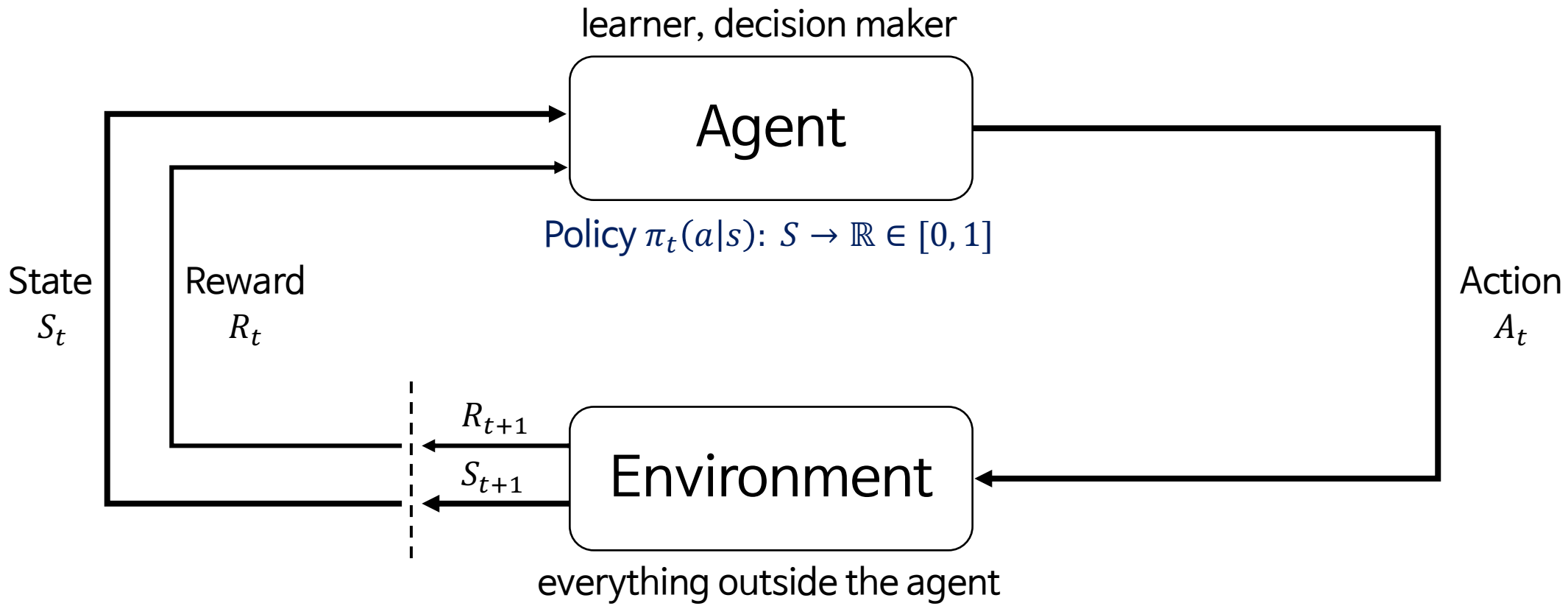
Introduction of Reinforcement Learning and Its Application for Autonomous UAV Control

아주대학교 국방디지털융합학과
201621116 강준하

목차

Reinforcement Learning?

- 상호 작용을 통해 목표를 달성하는 방법을 배우는 문제



Markov Decision Process

: tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$: set of states
- $\mathcal{A}$: set of actions
- $\mathcal{P}$: state transition probability matrix
- $\mathcal{R}$: reward function
- γ : discount factor

Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

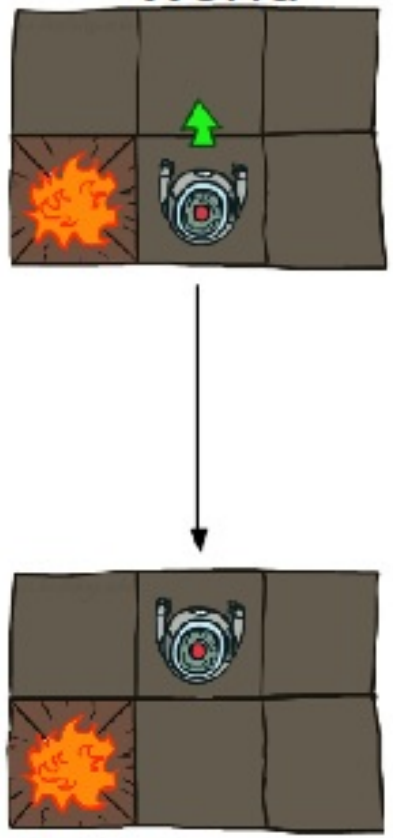
A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

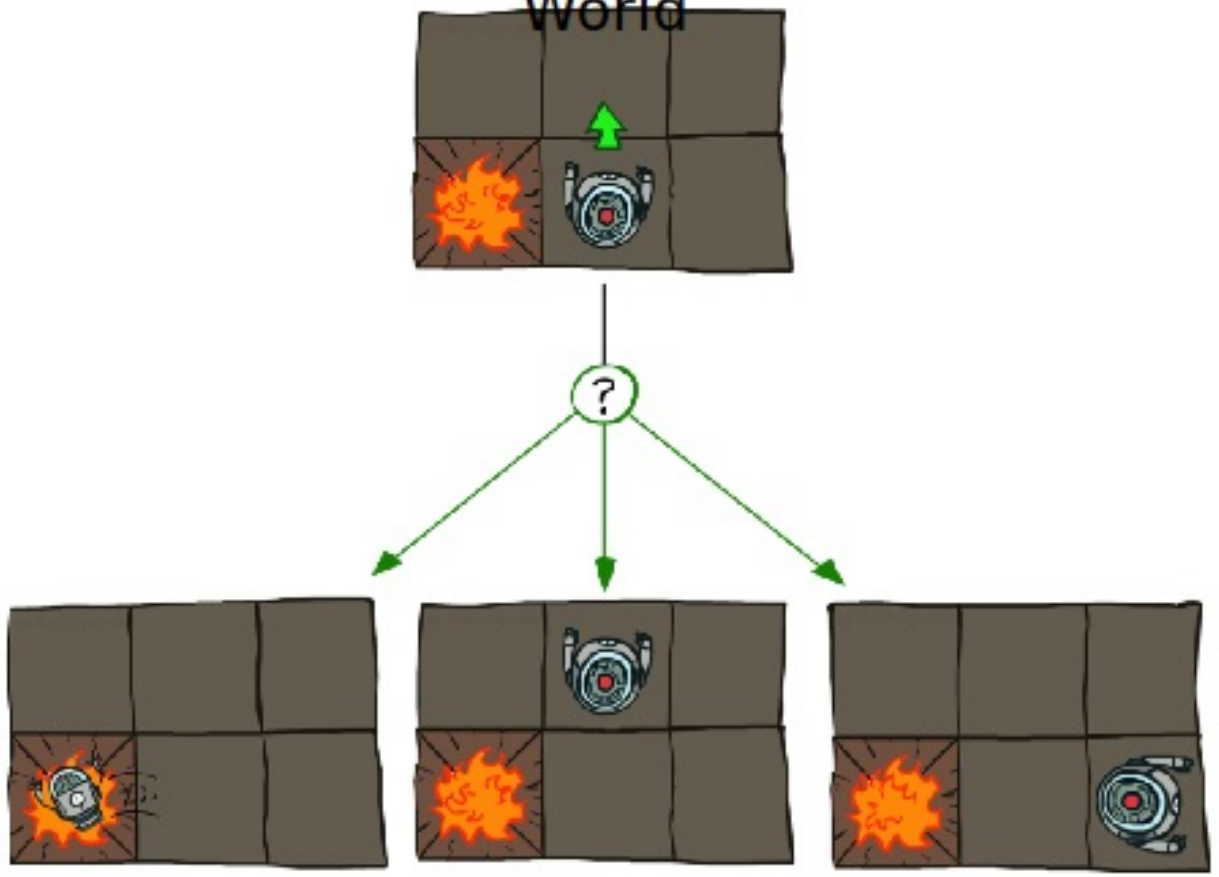
State transition probability matrix

$$\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

Deterministic Grid World



Stochastic Grid World

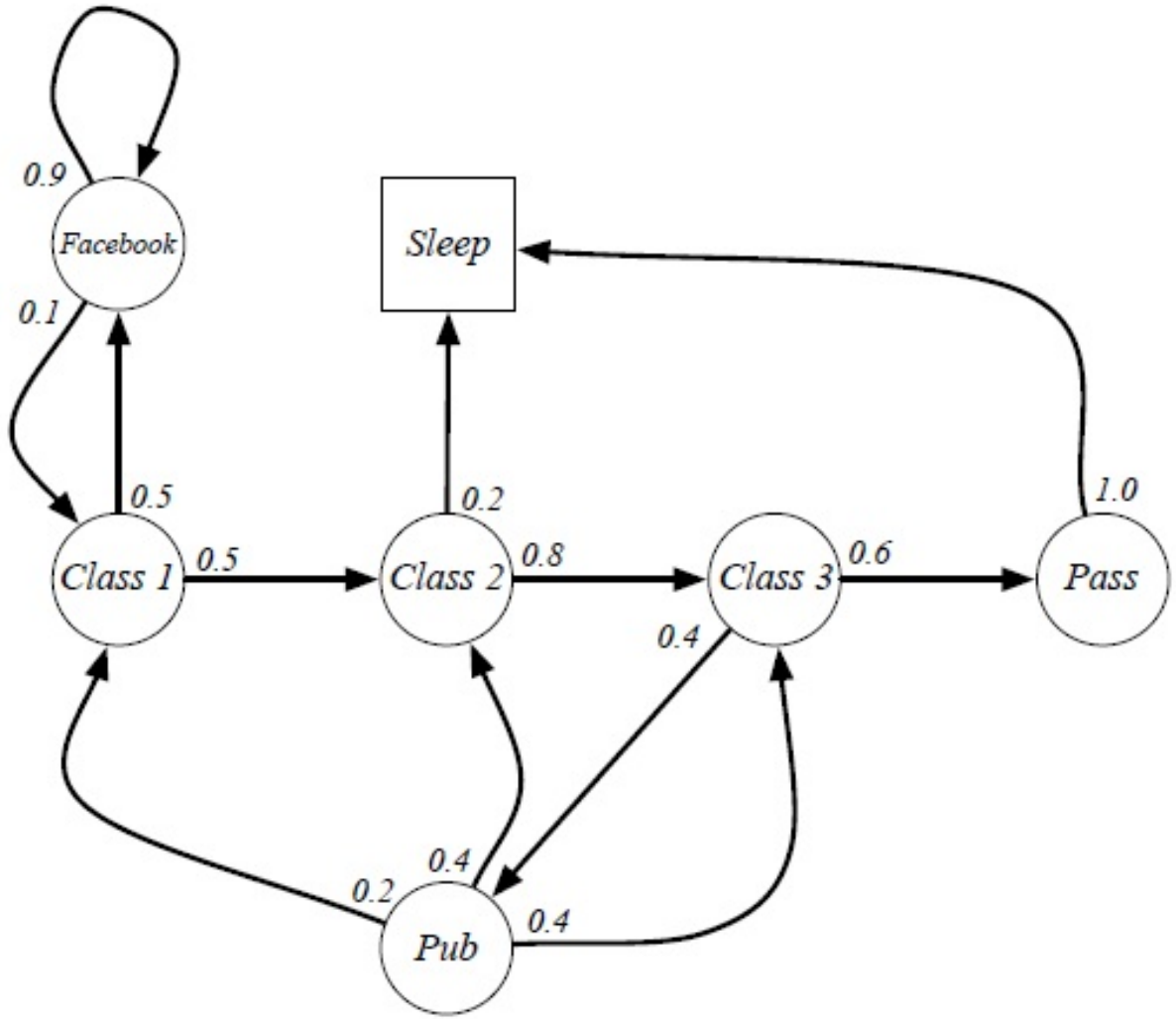


Markov Chain? Markov Property?

- Memoryless!
- Future only depends on present
- Current state is a sufficient statistic of agent's history
- No need to remember agent's history

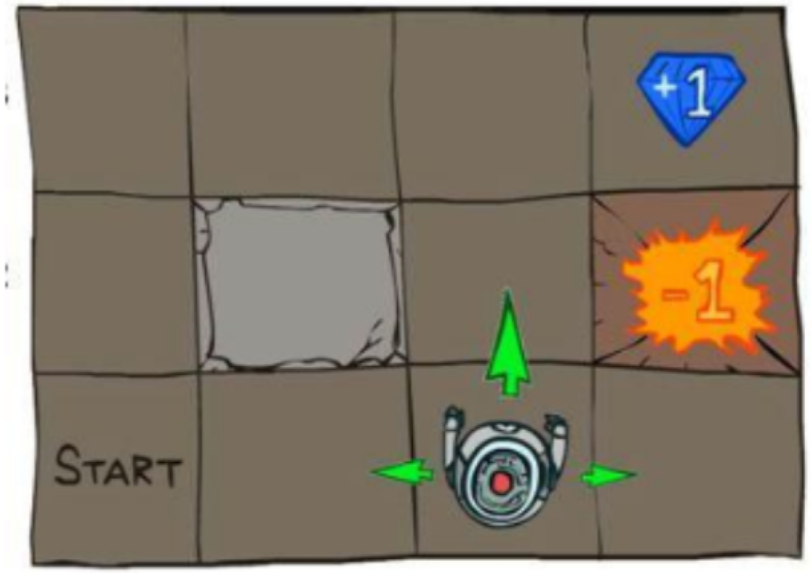
- 일단은 이거만 알고 가도록 하자...

Example: Student Markov Chain



Markov Decision Process (cont.)

- A sequence of discrete *time steps*
  - $t = 0, 1, 2, 3, \cdots$
- Environment's *state*
  - $S_t \in \mathcal{S}$
- Agent's *action*
  - $A_t \in \mathcal{A}(s)$
- One time step later, in part as a consequence of its action, the agent receives a numerical *reward*
  - $R_t \in \mathcal{R} \subset \mathbb{R}$
- And finds itself in a new state, S_{t+1}.



Markov Decision Process (cont.)

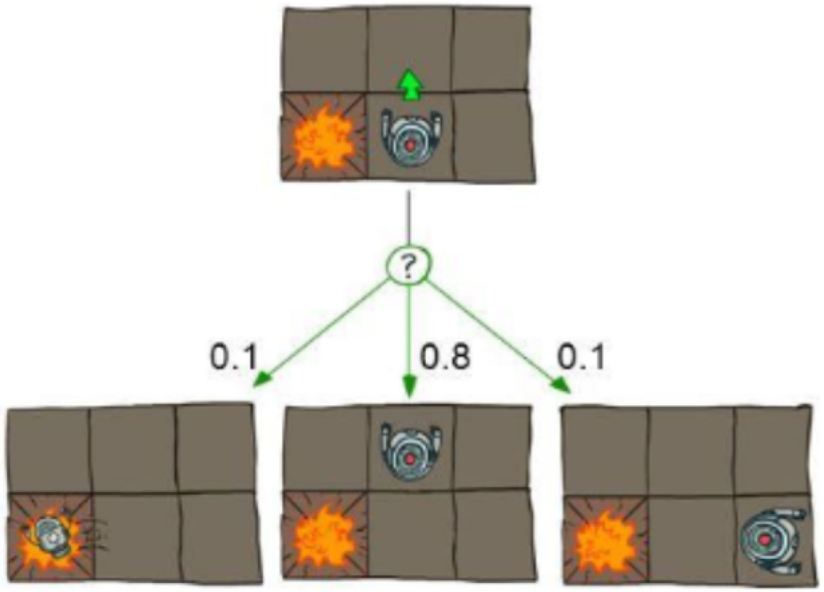
- $(S_0, A_0), (R_1, S_1, A_1), (R_2, S_2, A_2), (R_3, S_3, \cdots$
- R_t, S_t : the random variables
- For particular values $s' \in \mathcal{S}, r \in \mathcal{R}$ …
  - $p(s', r|s, a) \coloneqq \Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$
  - $\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$.
- The function p : $\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is an ordinary deterministic function of four arguments…

Markov Decision Process (cont.)

- Then, we can also make state-transition probabilities function $p: \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0,1]$ (compute anything else one might want to know about the environment):
  - $p(s'|s,a) \coloneqq \Pr\{S_t = s' | S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$
- We can also compute the expected rewards for state-action pairs as a two-argument function $r: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:
  - $r(s,a) \coloneqq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$

Markov Decision Process (cont.)

- Or the expected rewards for state-action-next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:
  - $r(s, a, s') := \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}$



Markov Decision Process (cont.)

1

Worth Now

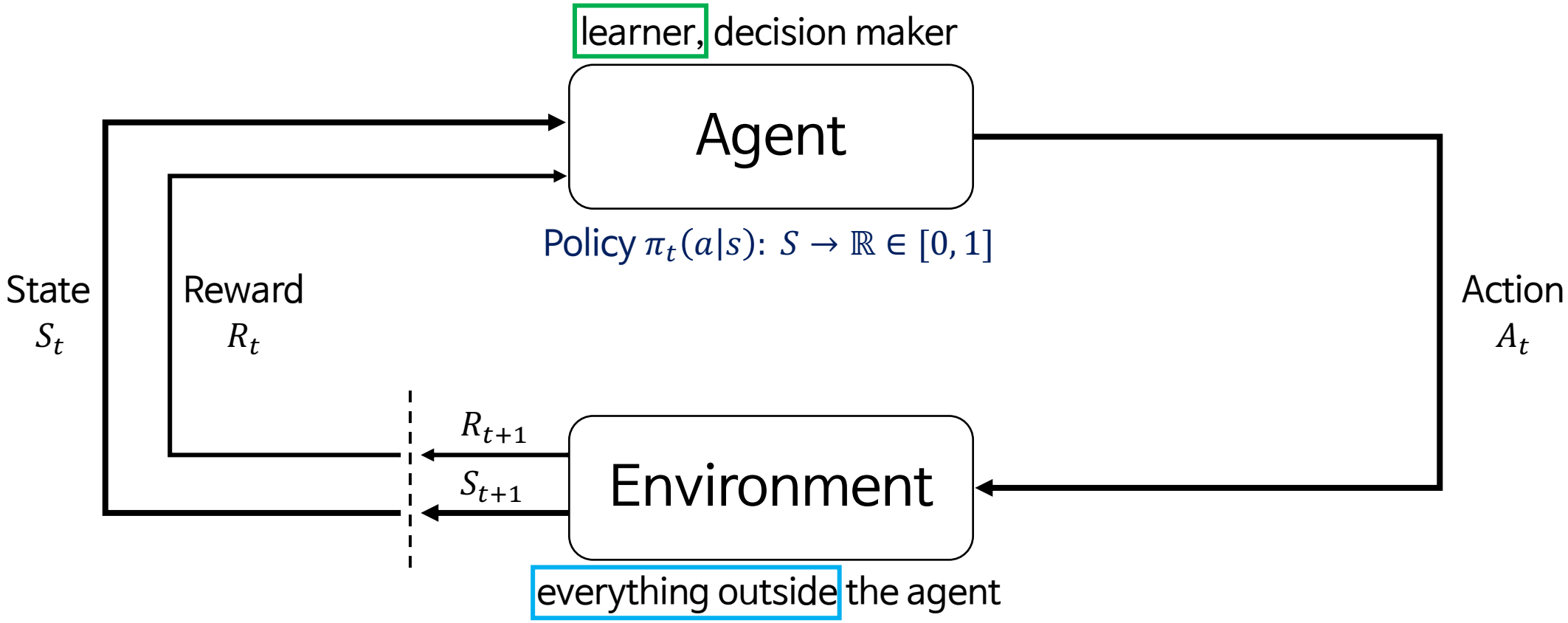
γ

Worth Next Step

γ^2

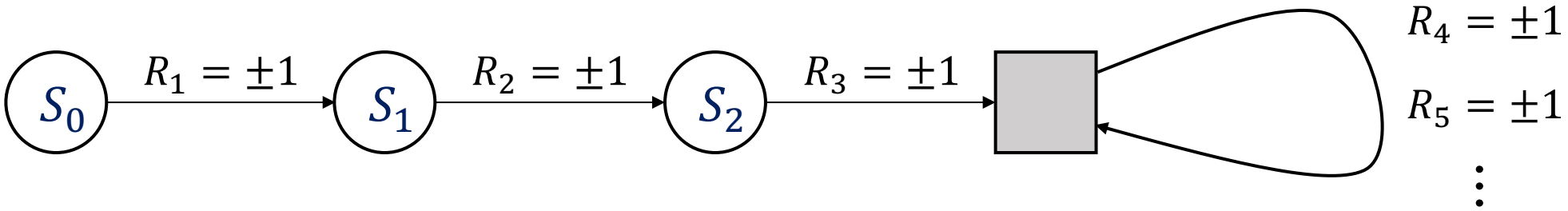
Worth In Two Steps

Agent–Environment Interface



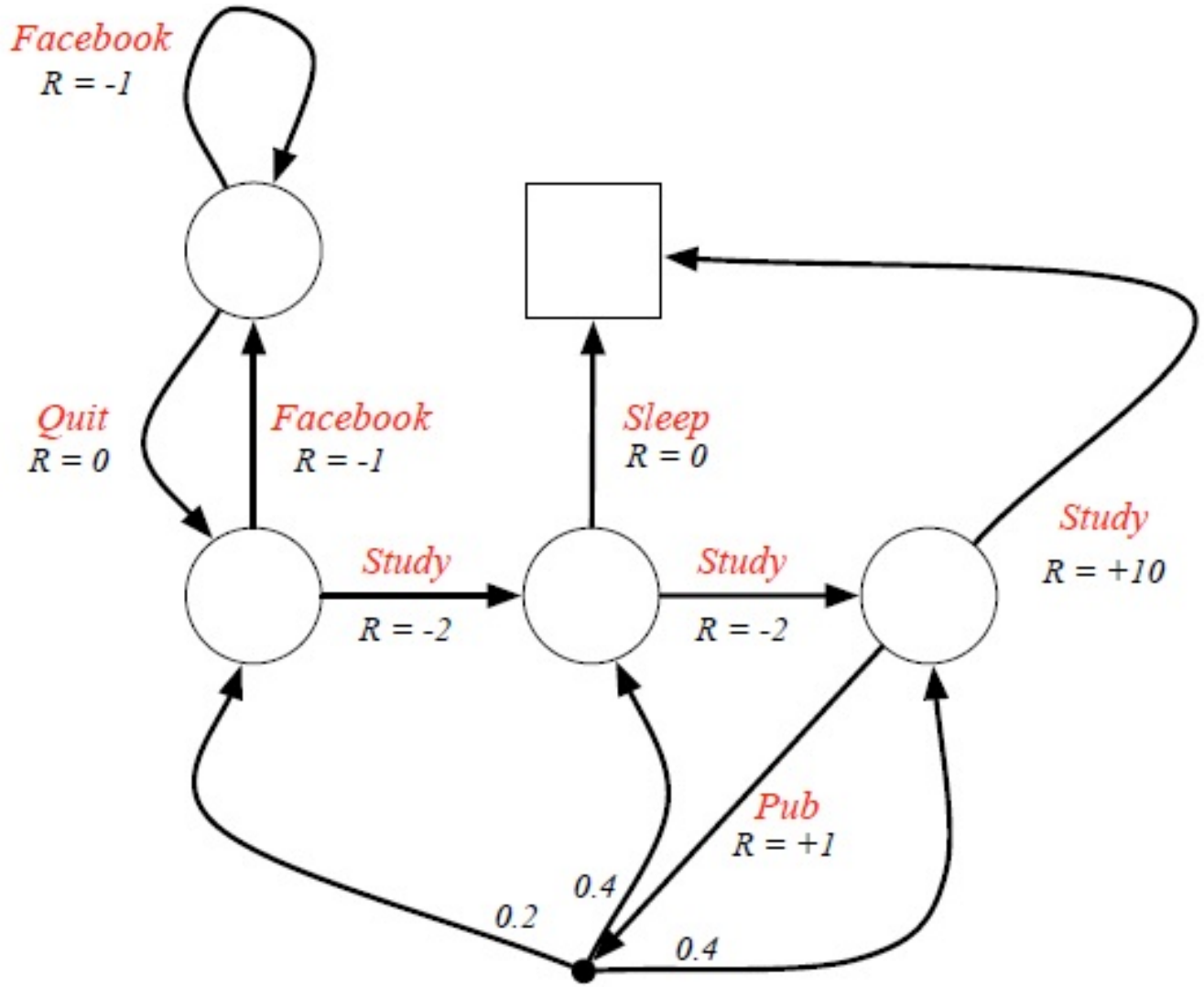
Agent-Environment Interface (cont.)

- Agent-environment interaction break down into
  - Sequence of separate episodes (episodic tasks)
    - $S_0, S_1, \cdots, S_T. R_t = 0 \; where \; t > T$



- ~~Or just one (continuing tasks)~~

Example : Student MDP Graph



Policy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

Return

Definition

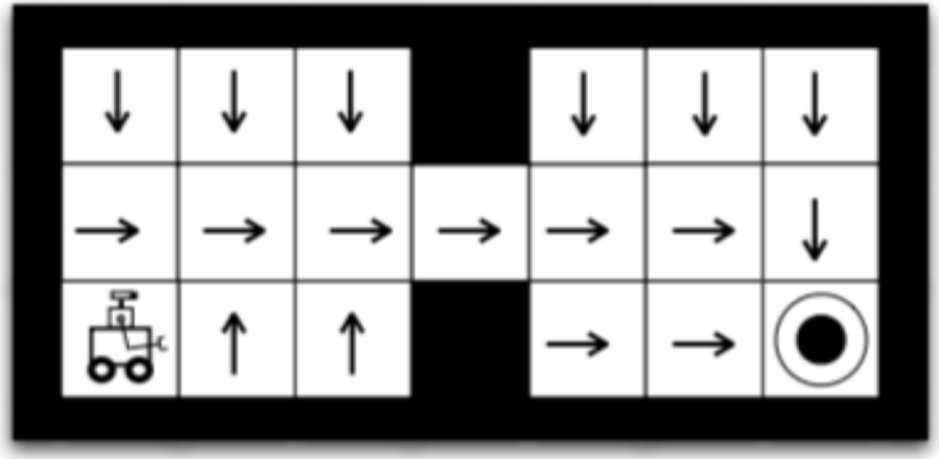
The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Policy and Return (summarize)

Policy

- $\pi: \mathcal{S} \rightarrow \mathcal{A}$
- Maps states to actions
- Gives an action for every state



Return

- Discounted sum of rewards
- Could be undiscounted, finite horizon

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

Our Goal?

Find π that maximizes expected return!

From Student MDP Graph Example

Sample returns for Student MRP:
Starting from $S_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	=	-3.41
C1 FB FB C1 C2 C3 Pub C1 ... FB FB FB C1 C2 C3 Pub C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	=	-3.20

Value Function

- Estimate ***how good*** it is for the agent to be in a given state
  - $\boldsymbol{V}: \boldsymbol{S} \rightarrow \mathbb{R}$
  - "*how good*" is defined in terms of **future rewards** that can be expected
    - 미래의 reward는 어떤 action을 취할지에 따라 달라진다(π).
- $v_\pi(s) = \mathbb{E}_\pi[R_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{T-t} \gamma^k r_{t+k+1} | S_t = s]$
  - where R_t is the total return and r_t is a immediate reward
- Optimal (state-)value function :

$$v_*(s) := \max_{s \in \mathcal{S}} v_\pi(s), \ for\ all\ s \in \mathcal{S}$$

$$v_\pi(s) = \mathbb{E}_\pi[R_t|S_t = s] \quad \left(\approx \max_{a\in\mathcal{A}} q_\pi(s,a)\right)$$

$$= \mathbb{E}_\pi[\sum_{k=0}^{T-t} \gamma^k r_{t+k+1}\,|S_t = s]$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=0}^{T-t} \gamma^k r_{t+k+2}\,|S_t = s]$$

recursive expression

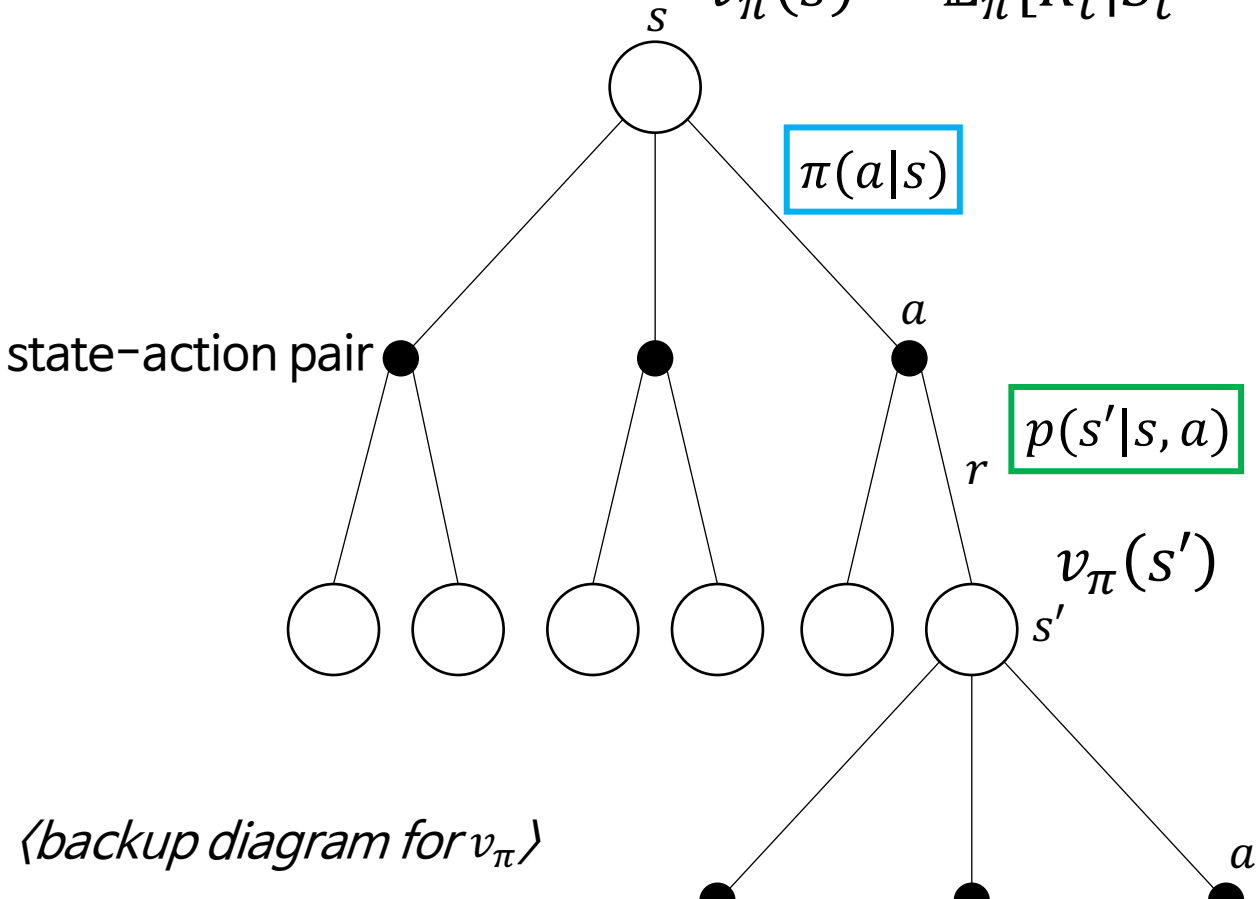
$$= \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=0}^{T-t} \gamma^k r_{t+k+2}\,|S_t = s'] \right]$$

$$= \sum_a \boxed{\pi(a|s)} \sum_{s'} \boxed{p(s'|s,a)} [r(s,a,s') + \gamma v_\pi(s')]$$

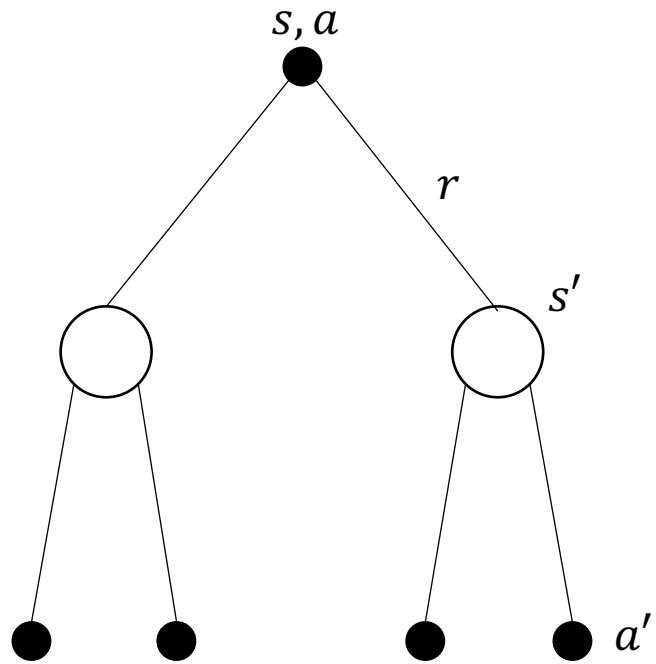
Bellman equation

$$\sum_{a} \pi(a|s) \sum_{s'} p(s'|s,a)\,[r(s,a,s') + \gamma v_\pi(s')]$$

$$v_\pi(s) = \mathbb{E}_\pi[R_t|S_t = s]$$



⟨backup diagram for v_π⟩



⟨backup diagram for q_π⟩

Action-Value Function

- **The value of taking action a** in state s under a policy π
  - $Q: s \rightarrow a|\pi$
  - *how good* it is for the agent to be in taking action a in state s under a policy π
- $q_\pi(s,a) \coloneqq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{T-t} \gamma^k r_{t+k+1}|S_t = s, A_t = a]$
- Optimal action-value function : $q_*(s) \coloneqq \max_{a \in \mathcal{A}(s)} q_\pi(s,a), for\ all\ s \in \mathcal{S}\ and\ a \in \mathcal{A}(s)$

Optimal Value Functions

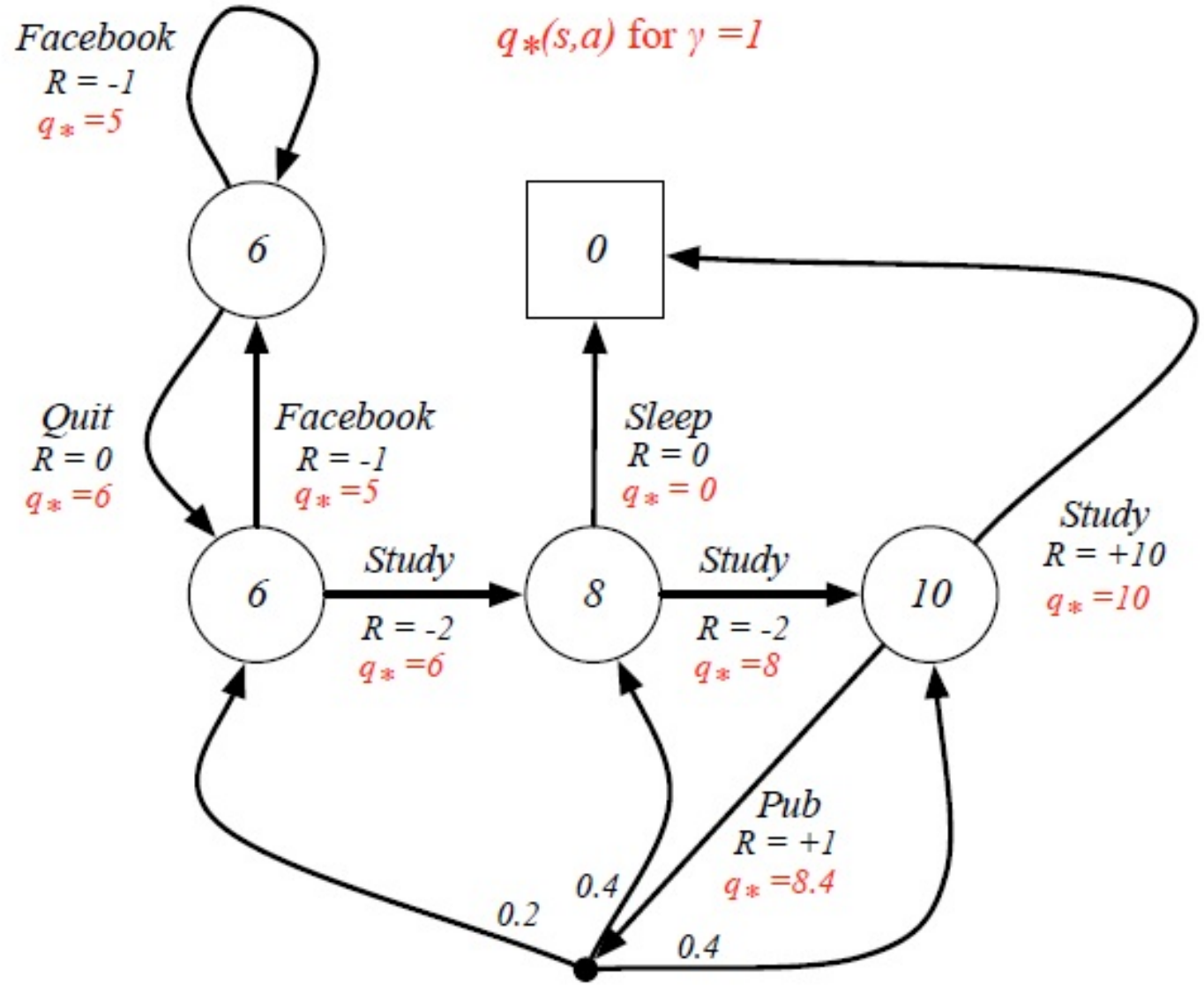
- Solving RL = **finding an optimal policy**

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Example : Student MDP Graph



Optimal Value Functions (cont.)

- π가 π'보다 낫다고 말할 수 있는 건 오직 π의 expected return이 π'보다 클 때
  - $v_\pi(s) \geq v_{\pi'}(s)$
  - $v_*(s) \coloneqq \max_\pi v_\pi(s)$
  - $q_*(s,a) \coloneqq \max_\pi q_\pi(s,a)$: the expected return for taking action a in state s
  - Express q_* in terms of v_*

$$q_*(s,a) = \mathbb{E}\left[\sum_{k=0}^{T-t} \gamma^k r_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

$$= \mathbb{E}\left[r_{t+1} + \gamma \sum_{k=0}^{T-t} \gamma^k r_{t+k+2} \,\middle|\, S_t = s, A_t = a\right]$$

$$= \mathbb{E}[r_{t+1} + \gamma v_*(s_{t+1}) | S_t = s, A_t = a]$$

$$= \mathbb{E}\left[r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a') \middle| S_t = s, A_t = a\right]$$

Bellman optimality equation

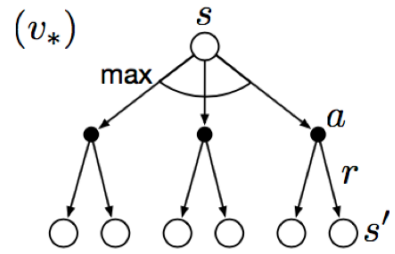
- $v_*(s)$ should be equal the expected return **for the best action from the state**

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_*(s')\right].
\end{aligned}
$$

Bellman optimality equation (cont.)

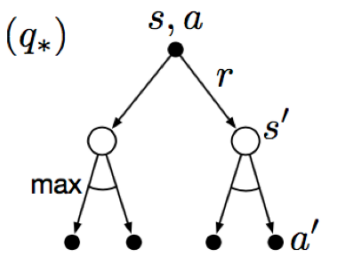
$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(s_{t+1}) | S_t = s, A_t = a]$$

$$= \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v_*(s')]$$



$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_a q_*(s_{t+1}, a') | S_t = s, A_t = a]$$

$$= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma \max_a q_*(s_{t+1}, a')]$$



- For **finite MDPs**, Bellman optimality equation **has a unique solution** independent of the policy
- DP are obtained by turning Bellman equations into **assignments** into **update rules** for improving approximations of value functions

Planning

Bellman (Optimality) Equation:

$$\forall s \in S : V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$$

Value Iteration:

$$\forall s \in S : V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V_i(s') \right]$$

Policy Iteration:

1) Policy evaluation:

$$\forall s \in S : V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s') [R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')]$$

2) Policy Improvement:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_k}(s') \right]$$

How to use

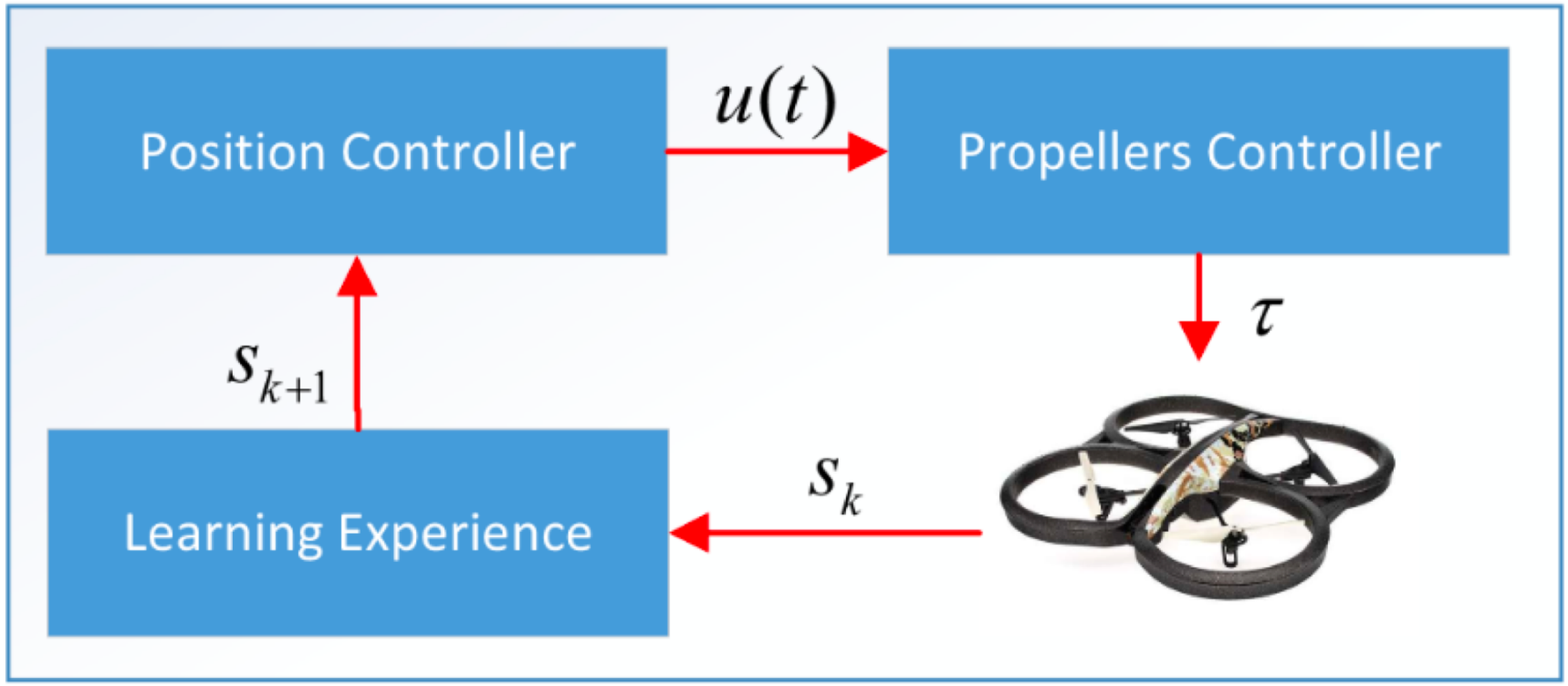
- Known MDP : Dynamic Programming
- Unknown MDP(model-free) : Monte-Carlo Learning
- Model-free + Learning at each time step : Temporal Difference
  (↑ On-policy reinforcement learning)

- Q-Learning : Off-policy reinforcement learning

UAV?

- Huy X. Pham, Hung. M. La, David Feil-Seifer, Luan V. Nguyen, “Autonomous UAV Navigation Using Reinforcement Learning”, arXiv preprint arXiv:1801.05086v1, 2018

Paper Review



Paper Review (cont.)

Algorithm 1: PID + Q-LEARNING.

Input: Learning parameters: Discount factor γ, learning rate α, number of episode N

Input: Control parameters: Control gains K_p, K_p, K_d, error radius d

Initialize $Q_0(s, a) \leftarrow 0$, $\forall s_0 \in S$, $\forall a_0 \in A$;

for $episode = 1 : N$ **do**

  Measure initial state s_0

  for $k = 0, 1, 2, \ldots$ **do**

    Choose a_k from A using policy (2)

    Take action a_k that leads to new state s_{k+1}:

    for $t = 0, 1, 2, \ldots$ **do**

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt}$$

    until $||p(t) - s_{k+1}|| \leq d$

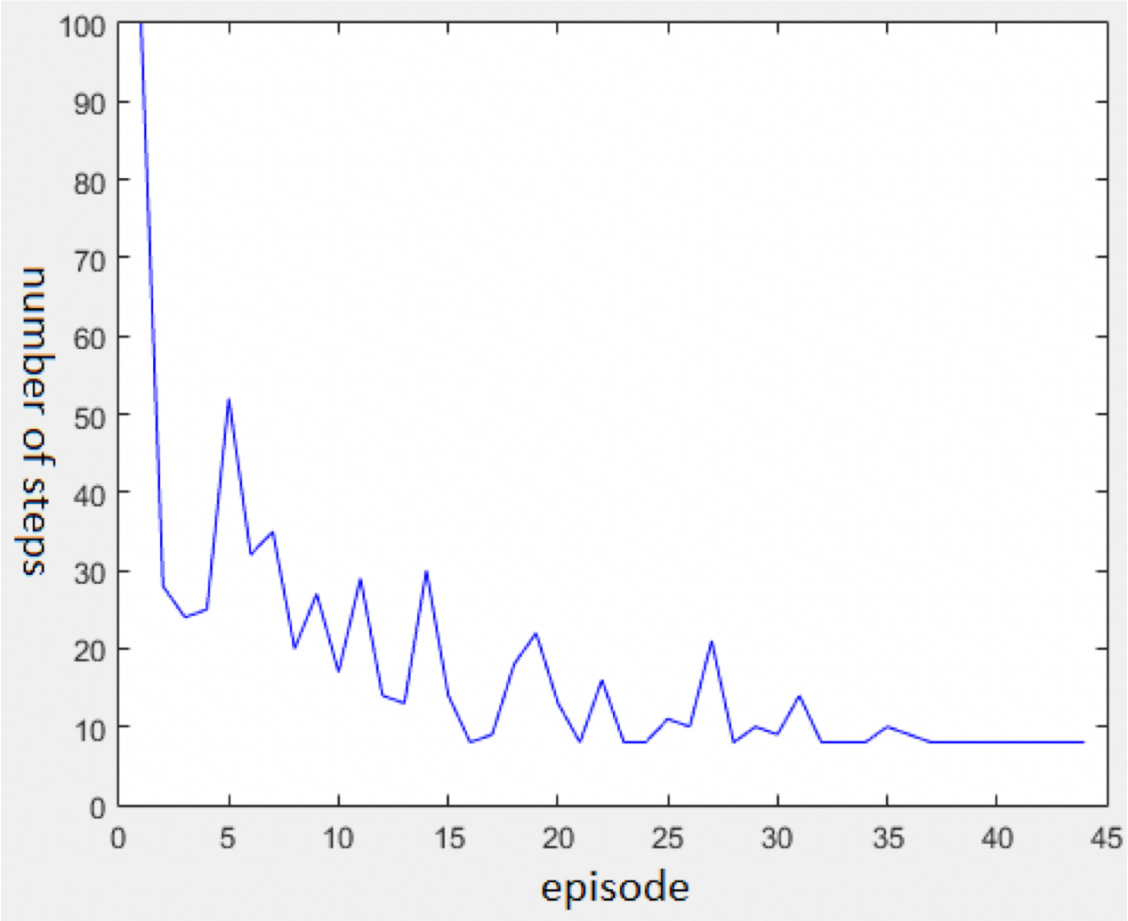
    Observe immediate reward r_{k+1}

    Update:

$$Q_{k+1}(s_k, a_k) \leftarrow (1 - \alpha) Q_k(s_k, a_k) + \alpha[r_{k+1} + \gamma \max_{a\prime} Q_k(s_{k+1}, a\prime)]$$

  until $s_{k+1} \equiv G$

Paper Review (cont.)



My work?

- 논문의 introduction에서 소개하길, 다양한 드론 관련 자동화 문제에 있어서 강화학습을 적용한 논문은 꽤나 있다고 한다.

Many UAV Problems

- “Learning swing-free trajectories for UAVs with a suspended load”
- “Controller design for quadrotor UAVs using reinforcement learning”
- “Design of attitude and path tracking controllers for quadrotor robots using reinforcement learning”
- “Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning”

- 이외에도 다수…

My work?

- 하지만 이런 문제들이 구체적으로 구현된 경우는 없다(본 논문의 경우는 그것을 실제로 구현한 경우).
- 이런 문제들을 직접 구현해보기?

Future work

- DP, Monte-Carlo Method, TD 등 강화학습 공부 계속
- 강화학습 외에도 최신 트렌드인 머신 러닝 공부를 통해 다른 방향으로 문제에 어떻게 접근할 수 있을지 고민
- 어떤 방향으로(논문의 경우 간단한 길찾기, hovering, landing 등의 경우에 적용 가능할 듯함) drone에 강화학습을 적용시킬지 탐구
- 강화학습 실제 구현 공부 (python)
- Python library를 이용한 bebop drone 제어 공부 (https://github.com/amymcgovern/pyparrot)

- 이외에도 '가능하다면' 컨트롤러 설계 이해를 위한 역학공부

Reference (Reinforcement Learning)

(Main reference)

- Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, incompleteideas.net/book/bookdraft2017nov5.pdf

(Photo reference)

- UC Berkeley Intro to AI Lecture notes, http://ai.berkeley.edu/lecture_slides.html
- University College London Reinforcement Learning Lecture notes (Prof. David Silver), http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Reference (UAV)

- Huy X. Pham, Hung. M. La, David Feil-Seifer, Luan V. Nguyen, “Autonomous UAV Navigation Using Reinforcement Learning”, arXiv preprint arXiv:1801.05086v1, 2018
- A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, “Learning swing-free trajectories for uavs with a suspended load,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4902 - 4909.
- H. Bou-Ammar, H. Voos, and W. Ertel, “Controller design for quadrotor uavs using reinforcement learning,” in *Control Appli- cations (CCA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2130 - 2135.

Reference (UAV)

- S. R. B. dos Santos, C. L. Nascimento, and S. N. Givigi, “Design of attitude and path tracking controllers for quad-rotor robots using reinforcement learning,” in *Aerospace Conference, 2012 IEEE*. IEEE, 2012, pp. 1 - 16.
- S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, “Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning,” in *Intelligent Robots and Sys- tems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 3712 - 3717.

감사합니다.