인공지능의 기초

마르코프 결정 과정

Markov Decision Processes

- A Markov decision process is an MRP with decisions: $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma} \rangle$
  - A set of states $S = \{s_1, s_2, \ldots, s_n\}$
  - A set of actions $A = \{a_1, a_2, \ldots, a_m\}$
  - Transition function $P: S \times A \to S$, $P_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
  - Reward function $R: S \times A \to \mathbb{R}$, $R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - Discount factor $\gamma \in [0,1]$

MEMO

Markov Decision Processes

- A policy π is a distribution over actions given states

$$\pi(a|s) = P[A_t = a|S_t = s]$$

  - MDP policies depend on the current state (not the history)
  - Policies are stationary (time-independent) $A_t \sim \pi(\cdot\,|S_t), \ \forall t > 0$

MEMO

Value Functions

- The state-value function $V_\pi(s)$ is the expected return starting from state s, under a policy π

$$V_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right]$$

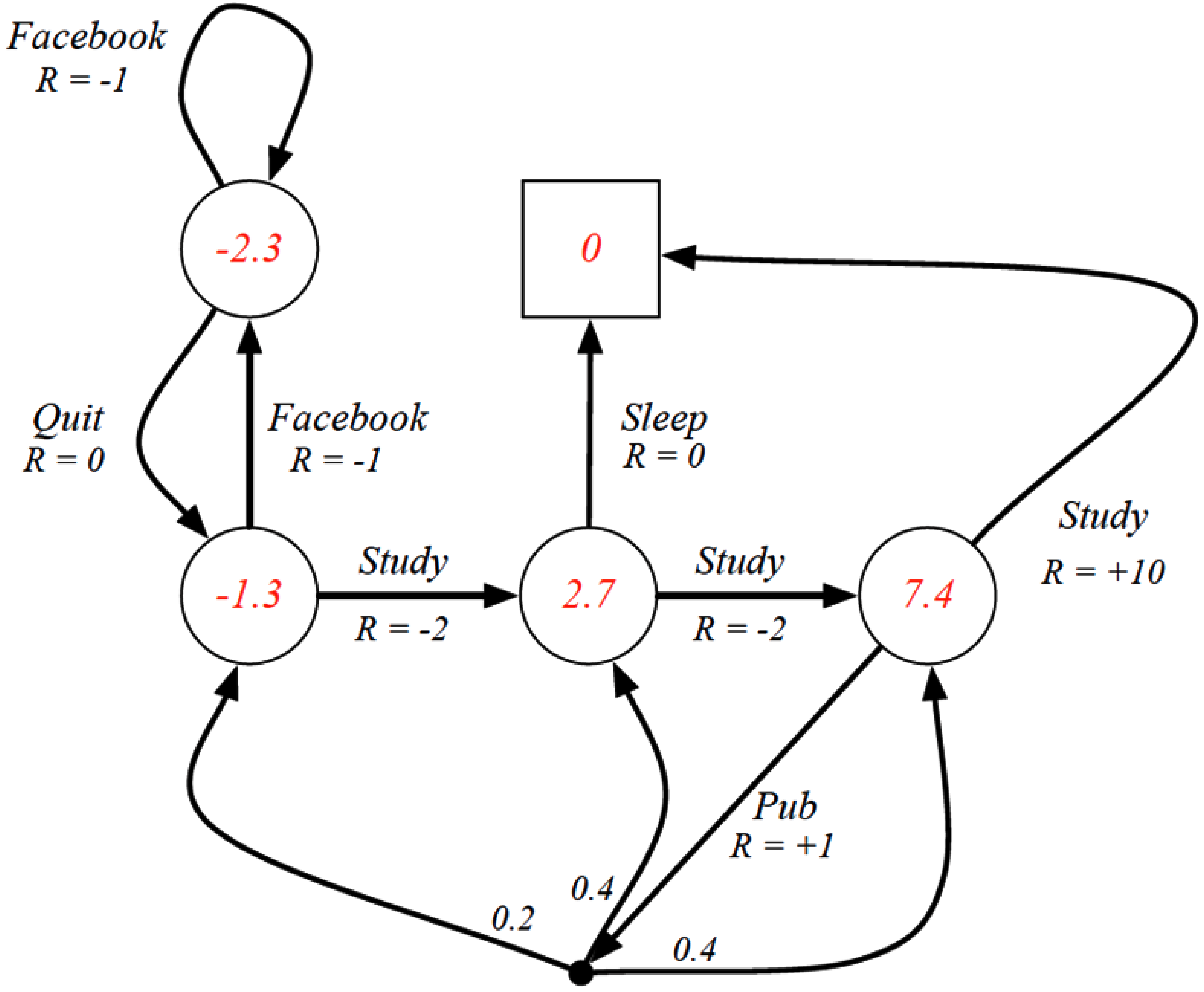
- The action-value function $Q_\pi(s, a)$ is the expected return starting from state s, taking action a, under a policy π

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

MEMO

Example of Student MDP

- Random policy with $\gamma = 1$
  - $V_\pi(s)$ for $\pi(a|s) = 0.5$



MEMO

Bellman Expectation Equation for MDPs

- **The value function can be decomposed into two parts:**
  - Immediate reward R_{t+1}
  - Discounted value of successor state $\gamma V(s_{t+1})$

- **The state-value function can be decomposed**

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi[G_t | s_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | s_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) | s_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | s_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]
\end{aligned}
$$

MEMO

Bellman Expectation Equation for MDPs

- The value function can be decomposed into two parts:
  - Immediate reward R_{t+1}
  - Discounted value of successor state $\gamma V(s_{t+1})$

- The state-value function can be decomposed

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1}) | S_t = s\right]$$

- The action-value function can be decomposed

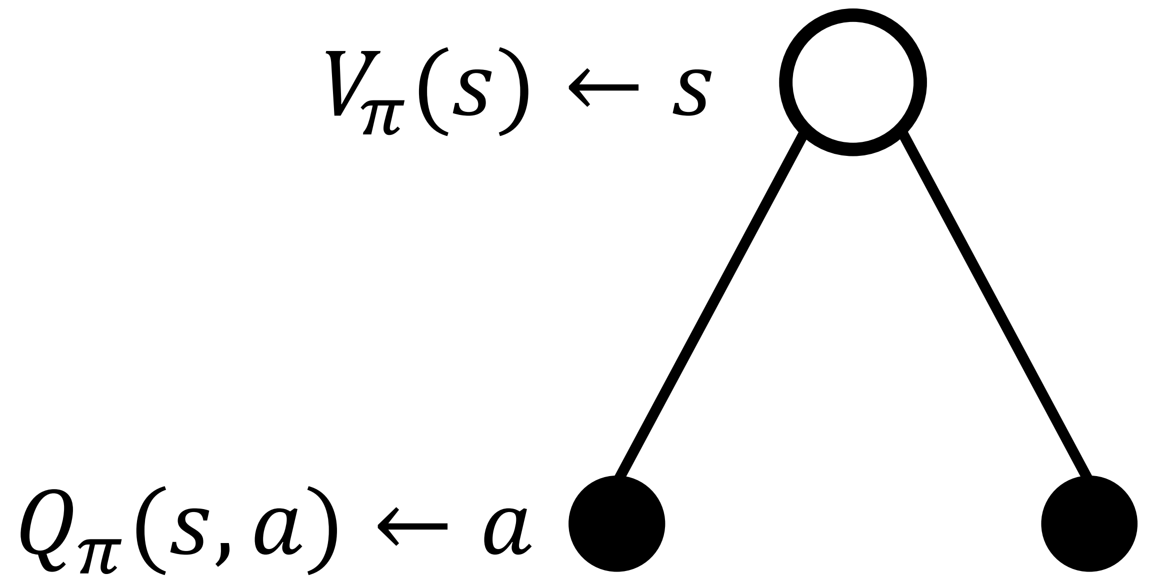
$$Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

MEMO

Bellman Equation for V_π and Q_π

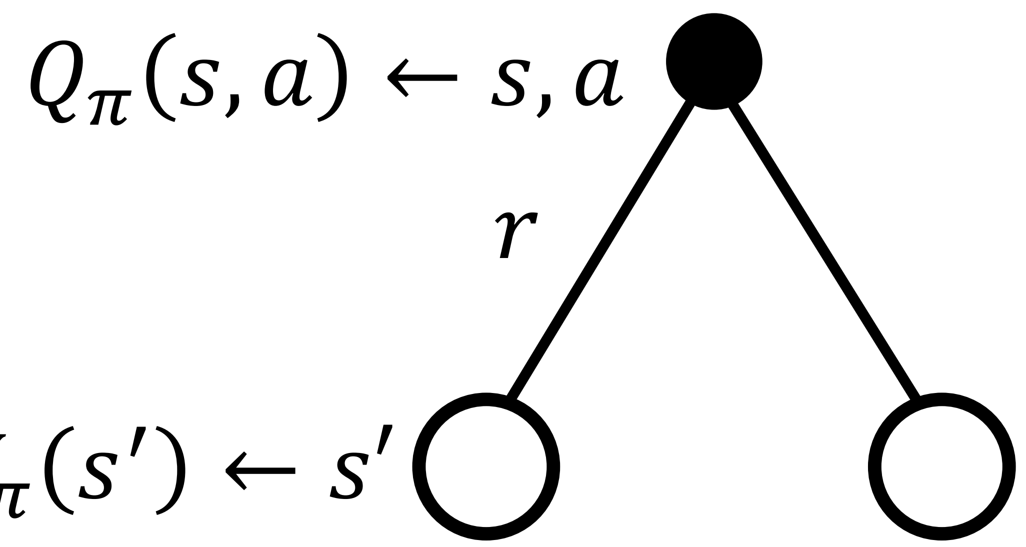
Bellman expectation equation for V_π

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) Q_\pi(s,a)$$



Bellman expectation equation for Q_π

$$Q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_\pi(s')$$

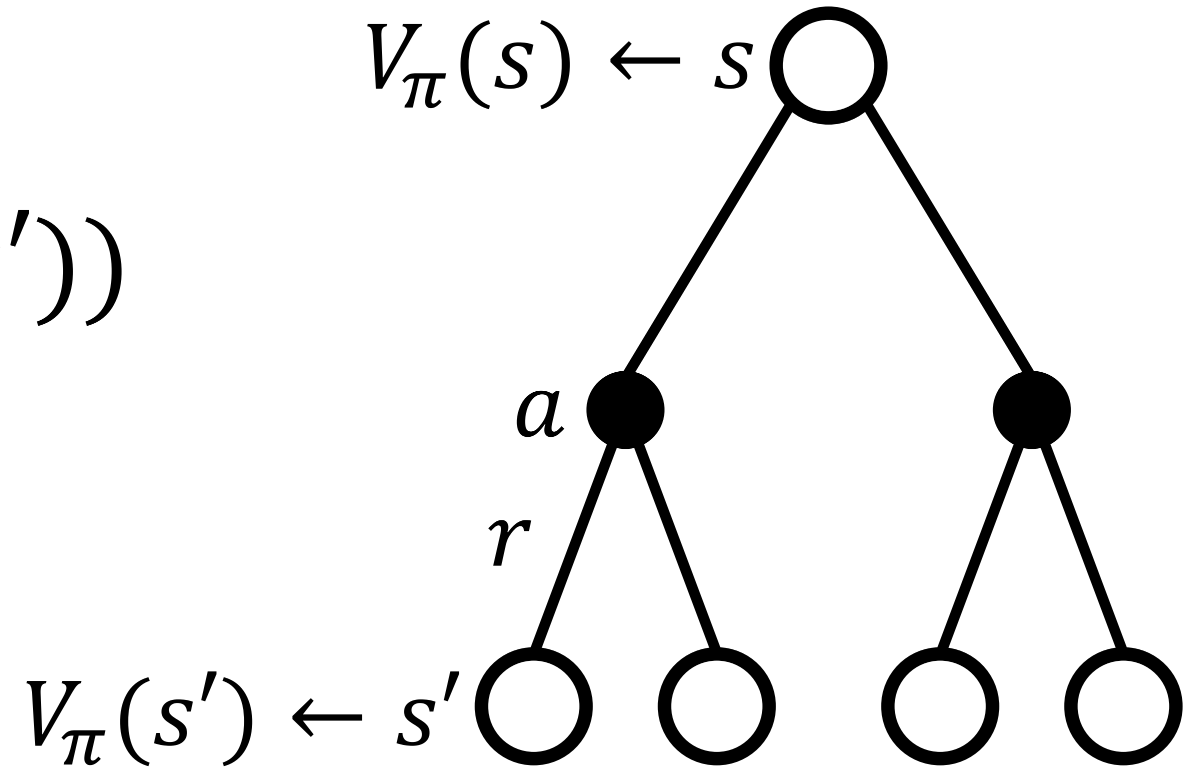


MEMO

Bellman Equation for V_π and Q_π

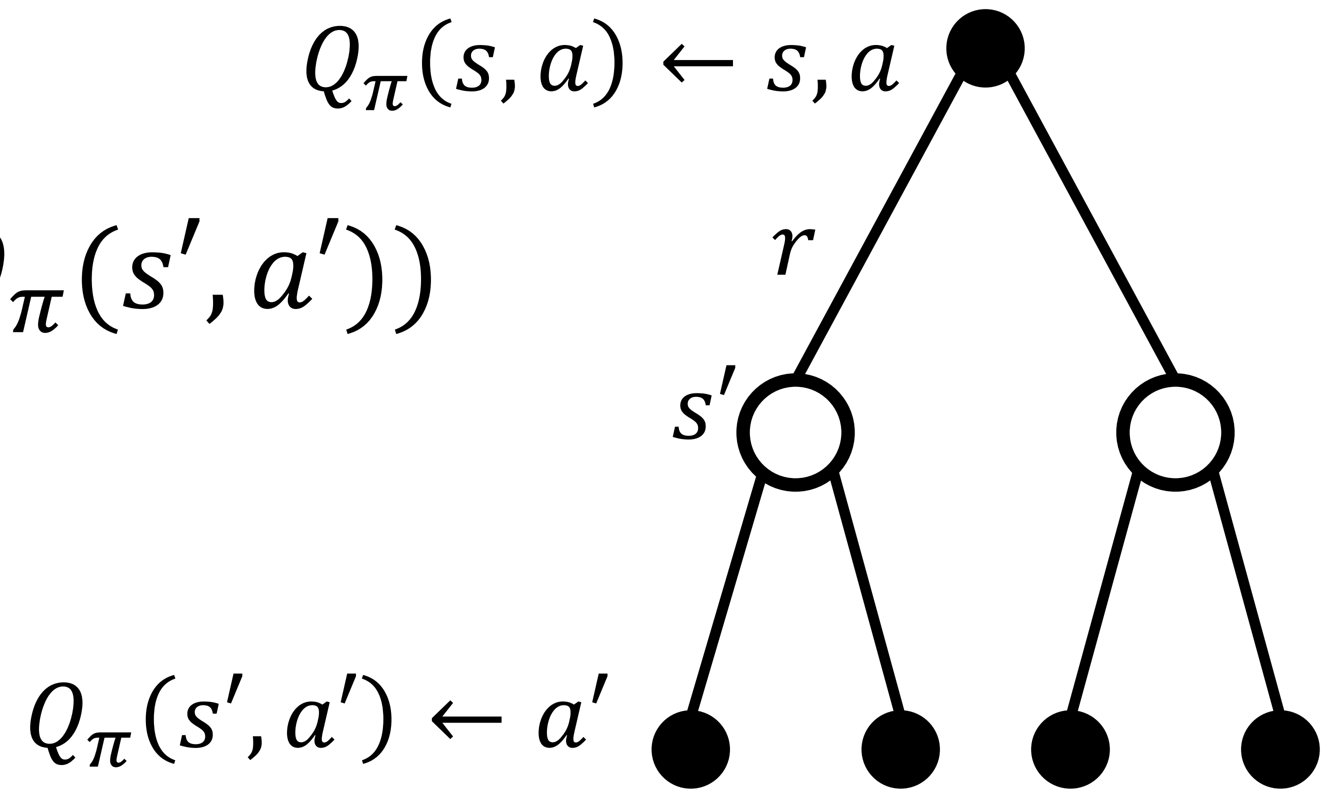
Bellman expectation equation for V_π (2)

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



Bellman expectation equation for Q_π (2)

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

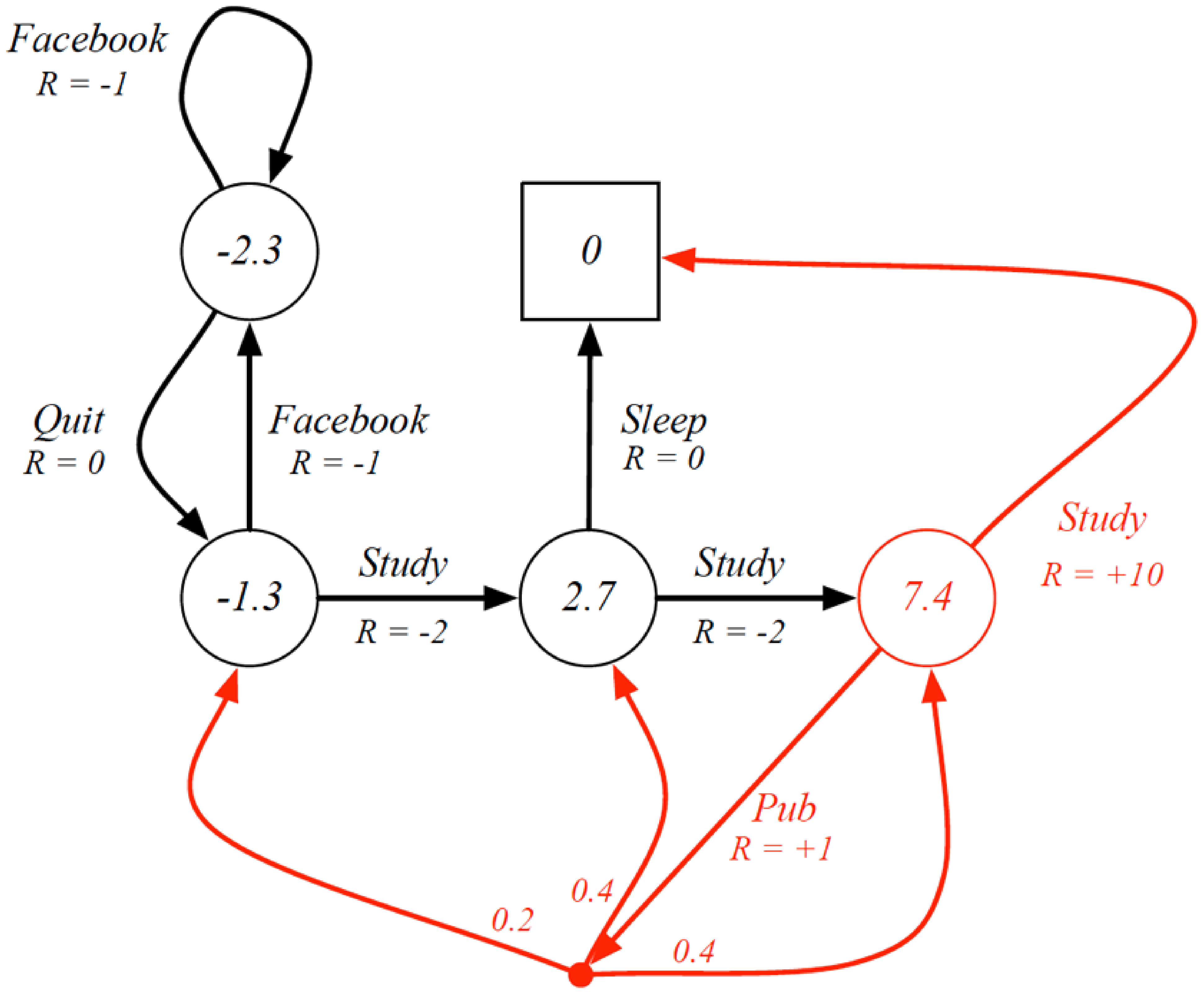


MEMO

Bellman Equation in Student MDP

$$V_\pi(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s\prime \in S} P_{ss\prime}^a V_\pi(s'))$$

$$V_\pi(s) = 0.5 * (1 + 0.2 * (-1.3) + 0.4 * 2.7 + 0.4 * 7.4) + 0.5 * 10 = 7.4$$



MEMO

Bellman Expectation Equation (Matrix Form)

- Can be expressed concisely in a matrix form

$$V_\pi = R^\pi + \gamma P^\pi V_\pi$$

$$\begin{bmatrix} V_\pi(s_1) \\ \dots \\ V_\pi(s_n) \end{bmatrix} = \begin{bmatrix} R_1^\pi \\ \dots \\ R_n^\pi \end{bmatrix} + \gamma \begin{bmatrix} P_{11}^\pi & \dots P_{1n}^\pi \\ & \dots \\ P_{n1}^\pi & \dots P_{nn}^\pi \end{bmatrix} \begin{bmatrix} V_\pi(s_1) \\ \dots \\ V_\pi(s_n) \end{bmatrix}$$

- It is a linear equation, so solved by

$$V_\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

  - Computational complexity is $O(n^3)$ for n states
  - Other approach? Dynamic programming, Monte-Carlo evaluation, Temporal-Difference learning

MEMO

Optimal Value Functions

- The optimal state value function $\boldsymbol{V_*(s)}$ is the max value function over all policies

$$V_*(s) = \max_{\pi} V_{\pi}(s)$$

- The optimal action-value function $\boldsymbol{Q_*(s,a)}$ is the maximum action-value function over all policies

$$Q_*(s,a) = \max_{\pi} Q_{\pi}(s,a)$$

- Theorems: For any MDP
  - There exists an optimal policy $\pi_* \geq \pi, \forall \pi$
  - All optimal policies achieve the optimal state-value, $V_{\pi_*}(s) = V_*(s)$
  - All optimal policies achieve the optimal action-value, $Q_{\pi_*}(s,a) = Q_*(s,a)$

MEMO

Finding an Optimal Policy

- An optimal policy can be found by maximizing over $\boldsymbol{Q_*(s,a)}$

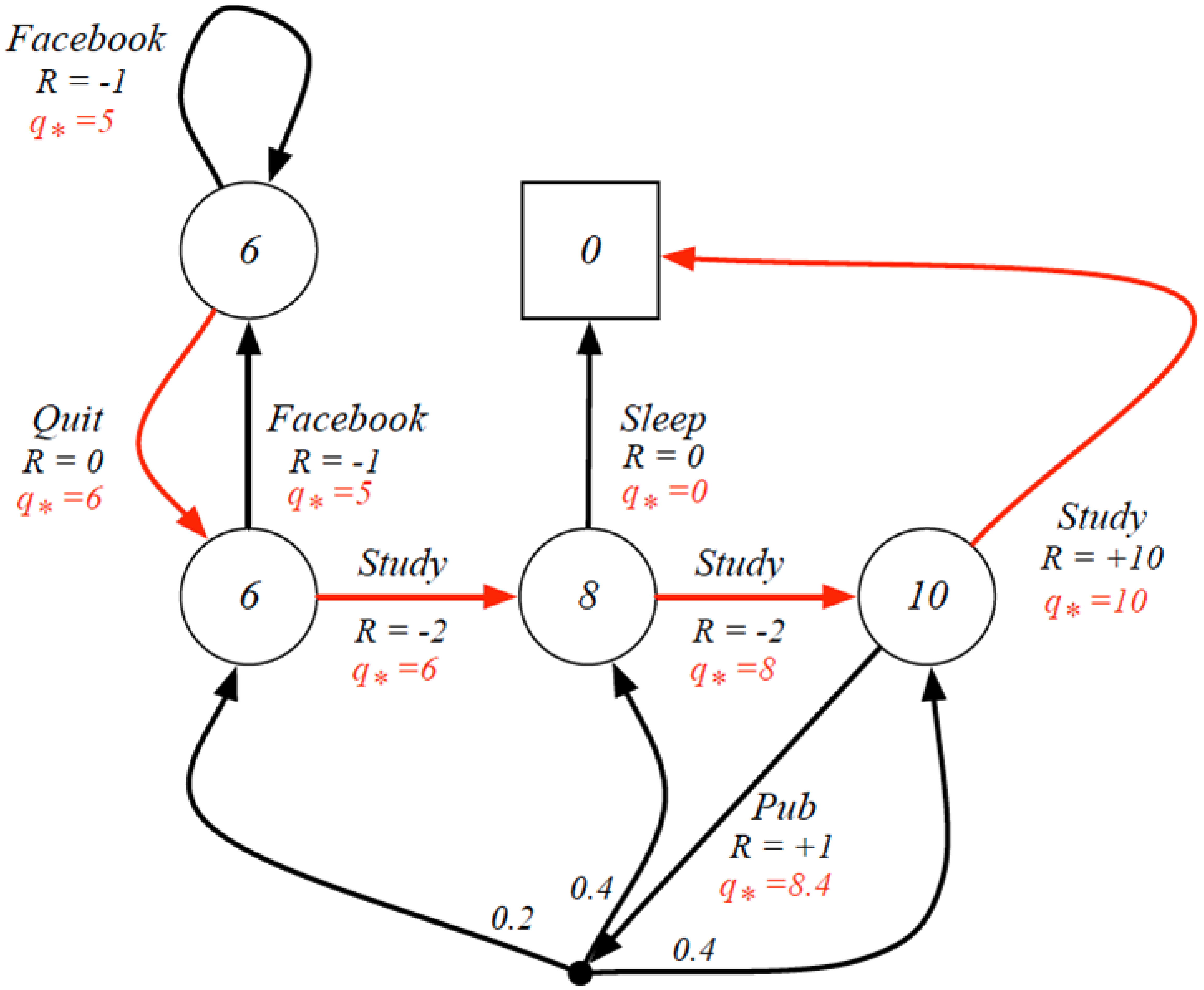
$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in A}{\operatorname{argmax}}\, Q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

  - If we know $Q_*(s,a)$, we immediately have the optimal policy

MEMO

Optimal Policy for Student MDP

$$\pi_*(a|s) \text{ for } \gamma = 1$$



MEMO

인공지능의 기초

동적 프로그래밍을 통한 마르코프 결정 과정

Dynamic Programming

- A very general solution method for problems which have two properties

  1. Optimal substructure
     - Optimal solution can be decomposed into subproblems
  2. 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

MEMO

Prediction and Control

Prediction: evaluate the future

- Given an MDP $\langle S, A, P, R, \gamma \rangle$ and a policy π
- Output: a value function V_π

Iterative policy evaluation!

Control: optimize the future

- Given an MDP $\langle S, A, P, R, \gamma \rangle$
- Output: optimal policy π_* (and optimal value function V_*)

MEMO

Iterative Policy Evaluation

- **Problem: evaluate a given policy π**

- **Solution: iteratively apply Bellman expectation backup**
  - Converge to a real V_π $(V_1 \rightarrow V_2 \rightarrow \cdots \rightarrow V_\pi)$
  - At each iteration $k+1$, for all states $s \in S$, update $V_{k+1}(s)$ from $V_k(s')$ where s' is a successor state of s

- **Iteratively compute until convergence**

$$\boldsymbol{V}^{k+1} = \boldsymbol{R}^{\pi} + \gamma \boldsymbol{P}^{\pi} \boldsymbol{V}^{k}$$

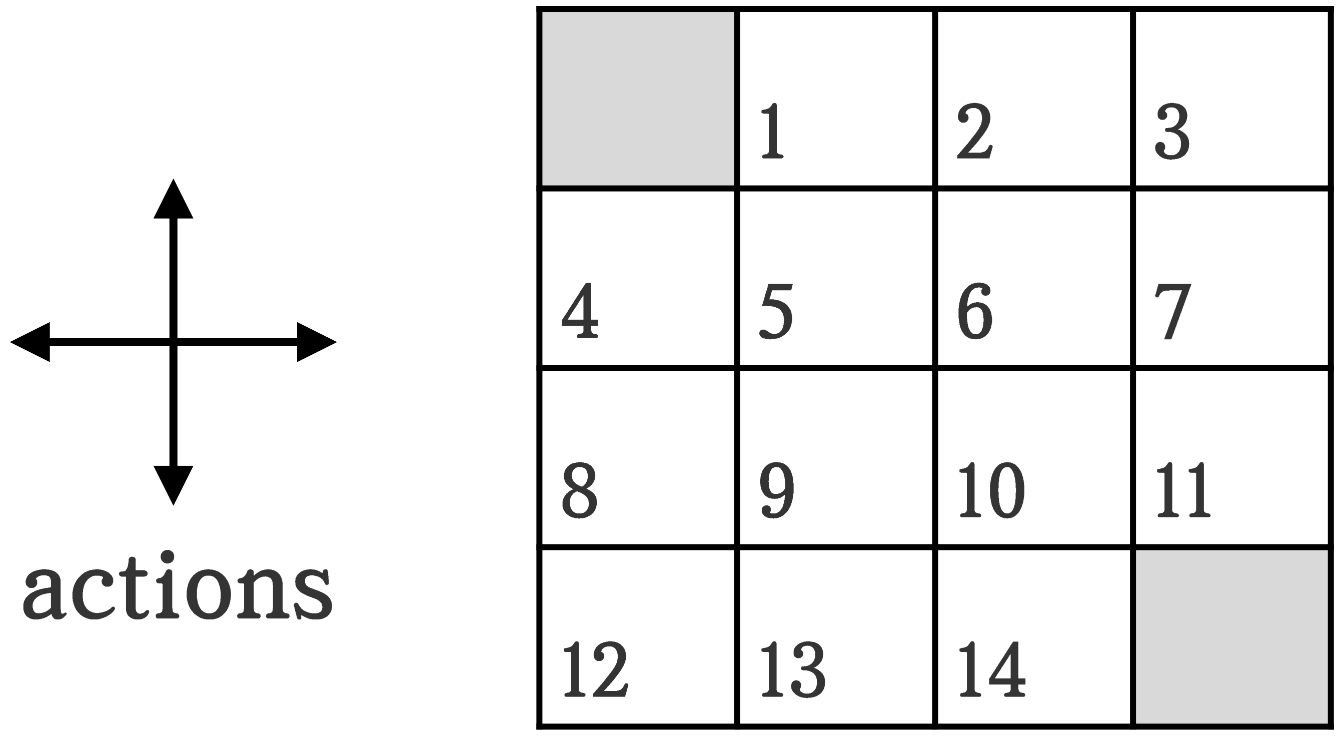
  - Matrix form of Bellman expectation equation

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

MEMO

Evaluating Random Policy in Small Gridworld

Problem setup



- Undiscounted episodic MDP ($\gamma = 1$)
- Terminal state: two 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

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

MEMO

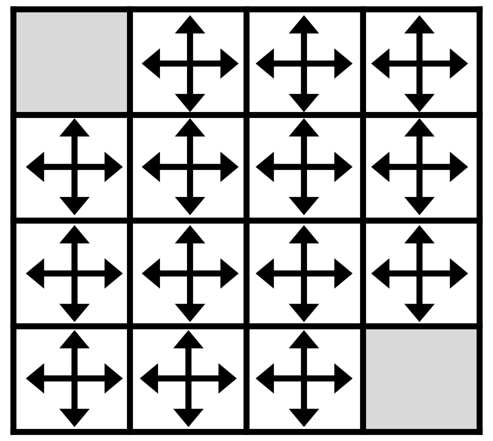
Iterative Policy Estimation in Small Gridworld

Problem setup

V^k for the random policy | Greedy policy w.r.t. V^k | Converged optimal policy

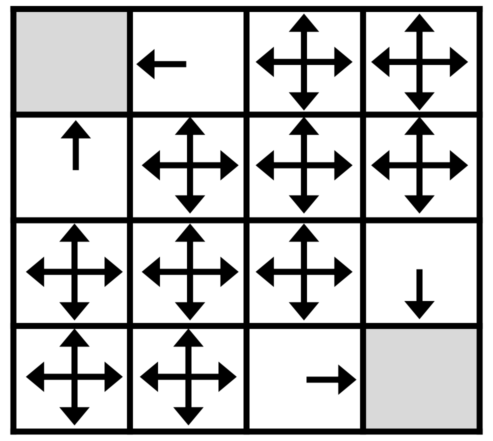
$k = 0$

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



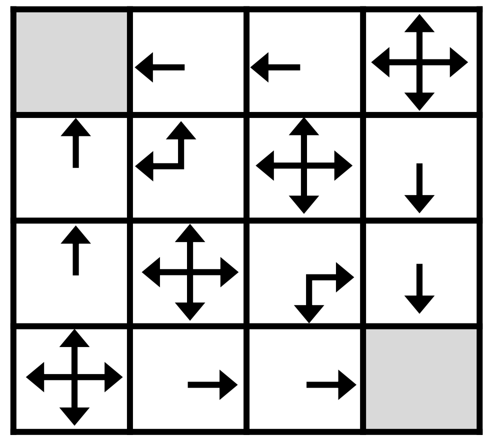
$k = 1$

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



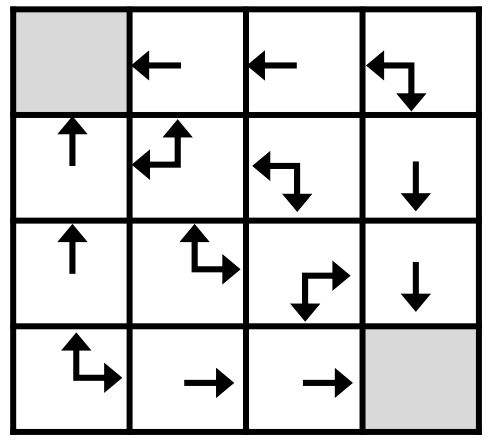
$k = 2$

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



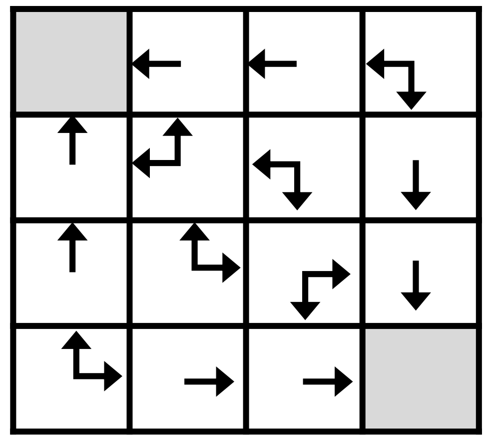
$k = 3$

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



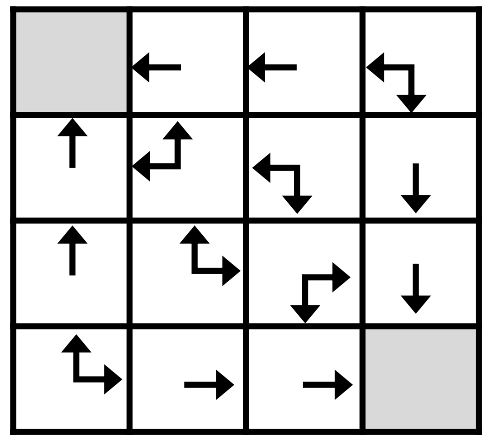
$k = 10$

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



$k = \cdots$

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



MEMO

Policy Improvement

- **Now we know how to evaluate V_π for a given policy π**

- **How to improve a policy?**
  - Initialize a policy π
  - Evaluate the policy π to compute V_π
  - Improve the policy by acting greedily with respect to V_π

$$\pi' = \mathrm{greedy}(V_\pi)$$

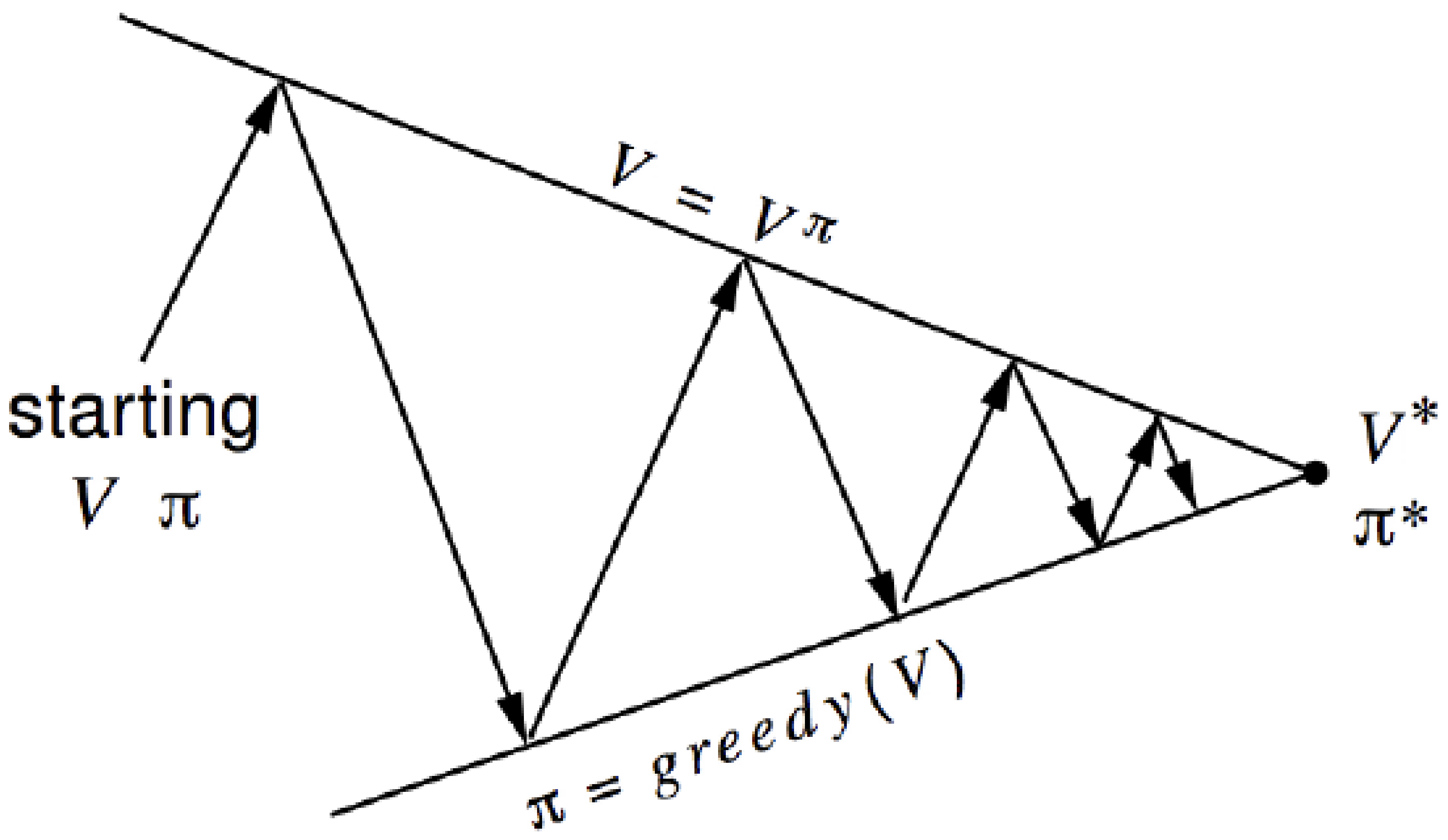
$$\pi'(s) = \operatorname*{argmax}_{a \in A} Q_\pi(s, a)$$

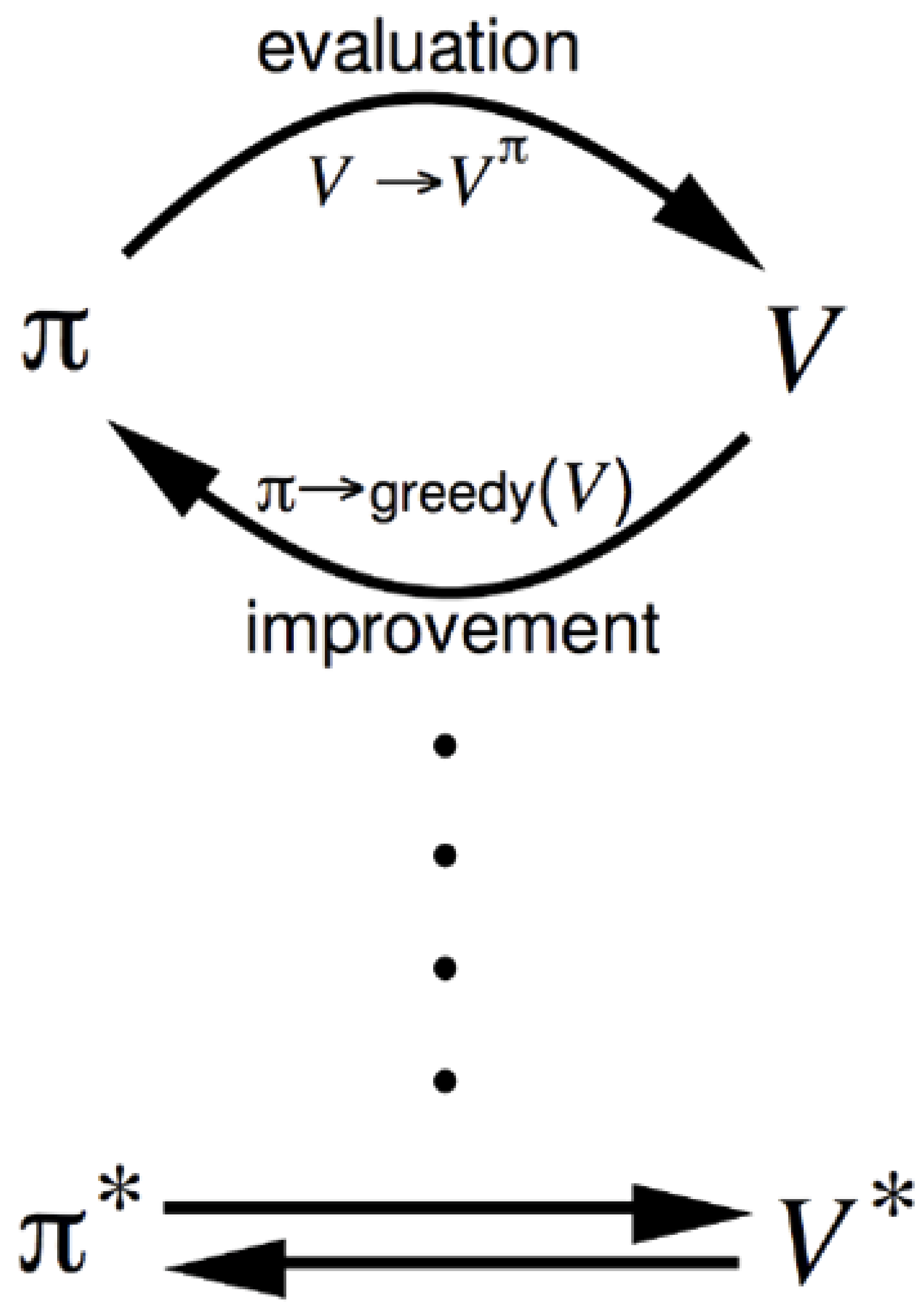
  - This process of policy iteration always converges to π_*

MEMO

Policy Iteration

- **Policy evaluation: estimate V_π**
  - Iterative policy evaluation
- **Policy improvement: generate $\pi' \geq \pi$**
  - Greedy policy improvement





MEMO

DP Algorithms

Problem	Bellman equation	Algorithm
Prediction	Bellman expectation equation	Iterative policy evaluation
Control	Bellman expectation equation + Greedy policy improvement	Policy 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

MEMO