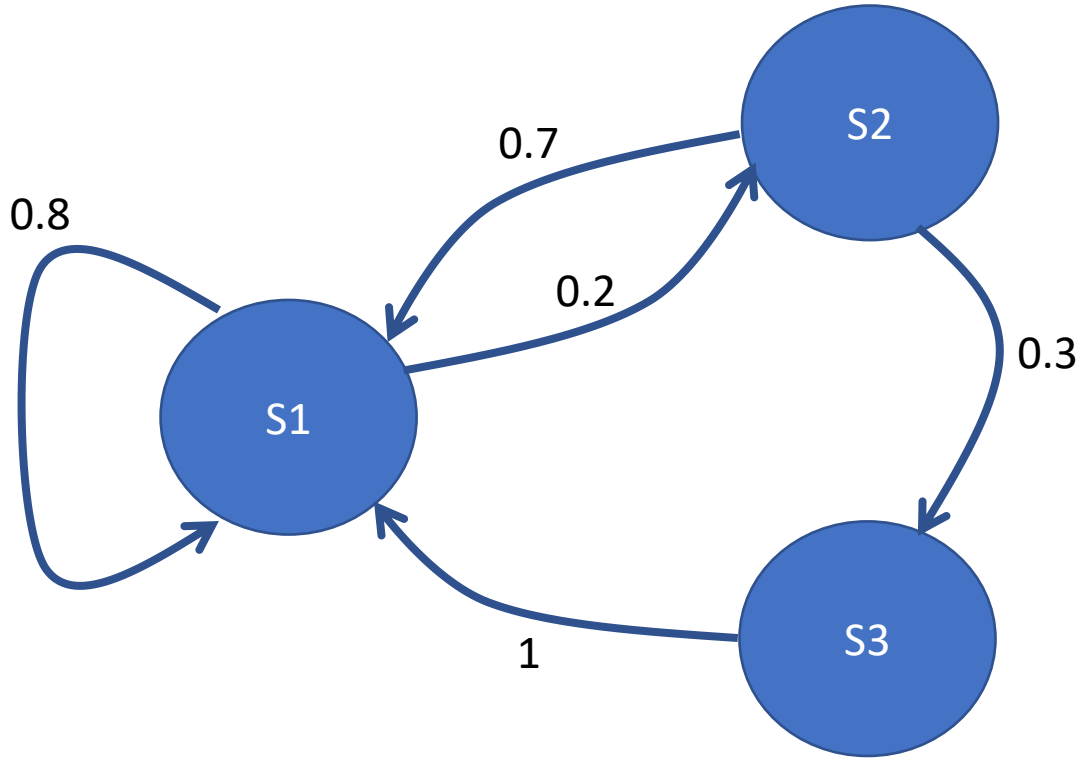
Modeling Robot Motion with MP



$$P = \begin{array}{c} \\ \text{Fine} \\ \text{Crash \#1} \\ \text{Crash \#2} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Fine} & \text{Crash \#1} & \text{Crash \#2} \end{array} \\ \begin{Bmatrix} 0.8 & 0.2 & 0 \\ 0.7 & 0 & 0.3 \\ 1 & 0 & 0 \end{Bmatrix} \end{array}$$

Episodes

- Let maxStep be 4



<S1, S2, S3, S1>

<S1, S2, S1, S1>

<S1, S1, S1, S1>

<S1, S1, S1, S1>

<S1, S2, S1, S2>

<S2, S3, S1, S1>

Markov Reward Process

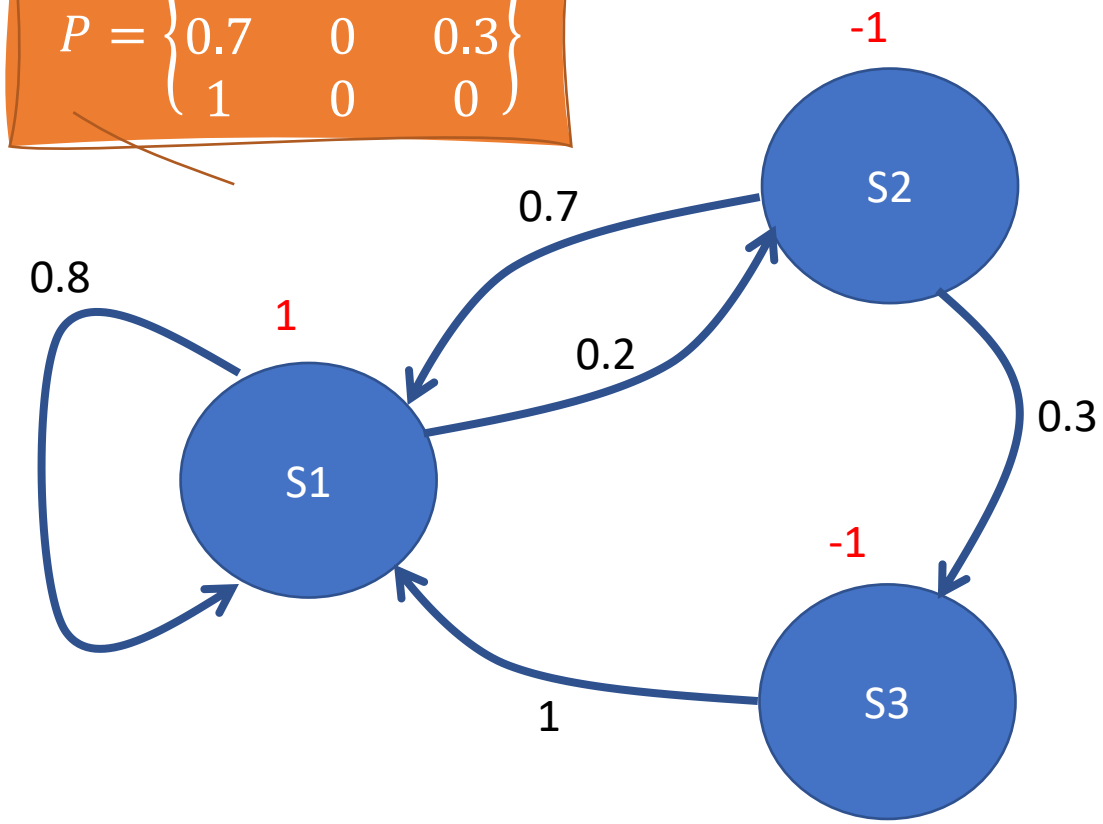
- 각 상태에서 보상 가치를 평가할 수 있다고 가정
- (S, P, R, γ) is called a Markov reward process, where γ is a discount factor between 0 and 1, and
  - S is a finite set of n states $\{s_1, s_2, \ldots, s_n\}$
  - P is a transition probability matrix such that $p_{ij} = Pr(S_{t+1} = s_j | S_t = s_i)$
  - $r(s_i)$ is an expected reward at t+1 from s_i, i.e., $r(s_i) = E[R_{t+1} | S_t = s_i]$

인덱스 t가 있는 대문자 R_t는 t의 확률 변수이고 소문자 $r(s_i)$는 보상 함수

왜 R_{t+1}의 "기대값"을 사용할까? t+1에 어떤 상태가 될지 결정적으로 알 수 없기 때문

Expected Rewards

$$P = \begin{Bmatrix} 0.8 & 0.2 & 0 \\ 0.7 & 0 & 0.3 \\ 1 & 0 & 0 \end{Bmatrix}$$



Q: Let $S_t = s_1$, $r(s_1) = ?$

$$r(s_1) = E[R_{t+1}|S_t = s_1]$$

$$= 1 \cdot \Pr(S_{t+1} = s_1|S_t = s_1)$$

$$+ -1 \cdot \Pr(S_{t+1} = s_2|S_t = s_1)$$

$$+ -1 \cdot \Pr(S_{t+1} = s_3|S_t = s_1)$$

$$= 1 \cdot 0.8 + (-1) \cdot 0.2 + (-1) \cdot 0$$

$$= 0.6$$

Return

- Return G_t
  - 시간 단계 t부터 미래까지의 보상의 총합
  - 일반적으로 보상은 다음 합산식과 같이 미래 보상에 대해 기하급수적으로 할인(할인 상수: γ)된 보상의 합으로 계산

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{i=1}^{\infty} \gamma^{i-1} R_{t+i}$$

Value Function and Bellman Formulation

- MRP의 가치 함수 $v(s_i)$는 현재 상태 s_i에서 예상되는 미래의 총 보상 (즉, $S_t = s_i$일 때의 리턴)

$$
\begin{aligned}
v(s_i) &= E[G_t|S_t = s_i] \\
&= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots |S_t = s_i] \\
&= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots)|S_t = s_i] \\
&= E[R_{t+1} + \gamma G_{t+1}|S_t = s_i] \\
&= E[R_{t+1}|S_t = s_i] + \gamma E[G_{t+1}|S_t = s_i] \\
&= r(s_i) + \gamma E[G_{t+1}|S_t = s_i] \\
&= r(s_i) + \gamma \sum_{j=1}^{n} p_{ij} E[G_{t+1}|S_{t+1} = s_j] \\
&= r(s_i) + \gamma \sum_{j=1}^{n} p_{ij} v(s_j)
\end{aligned}
$$

$$
E[G_{t+1}|S_t = s_i] = \sum_{j=1}^{n} p_{ij} E[G_{t+1}|S_{t+1} = s_j, S_t = s_i]
$$

Value Function and Bellman Formulation

- MRP의 가치 함수 $v(s_i)$는 현재 상태 s_i에서 예상되는 미래의 총 보상 (즉, $S_t = s_i$일 때의 리턴)

$$v(s_i) = E[G_t|S_t = s_i]$$
$$= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots | S_t = s_i]$$
$$= r(s_i) + \gamma \sum_{j=1}^{n} p_{ij} E[G_{t+1}|S_{t+1} = s_j]$$
$$= r(s_i) + \gamma \sum_{j=1}^{n} p_{ij} v(s_j)$$

Bellman Equation:

$$v(s_i) = r(s_i) + \gamma \sum_{j=1}^{n} p_{ij} v(s_j)$$

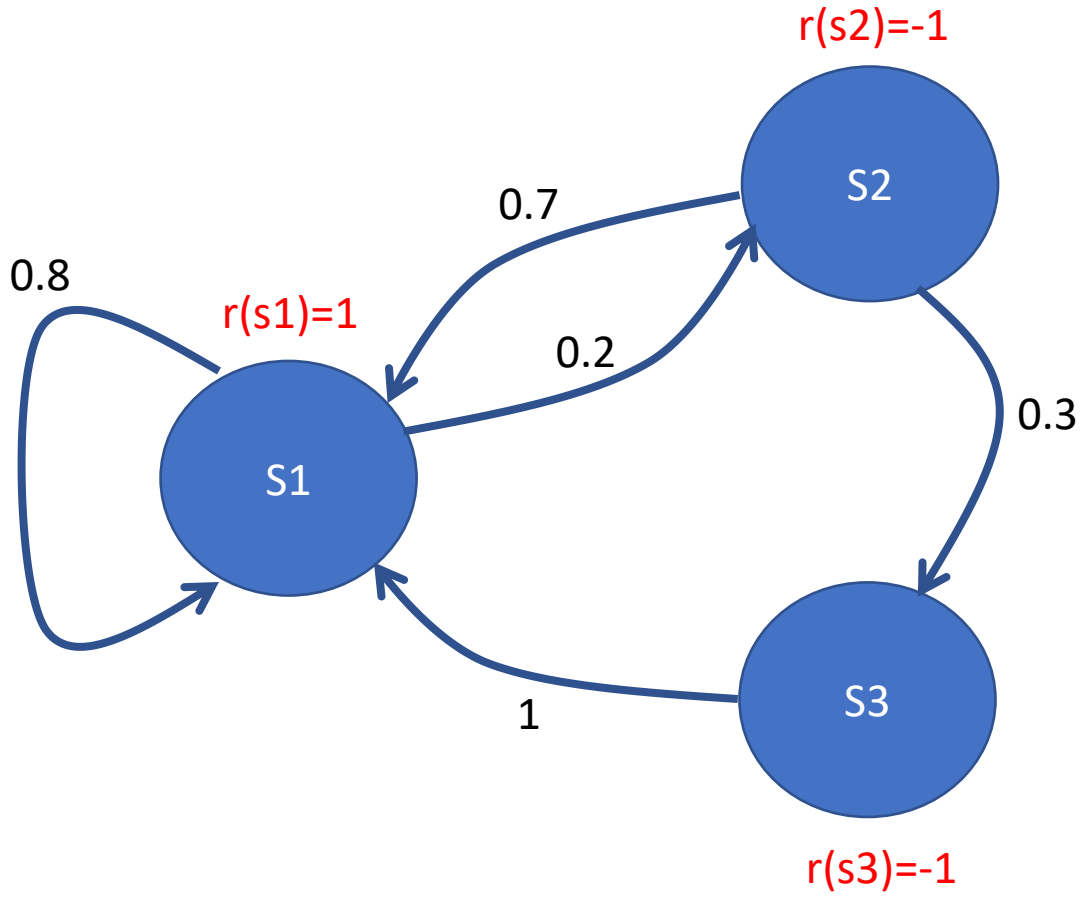
Bellman Function for MRP

- For each state $s_i \in \{s_1, \dots, s_n\}$,
  - $v(s_i) = r(s_i) + \gamma \sum_{j=1}^{n} p_{ij} v(s_j)$ should be hold

n개의 등식과 n개의 변수를 가진 n차 연립방정식!

$v(s_i)$ must satisfy the self-consistency condition given by the Bellman equation

Modeling Robot Motion with MRP



$\gamma = 0.5$

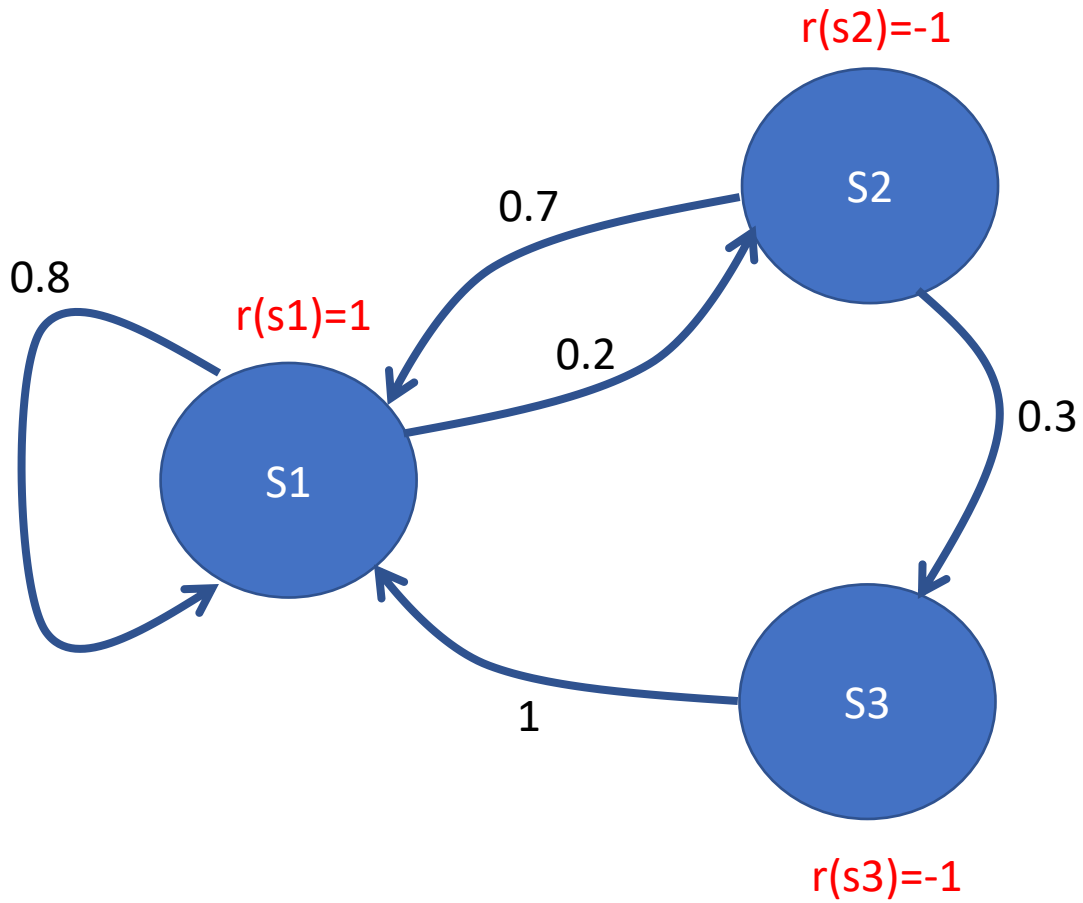
$$v(s_1) = r(s_1) + \gamma\big(0.8 \cdot v(s_1) + 0.2 \cdot v(s_2)\big)$$

$$v(s_2) = r(s_2) + \gamma\big(0.7 \cdot v(s_1) + 0.3 \cdot v(s_3)\big)$$

$$v(s_3) = r(s_3) + \gamma\big(v(s_1)\big)$$

Equations with 3 variables!

Modeling Robot Motion with MRP



$\gamma = 0.5$

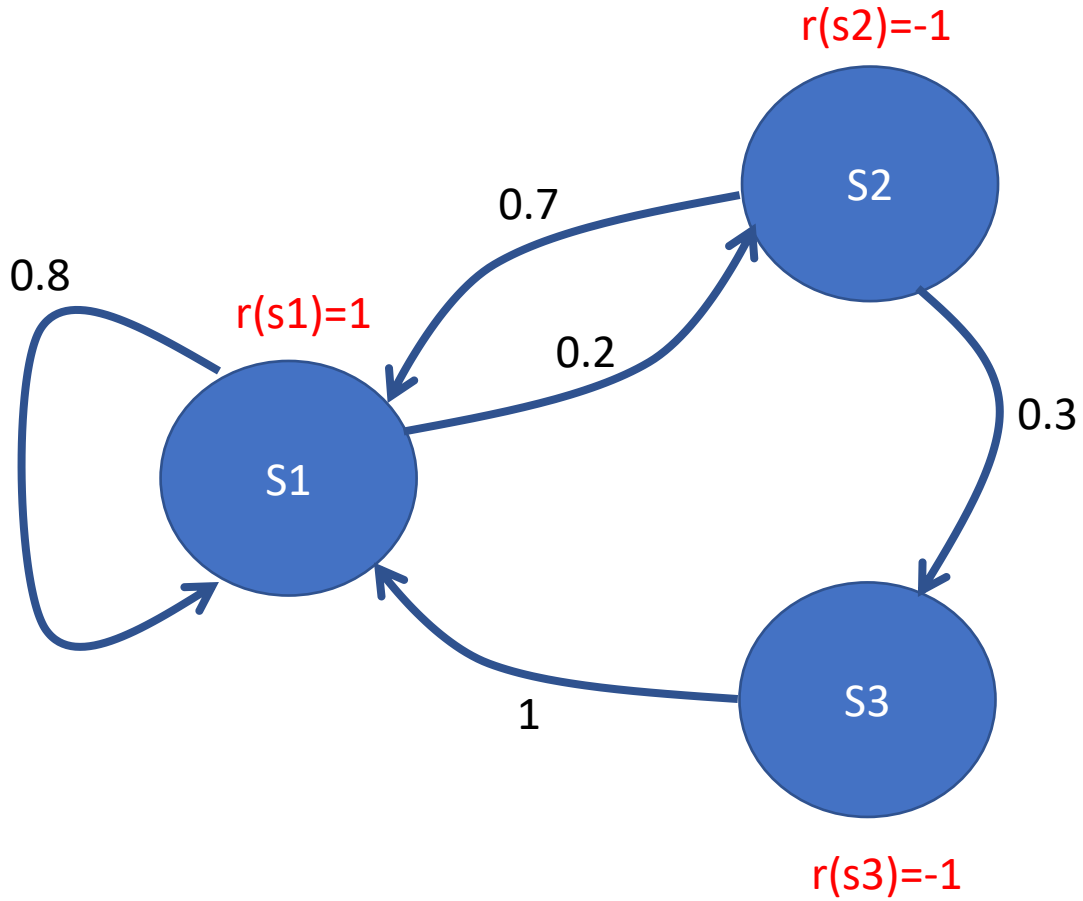
$$x = 1 + 0.5(0.8 \cdot x + 0.2 \cdot y)$$

$$y = -1 + 0.5(0.7 \cdot x + 0.3 \cdot z)$$

$$z = -1 + 0.5(x)$$

Equations with 3 variables!

Modeling Robot Motion with MRP



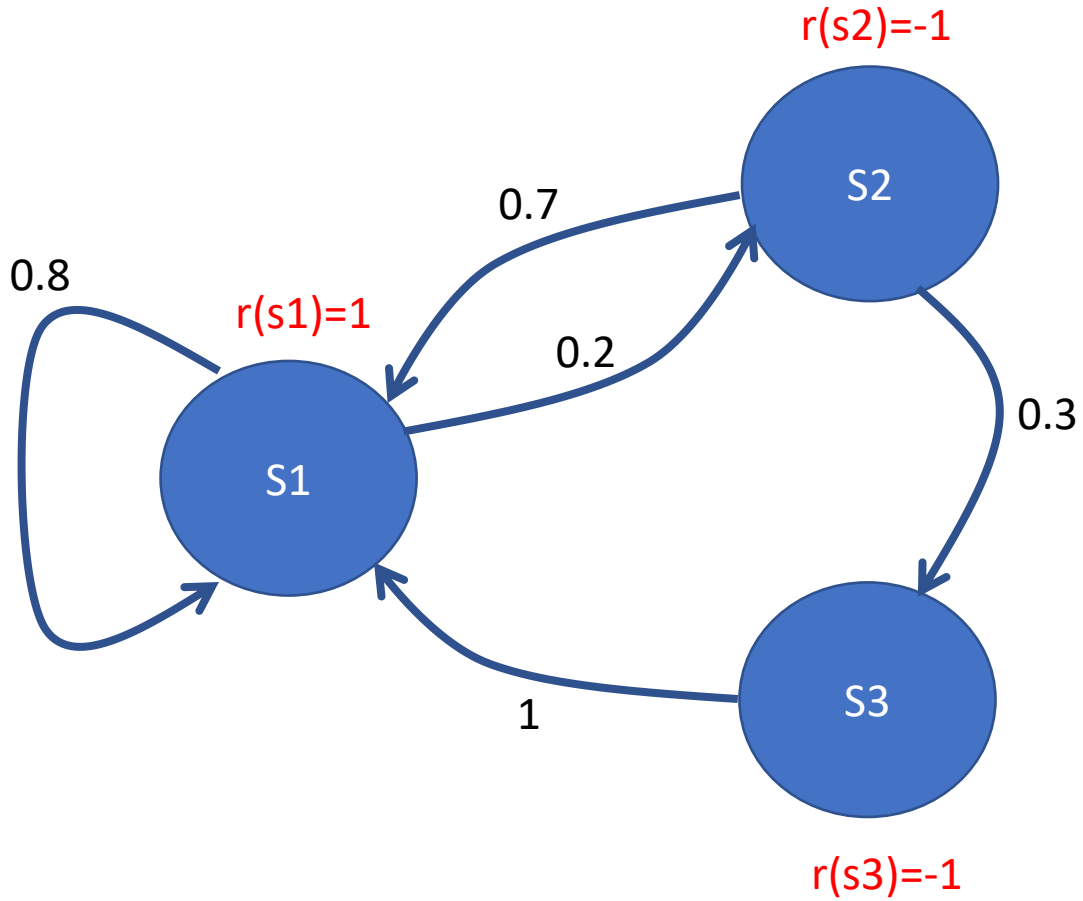
$$0.6x - 0.1y = 1$$

$$-0.35x + y - 0.15z = -1$$

$$-0.5x + z = -1$$

Equations with 3 variables!

Modeling Robot Motion with MRP



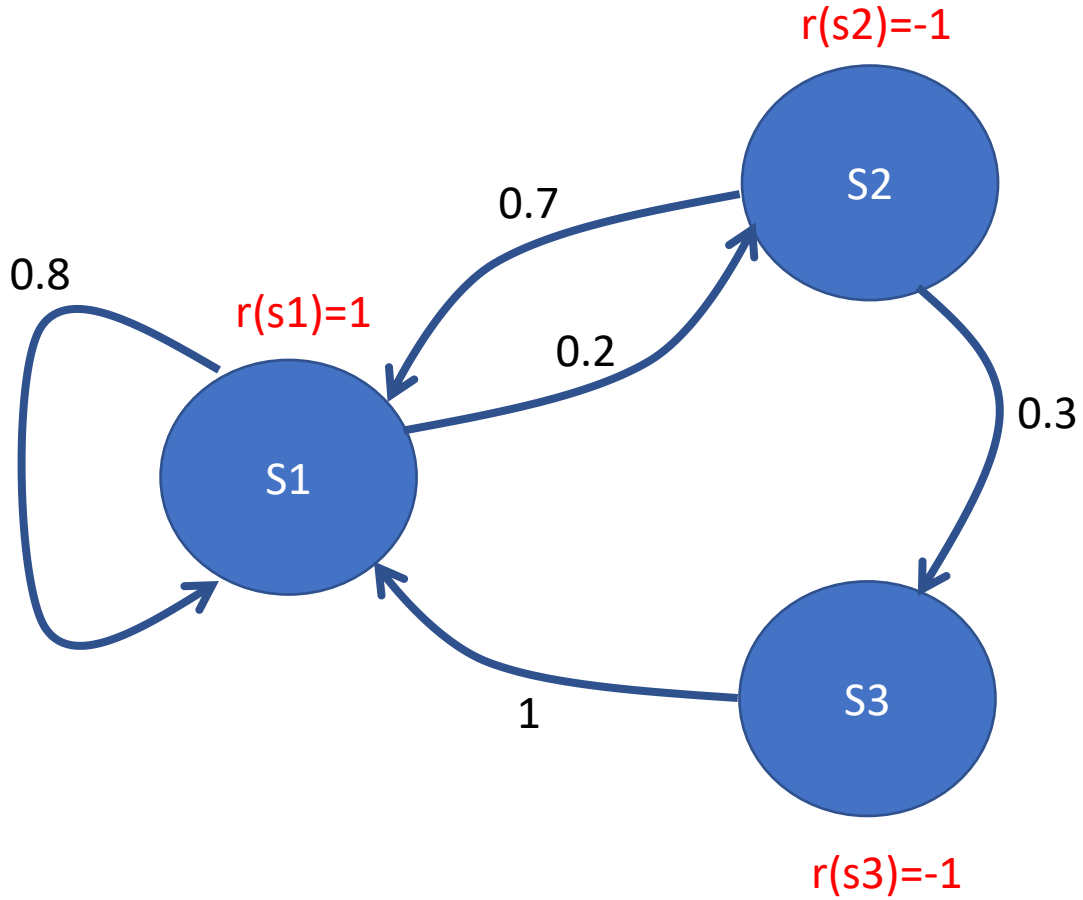
$$x = \frac{0.885}{0.5575}$$

$$y = \frac{-0.265}{0.5575}$$

$$z = \frac{-0.115}{0.5575}$$

Equations with 3 variables!

Modeling Robot Motion with MRP



$$\begin{bmatrix} v(s_1) \\ v(s_2) \\ v(s_3) \end{bmatrix} = \begin{bmatrix} r(s_1) \\ r(s_2) \\ r(s_3) \end{bmatrix} + \gamma \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0.7 & 0 & 0.3 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v(s_1) \\ v(s_2) \\ v(s_3) \end{bmatrix}$$

$$v = R + \gamma P v$$

Markov Decision Process

- 마르코프 의사 결정 프로세스(MDP)는 의사 결정 단계가 있는 마르코프 보상 프로세스
- (S, A, P, R, γ) is called a Markov decision process, where
  - S is a finite set of n states $\{s_1, s_2, \ldots, s_n\}$
  - A is a finite set of m actions $\{a_1, a_2, \ldots, a_m\}$
  - P is a transition probability matrix such that $p_{ij}^k = Pr(S_{t+1} = s_j | S_t = s_i, A_t = a_k)$
  - $r(s_i, a_k)$ is an expected reward at t+1 by taking a_k from s_i, i.e., $r(s_i, a_k) = E[R_{t+1} | S_t = s_i, A_t = a_k]$

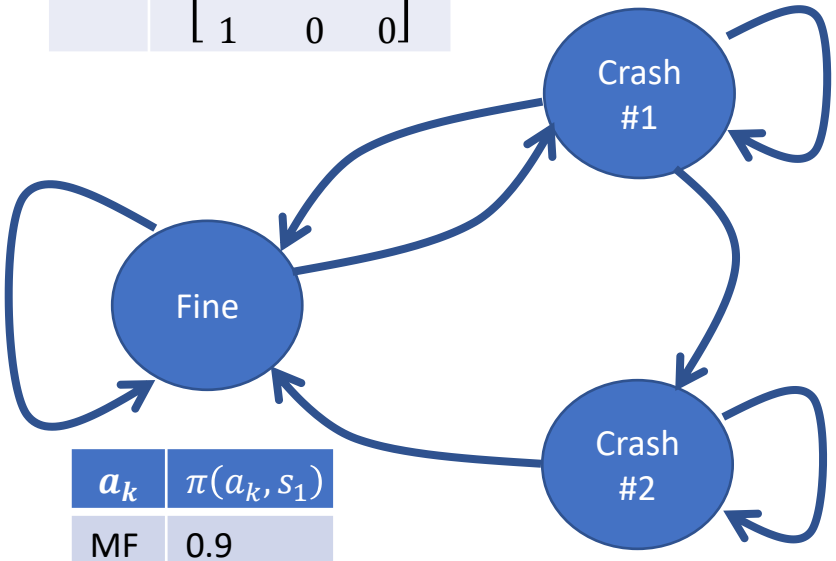
Note that the reward is for the step of t+1, not that of t

Modeling Robot Motion with MDP

- Let
  - A = { a_1='Move forward', a_2='Turn left' }
  - S = {s_1='Fine', s_2='Crash #1', s_2='Crash #2'}
- Then,
  - the probability of transition $s_1 \rightarrow s_2$ is

$$P_{12}^{\pi} = \sum_{k=1}^{m} \pi(a_k, s_1) p_{12}^{k}$$

a_k	P^{a_k}
MF	$\begin{bmatrix} 0.7 & 0.3 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
TL	$\begin{bmatrix} 0.9 & 0.1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$



a_k	$\pi(a_k, s_1)$
MF	0.9
TL	0.1

Policy

- A policy $\pi(a_k, s_i)$ is a probability of an action a_k given a state s_i

$$\pi(a_k, s_i) = \Pr(A_t = a_k | S_t = s_i)$$

- Given an MDP (S, A, P, R, γ) and a policy π,

$$p_{ij}^{\pi} = \sum_{k=1}^{m} \pi(a_k, s_i) p_{ij}^{k}$$

The probability of $s_1 \rightarrow s_2$ through all possible actions

$$r(s_i) = \sum_{k=1}^{m} \pi(a_k, s_i) r(s_i, a_k)$$

가능한 모든 행동을 통해 t+1 단계에서 예상되는 보상

The expected reward at step t+1 by the action a_k

Value Function

- The action-value function $q_\pi(s_i, a_k)$ is the expected return G_k (i.e., the discounted total rewards) starting from the state s_i by taking the action a_k following the policy π

$$q_\pi(s_i, a_k) = E[G_t | S_t = s_i, A_t = a_k]$$

- The state-value function $v_\pi(s_i)$ is then,

$$v_\pi(s_i) = E[G_t | S_t = s_i] = \sum_{k=1}^{m} \pi(a_k, s_i) q_\pi(s_i, a_k)$$

Bellman Equation

- Considering the policy π, Bellman equation for $q_\pi(s_i, a_k)$ in MDP can be obtained as

$$
\begin{aligned}
q_\pi(s_i, a_k) &= E_\pi[G_t | S_t = s_i, A_t = a_k] \\
&= E_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s_i, A_t = a_k] \\
&= E_\pi[R_{t+1}] + \gamma E_\pi[G_{t+1} | S_t = s_i, A_t = a_k] \\
&= r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^k E_\pi[G_{t+1} | S_{t+1} = s_j, A_t = a_k] \\
&= r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^k E_\pi[G_{t+1} | S_{t+1} = s_j] \\
&= r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^k v_\pi(s_j)
\end{aligned}
$$

Bellman Equations

Represent $v_\pi(\cdot)$ as an equation of $v_\pi(\cdot)$ and $q_\pi(\cdot)$ as an equation of $q_\pi(\cdot)$

- Considering the policy π, Bellman equation for $v_\pi(s_i)$ can be obtained as

$$\begin{aligned} v_\pi(s_i) &= E_\pi[G_t|S_t = s_i] \\ &= \sum_{k=1}^{m} \pi(a_k, s_i) q_\pi(s_i, a_k) \\ &= \sum_{k=1}^{m} \pi(a_k, s_i)\left(r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^k v_\pi(s_j)\right) \end{aligned}$$

- For $q_\pi(s_i, a_k)$,

$$\begin{aligned} q_\pi(s_i, a_k) &= r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^k v_\pi(s_j) \\ &= r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^k \sum_{\ell=1}^{m} \pi(a_\ell, s_j) q_\pi(s_j, a_\ell) \end{aligned}$$

Bellman Equations in Matrix Form

$$v_\pi(s_i) = \sum_{k=1}^{m} \pi(a_k, s_i)\left(r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^{k} v_\pi(s_j)\right)$$

$$v_\pi(s_i) = \underbrace{[\pi(a_1, s_i) \quad \dots \quad \pi(a_m, s_i)]}_{\pi(s_i)} \left(\underbrace{\begin{bmatrix} r(s_i, a_1) \\ \vdots \\ r(s_i, a_m) \end{bmatrix}}_{R(s_i)} + \gamma \underbrace{\begin{bmatrix} p_{i1}^{1} & \cdots & p_{in}^{1} \\ \vdots & \ddots & \vdots \\ p_{i1}^{m} & \cdots & p_{in}^{m} \end{bmatrix}}_{P_i} \underbrace{\begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix}}_{v_\pi} \right)$$

$$v_\pi(\mathrm{s_i}) = \pi(s_i) \cdot (R(s_i) + \gamma P_i \cdot v_\pi)$$

Bellman Equations in Tensor Form

$v_\pi, (\mathrm{n},)$ $\pi, (n, m)$ $R_\pi, (n, m)$ $P, (n, m, n)$ $v_\pi, (\mathrm{n},)$

$$v_\pi(s_1) = [\pi(a_1, s_1) \quad \dots \quad \pi(a_m, s_1)] \left(\begin{bmatrix} r(s_1, a_1) \\ \vdots \\ r(s_1, a_m) \end{bmatrix} + \gamma \begin{bmatrix} p_{11}^1 & \cdots & p_{1n}^1 \\ \vdots & \ddots & \vdots \\ p_{11}^m & \cdots & p_{1n}^m \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right)$$

$$v_\pi(s_i) = [\pi(a_1, s_i) \quad \dots \quad \pi(a_m, s_i)] \left(\begin{bmatrix} r(s_i, a_1) \\ \vdots \\ r(s_i, a_m) \end{bmatrix} + \gamma \begin{bmatrix} p_{i1}^1 & \cdots & p_{in}^1 \\ \vdots & \ddots & \vdots \\ p_{i1}^m & \cdots & p_{in}^m \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right)$$

$$v_\pi(s_n) = [\pi(a_1, s_n) \quad \dots \quad \pi(a_m, s_n)] \left(\begin{bmatrix} r(s_n, a_1) \\ \vdots \\ r(s_n, a_m) \end{bmatrix} + \gamma \begin{bmatrix} p_{n1}^1 & \cdots & p_{nn}^1 \\ \vdots & \ddots & \vdots \\ p_{n1}^m & \cdots & p_{nn}^m \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right)$$

$$v_\pi = \pi \odot (R + \gamma P \cdot v_\pi)$$

* $\odot$: element-wise dot product

Solving Bellman Optimal Equation

- Bellman Optimality Equation 은 보통 비선형 (식 안에 max가 있음!)
  - 닫힌 형식의 해 (closed-form solution) 가 없음
- 반복알고리즘 (iterative solution methods)
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

Iterative Policy Evaluation

- 주어진 정책 π을 평가합니다. 즉, 얼마나 좋은 지 계산합니다.
- Algorithm
  - k ← 1
  - Initialize $v^{(0)}(s_i) = 0$
  - While $v^{(k)}(s_i)$ converges
    - For all states $s_i \in S$
      - $v^{(k+1)}(s_i) = \sum_{\ell=1}^{m} \pi(a_\ell, s_i)\left(r(s_i, a_\ell) + \gamma \sum_{j=1}^{n} p_{ij}^{\ell} v^{(k)}(s_j)\right)$

Jacobi method를 이용해 해를 구함!

(Greedy) Policy Improvement

- 최대 수익률의 작업 (즉, $\underset{a \in A}{\operatorname{argmax}}\, q_{\pi(k)}(s_i, a)$)을 선택하여 정책을 업데이트
- That is,

$$r(s_i, a) + \gamma \sum_{j=1}^{n} p_{ij}^{a} v_{\pi^{(k)}}(s_j)$$

$$\forall s_i \in S, \pi^{(k+1)}(a_\ell, s_i) = \begin{cases} 1 & if\ a = \underset{a \in A}{arg\max}\, q_{\pi^{(k)}}(s_i, a) \\ 0 & otherwise \end{cases}$$

- Algorithm
  - Repeat until $\pi^{(k+1)}(\cdot, s_i) = \pi^{(k)}(\cdot, s_i), \forall s_i \in S$
    - For each $s_i \in S$
      - $\pi^{(k+1)}(\cdot, s_i) = OneHot\left(m, \arg\max_{a \in A} r(s_i, a) + \gamma \sum_{j=1}^{n} p_{ij}^{a} v_{\pi^{(k)}}(s_j)\right)$

Iterative Policy Optimization

- Algorithm
  - k ← 1
  - Initialize $\pi^{(1)}(a_k, s_i)$ randomly to hold $\sum_{k=1}^{m} \pi^{(1)}(a_k, s_i)$, $\forall s_i \in S$
  - Repeat
    - Compute $v^{(k)}(s_i), \forall s_i \in S$ by policy evaluation
    - Update $\pi^{(k+1)}(a_k, s_i)$, $\forall s_i \in S, a_i \in A$ by policy improvement
    - If $\pi^{(k+1)} = \pi^{(k)}$, stop

Recall: Keywords

- Markov process ★
- Markov decision process ★★★
- Return ★★★★★
- Policy ★★★★★
- Value function ★★★★★
- Bellman equation ★★
- Policy Evaluation ★
- Policy Improvement ★

Example: Pathfinding In Gridworld

- 문제
  - 터미널이 아닌 셀에서 시작
  - 정책에 따라 회색으로 음영 처리된 터미널 셀 중 하나에 도달합니다.
  - Actions = {up, down, left, right}
  - 보상은 터미널을 제외한 모든 상태에 대해 -1
- Initially,
  - $\pi(s_i) = \{0.25, 0.25, 0.25, 0.25\}$ for all states s_i (i.e., 14 cells}

15	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

기호 ✥ 를 사용하여 모든 방향에 대한 균일한 분포를 나타냄

State Transition

The actions:
1: up, 2: down,
3: left, 4: right

액션으로 인해 에이전트가 그리드에서 벗어날 수 있는 경우, 상태는 변경되지 않습니다.

상태 전환은 결정론적입니다. 즉, 상태는 동작에 따라 정확하게 이동합니다. 예를 들어, 7에서 동작 1 (위)은 7→ 3만 변경합니다.

s_i

15		2	3
4	5	6	7
8	9	10	11
12	13	14	15

$$\{p_{ij}^{1}\} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array}$$

Expected Reward

$r(s_1,*) = \{-1,-1,1,-1\}$

15	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$$\{r(s_i, a_k)\}^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \vdots \end{array} \begin{array}{c} \text{1: up,} \quad \text{2: down,} \quad \text{3: left,} \quad \text{4: right} \\ \begin{Bmatrix} -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ \vdots & \vdots & \vdots & \vdots \end{Bmatrix} \end{array}$$

Initial Gridworld

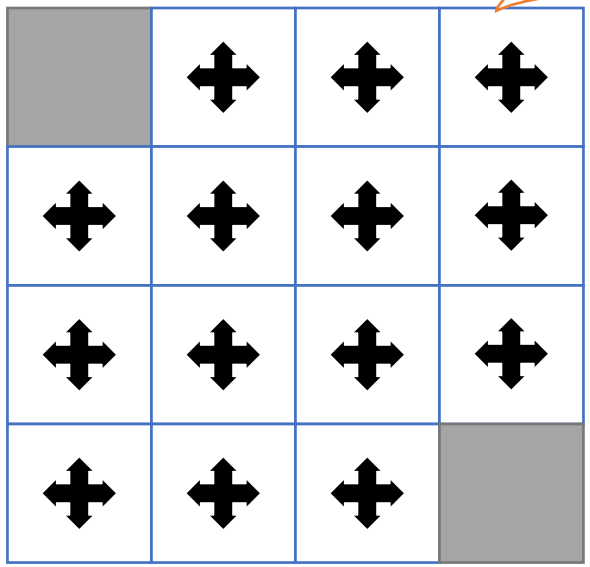
$v_1(s_i)$

$v_\pi(s_i) = 0$

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

$\pi_1(s_i)$

$\pi(s_i) = \{0.25, 0.25, 0.25, 0.25\}$



Policy Evaluation: 1st Round

Set $\gamma = 1$

$v_1(s_i)$

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

$$v_\pi(s_i) = \sum_{k=1}^{m} \pi(a_k, s_i)\left(r(s_i, a_k) + \gamma \sum_{j=1}^{n} p_{ij}^{k} v_\pi(s_j)\right)$$

$$v_\pi(s_1) = 0.5(-1 + v_\pi(s_1)) + 0.5(-1 + v_\pi(s_5)) + 0.5(1 + v_\pi(s_{15})) + 0.5(-1 + v_\pi(s_2))$$

$$v_\pi(s_2) = 0.5(-1 + v_\pi(s_2)) + 0.5(-1 + v_\pi(s_6)) + 0.5(-1 + v_\pi(s_1)) + 0.5(-1 + v_\pi(s_3))$$

$$v_\pi(s_3) = 0.5(-1 + v_\pi(s_3)) + 0.5(-1 + v_\pi(s_7)) + 0.5(-1 + v_\pi(s_2)) + 0.5(-1 + v_\pi(s_3))$$

$$v_\pi(s_4) = 0.5(1 + v_\pi(s_{15})) + 0.5(-1 + v_\pi(s_8)) + 0.5(-1 + v_\pi(s_4)) + 0.5(-1 + v_\pi(s_5))$$

$$\vdots$$

Numpy

- Transition matrix

```
import numpy as np

P = np.array([
# up
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 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, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]],
# down
```

```
# down
[[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]],
# left
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]],
```

```
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]],

# right
[[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]], dtype='float32')

P.shape
```

Recall The Bellman Equations in Tensor Form

$v_\pi, (\mathrm{n},)$ $\pi, (n, m)$ $R_\pi, (n, m)$ $P, (n, m, n)$ $v_\pi, (\mathrm{n},)$

$$v_\pi(s_1) = [\pi(a_1, s_1) \quad \dots \quad \pi(a_m, s_1)] \left(\begin{bmatrix} r(s_1, a_1) \\ \vdots \\ r(s_1, a_m) \end{bmatrix} + \gamma \begin{bmatrix} p_{11}^1 & \cdots & p_{1n}^1 \\ \vdots & \ddots & \vdots \\ p_{11}^m & \cdots & p_{1n}^m \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right)$$

$$v_\pi(s_i) = [\pi(a_1, s_i) \quad \dots \quad \pi(a_m, s_i)] \left(\begin{bmatrix} r(s_i, a_1) \\ \vdots \\ r(s_i, a_m) \end{bmatrix} + \gamma \begin{bmatrix} p_{i1}^1 & \cdots & p_{in}^1 \\ \vdots & \ddots & \vdots \\ p_{i1}^m & \cdots & p_{in}^m \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right)$$

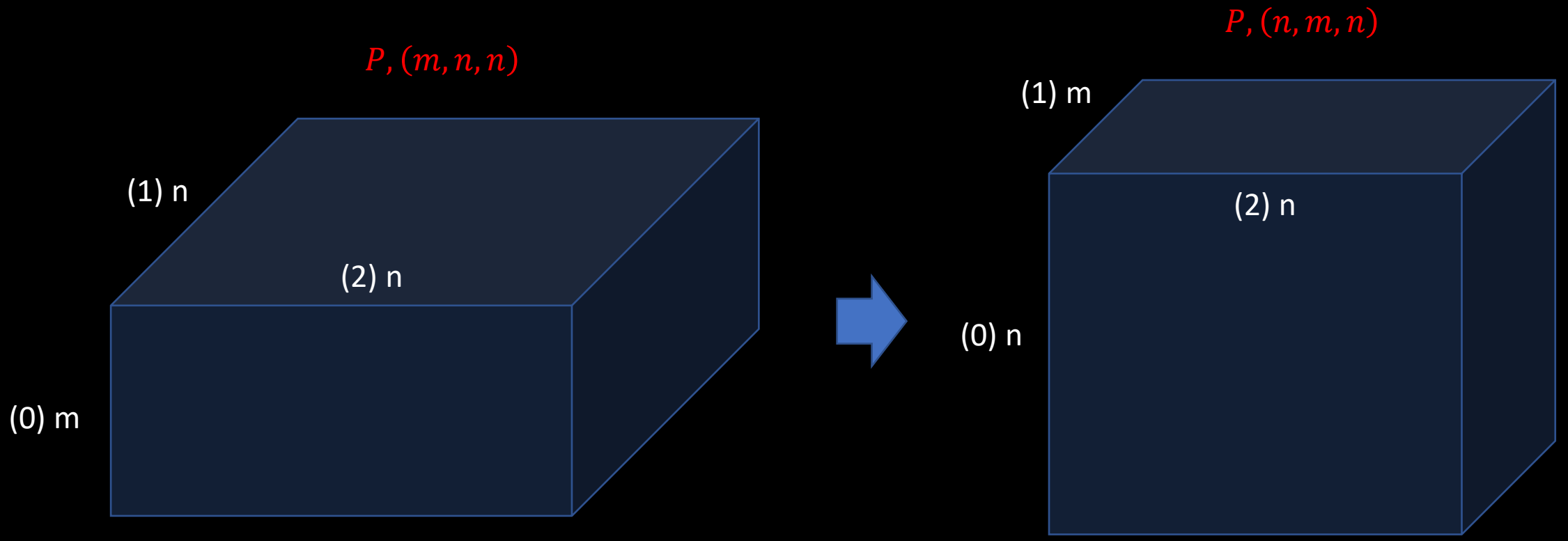
$$v_\pi(s_n) = [\pi(a_1, s_n) \quad \dots \quad \pi(a_m, s_n)] \left(\begin{bmatrix} r(s_n, a_1) \\ \vdots \\ r(s_n, a_m) \end{bmatrix} + \gamma \begin{bmatrix} p_{n1}^1 & \cdots & p_{nn}^1 \\ \vdots & \ddots & \vdots \\ p_{n1}^m & \cdots & p_{nn}^m \end{bmatrix} \begin{bmatrix} v_\pi(s_1) \\ \vdots \\ v_\pi(s_n) \end{bmatrix} \right)$$

$$v_\pi = \pi \odot (R + \gamma P \cdot v_\pi)$$

* $\odot$: element-wise dot product

Transpose

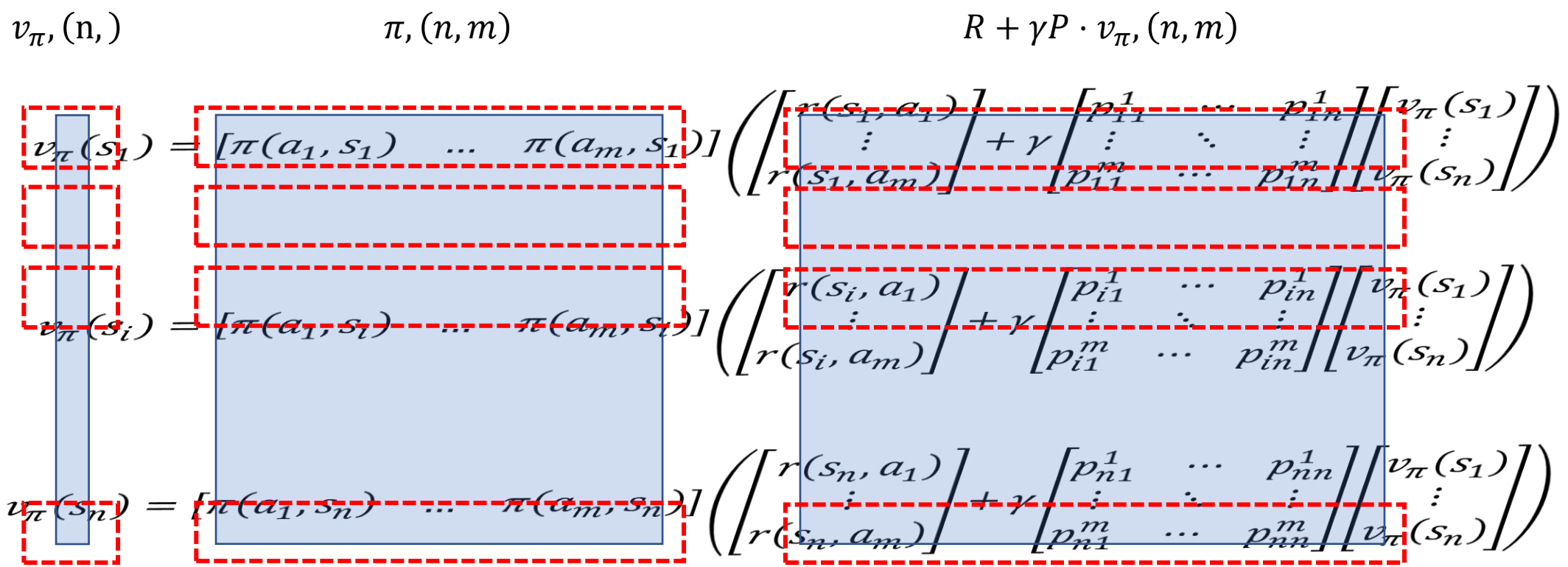
```
P = P.transpose(1, 0, 2)
P.shape
```



Reward & Initializing π and V

```
R = np.array([
  [-1, -1, -1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0], # up
  [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, -1, -1, -1, 0], # down
  [ 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0], # left
  [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 1, 0]], #right
  dtype='float32')

R = R.transpose()

pi = np.ones((15, 4), dtype='float32') * 0.25

V = np.zeros((30, 15), dtype='float32')
```

Recall The Bellman Equations in Tensor Form

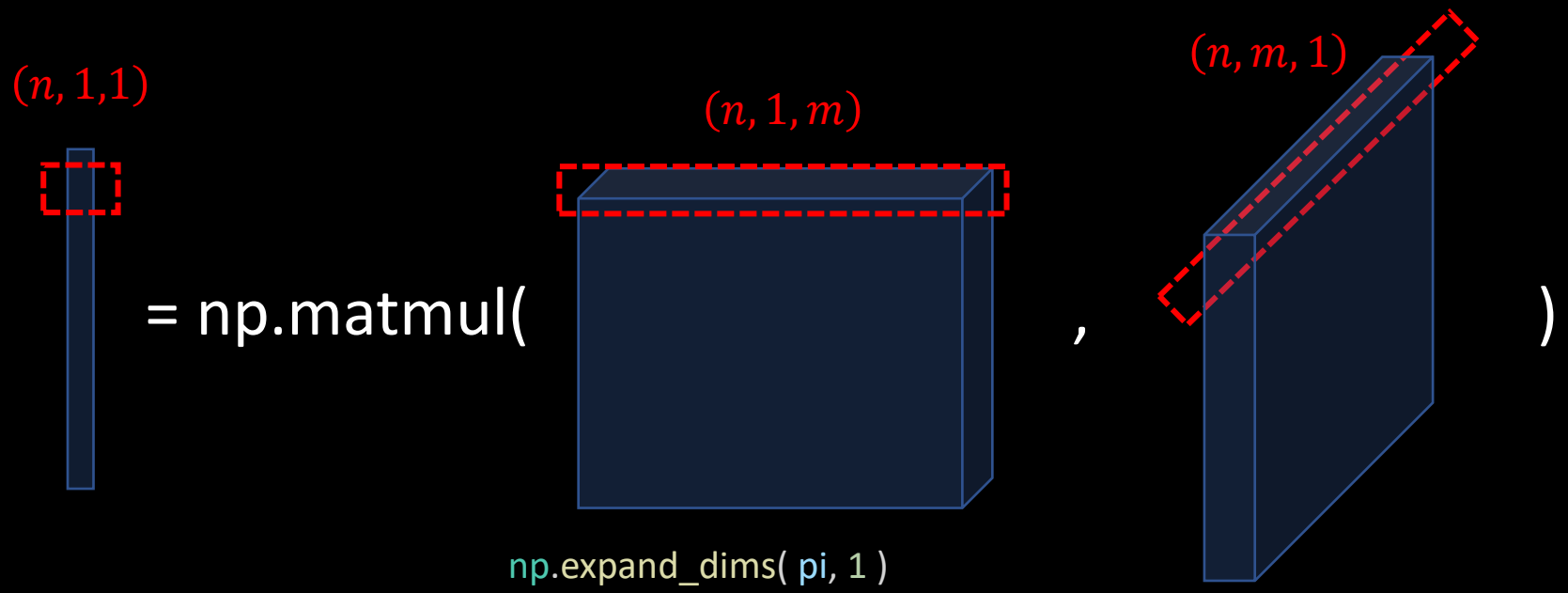


$$v_\pi = \pi \odot (R + \gamma P \cdot v_\pi)$$

* ⊙: element-wise dot product

Iterative Policy Evaluation

```
for i in range(maxIter-1):
  V[i+1] = np.squeeze(
    np.matmul(
      np.expand_dims( pi, 1 ),
      np.expand_dims( R + 0.6 * np.dot(P, V[i]), 2 )))
```



Policy Improvement: 1st Round

- Recall
  - $\forall s_i \in S, \pi^{(k+1)}(\cdot, s_i) = OneHot\left(m, \arg\max_{a \in A} r(s_i, a) + \gamma \sum_{j=1}^{n} p_{ij}^{a} v_{\pi^{(k)}}(s_j)\right)$

np.squeeze(np.expand_dims(R + 0.6 * np.dot(P, V[i]), 2))

```
a_idx = np.argmax(
    np.squeeze(np.expand_dims( R + 0.6 * np.dot(P, v), 2 )), axis=1)
pi = np.zeros((15, 4), dtype='float32’)
pi[range(15), a_idx] = 1.
```

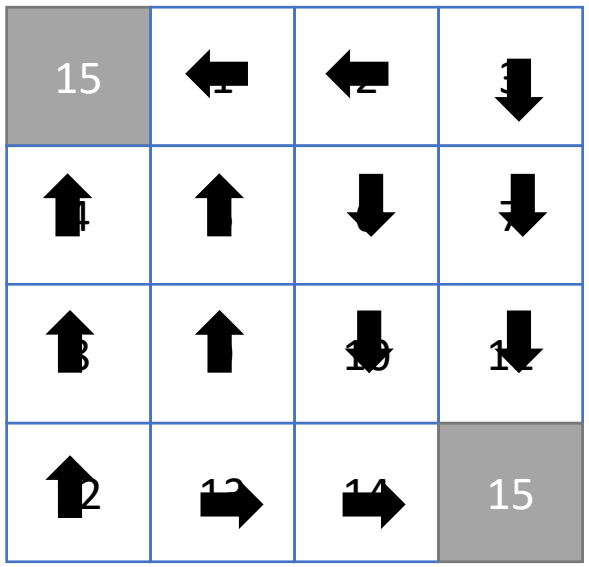
In Summary,

```
def policy_eval(P, R, pi, maxiter=30):
  V = np.zeros((maxiter, 15), dtype='float32’)
  for i in range(maxiter-1):
    V[i+1] = np.squeeze(
      np.matmul(
        np.expand_dims( pi, 1 ),
        np.expand_dims( R + 0.6 * np.dot(P, V[i]), 2 )))
  return V[maxiter-1]

def policy_upd(P, R, v):
  a_idx = np.argmax(np.squeeze(np.expand_dims( R + 0.6 * np.dot(P, V[i]), 2 )), axis=1)
  pi = np.zeros((15, 4), dtype='float32’)
  pi[range(15), a_idx] = 1.
  return pi

pi_old = None
pi = np.ones((15, 4), dtype='float32') * 0.25

while not np.all(np.equal(pi_old, pi)):
  pi_old = pi.copy()
  v = policy_eval(P, R, pi)
  pi = policy_upd(P, R, v)
print(pi)
```

Policy Improvement: After 1st Round

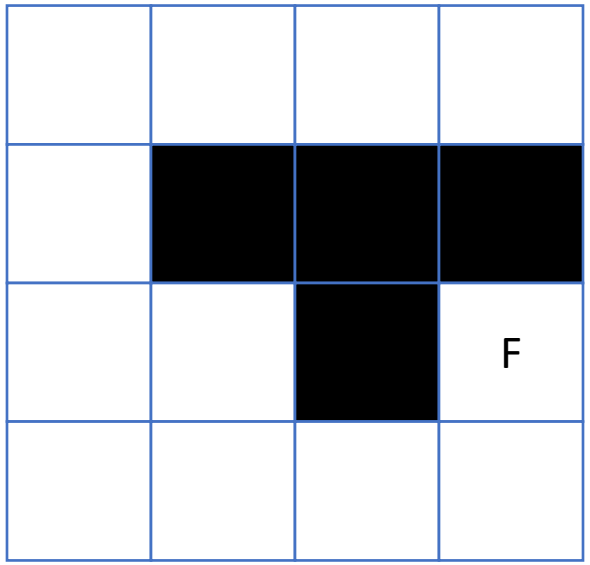
```
array([
  [0., 0., 1., 0.],
  [0., 0., 1., 0.],
  [0., 1., 0., 0.],
  [1., 0., 0., 0.],
  [1., 0., 0., 0.],
  [0., 1., 0., 0.],
  [0., 1., 0., 0.],
  [1., 0., 0., 0.],
  [1., 0., 0., 0.],
  [0., 1., 0., 0.],
  [0., 1., 0., 0.],
  [1., 0., 0., 0.],
  [0., 0., 0., 1.],
  [0., 0., 0., 1.],
  [1., 0., 0., 0.]], dtype=float32)
```

The actions:
1: up, 2: down, 3: left, 4: right



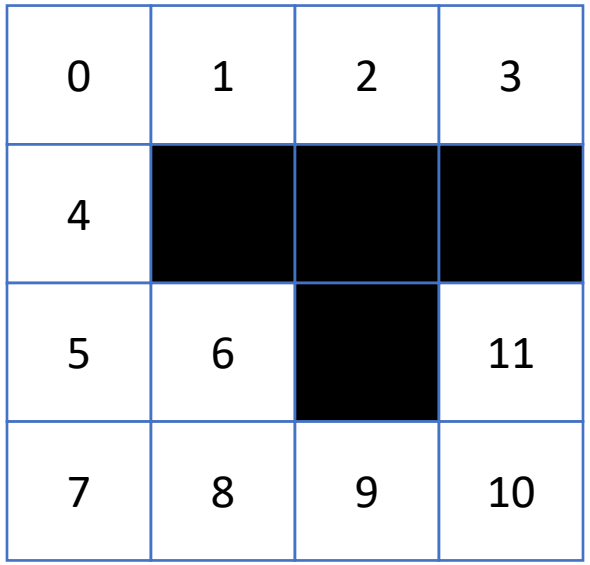
Quiz: Path Finding

- 문제
  - F에 도착할 수 있는 정책을 계산하시오
  - Actions = {up, down, left, right}



Quiz: Path Finding

- 문제
  - F에 도착할 수 있는 정책을 계산하시오
  - Actions = {up, down, left, right}



Policy Improvement: After 1st Round

The actions:
1: up, 2: down, 3: left, 4: right

```
array([
  [[0. 1. 0. 0.]
   [0. 0. 1. 0.]
   [0. 0. 1. 0.]
   [0. 0. 1. 0.]
   [0. 1. 0. 0.]
   [0. 1. 0. 0.]
   [0. 1. 0. 0.]
   [0. 0. 0. 1.]
   [0. 0. 0. 1.]
   [0. 0. 0. 1.]
   [1. 0. 0. 0.]
   [1. 0. 0. 0.]], dtype=float32)
```