slides are adapted from David Silver

COMP5212: Machine Learning

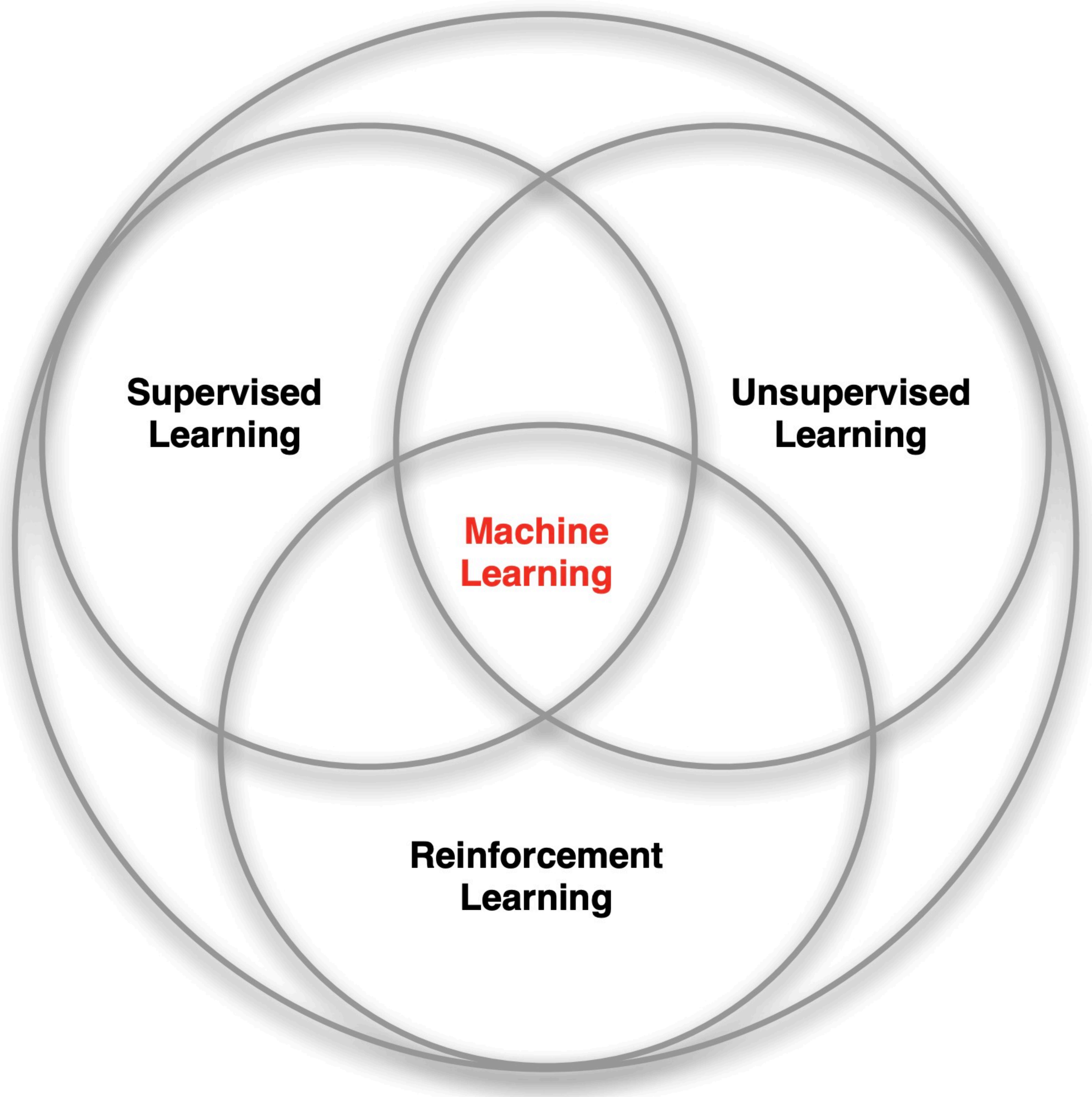
Lecture 20

Minhao Cheng

Logistics

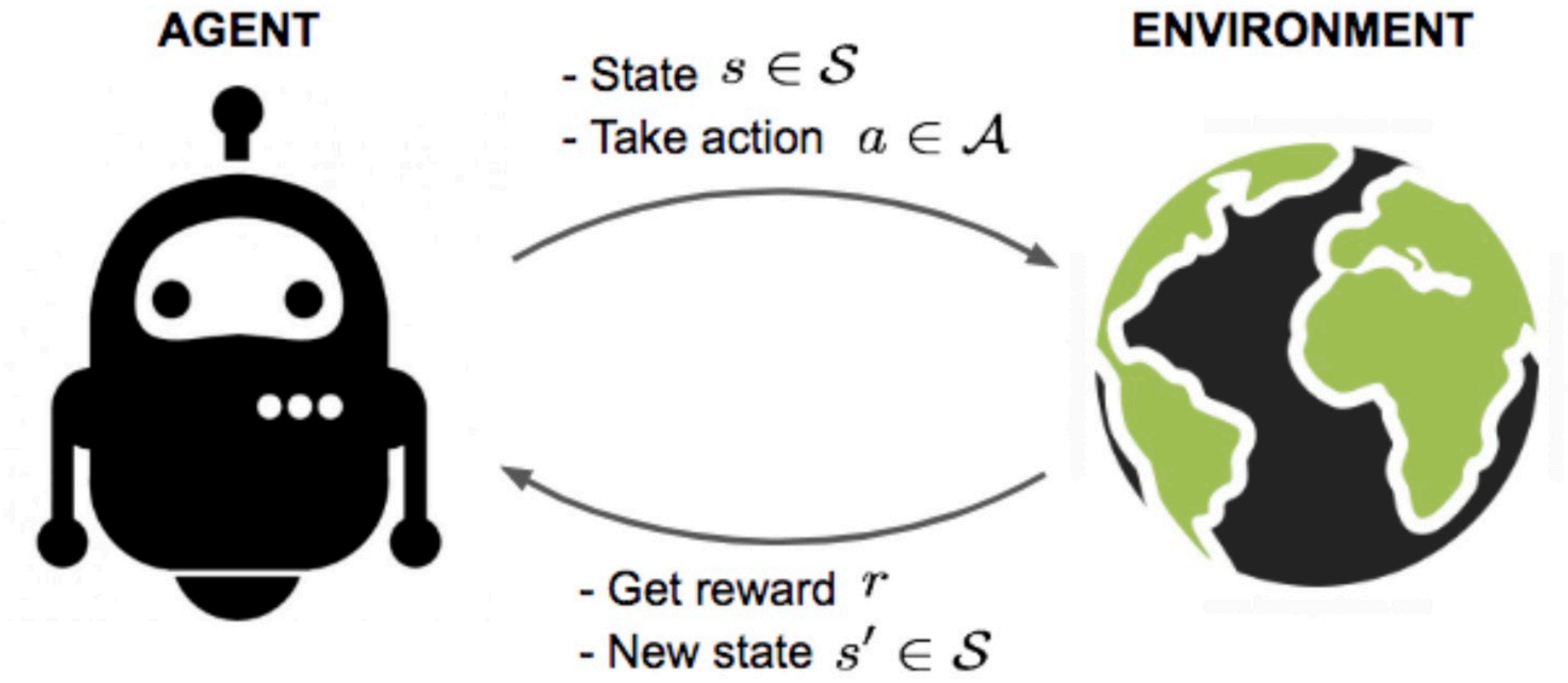
- Final project:
  - Presentation: 6-7 mins on the last two lectures
  - Report: due in Dec 15th
- Homework 3 will be out this week (written and programming combined)

Branches of Machine Learning



Reinforcement Learning
Introduction

- The goal of Reinforcement Learning (RL) is to learn a good strategy for the agent from experimental trials and relative simple feedback received.



Reinforcement Learning

Characteristics of Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential, non i.i.d data)
  - Agent’s actions affect the subsequent data it receives

Reinforcement Learning

Markov Decision Processes

- Markov decision processes formally describe an environment for reinforcement learning
- Where the environment is fully observable
- i.e. The current state completely characterises the process
- Almost all RL problems can be formalised as MDPs

Markov Decision Processes

Markov Property

"The future is independent of the past given the present"

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, ..., S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

Markov Decision Processes

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

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

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \textit{from} \overset{\textit{to}}{\begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}}$$

where each row of the matrix sums to 1.

Markov Decision Processes

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, ...$ with the Markov property.

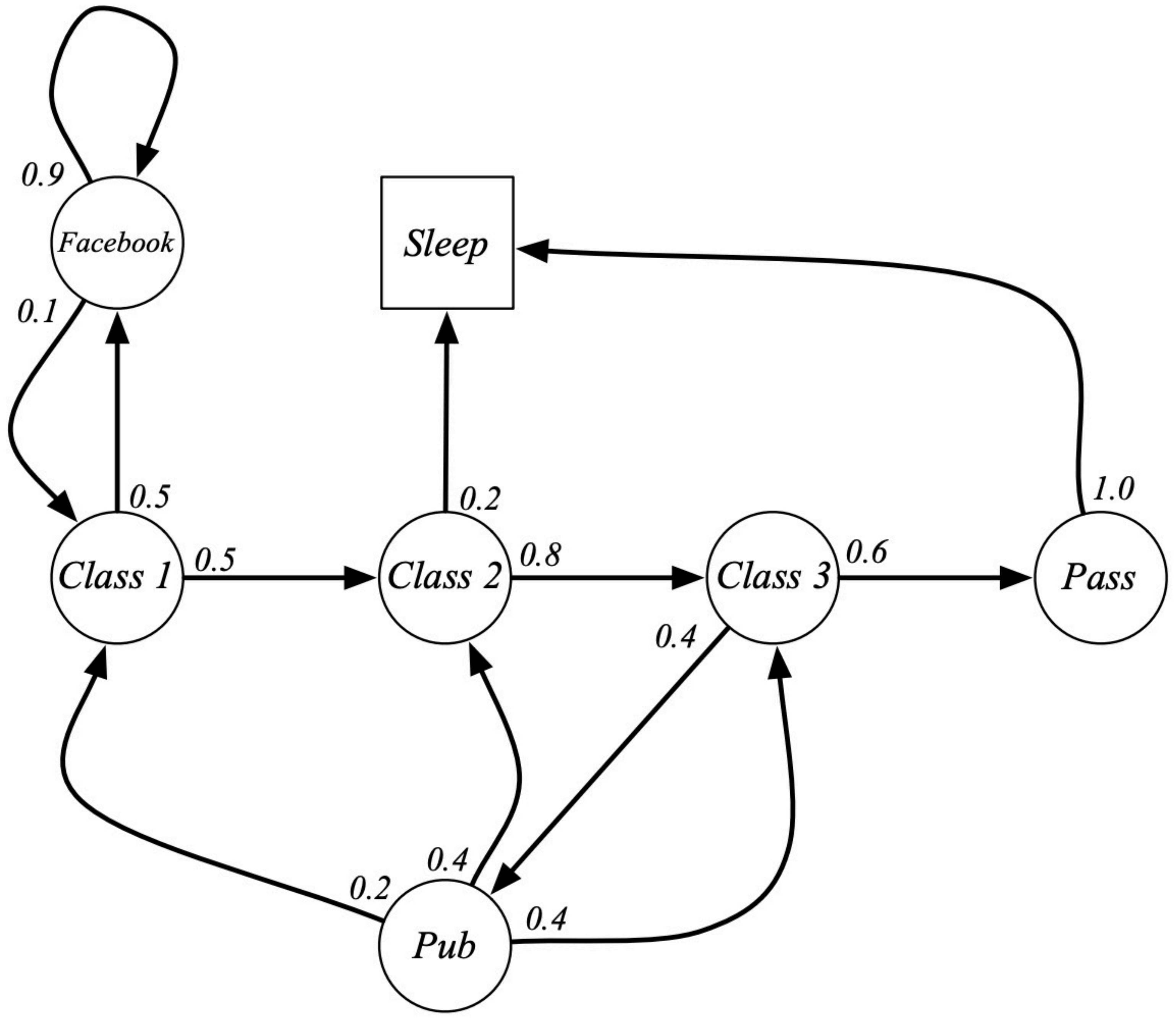
Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$

Markov Decision Processes

Example



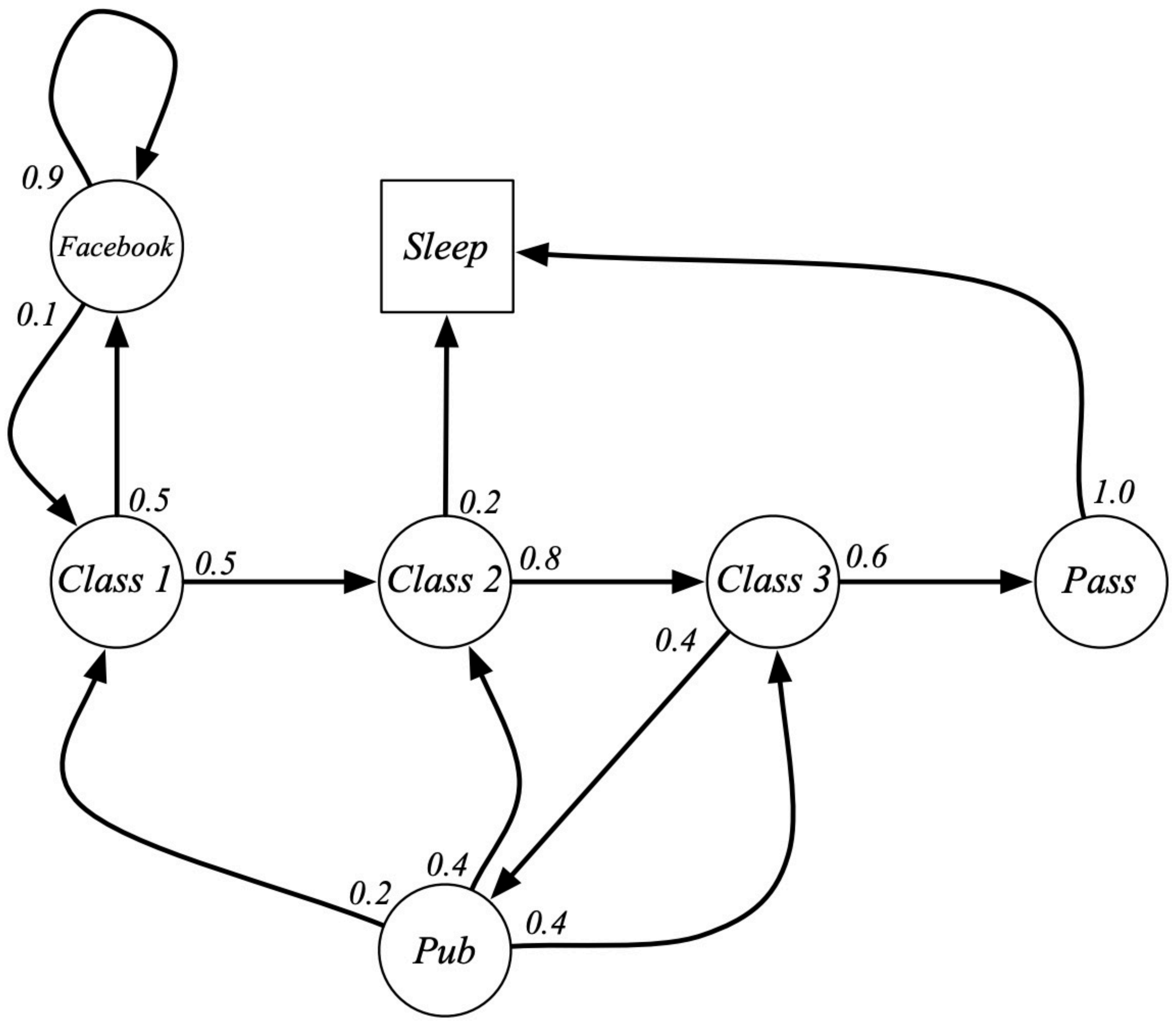
Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Markov Decision Processes

Example



$\mathcal{P} =$

	C1	*C2*	*C3*	*Pass*	*Pub*	*FB*	*Sleep*
C1		0.5				0.5	
C2			0.8				0.2
C3				0.6	0.4		
Pass							1.0
Pub	0.2	0.4	0.4				
FB	0.1					0.9	
Sleep							1

Markov Decision Processes

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,
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Markov Decision Processes

Return

Definition

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

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Markov Decision Processes

Policy

Definition

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

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

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent), $A_t \sim \pi(\cdot|S_t), \forall t > 0$

Markov Decision Processes

Value Function

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

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

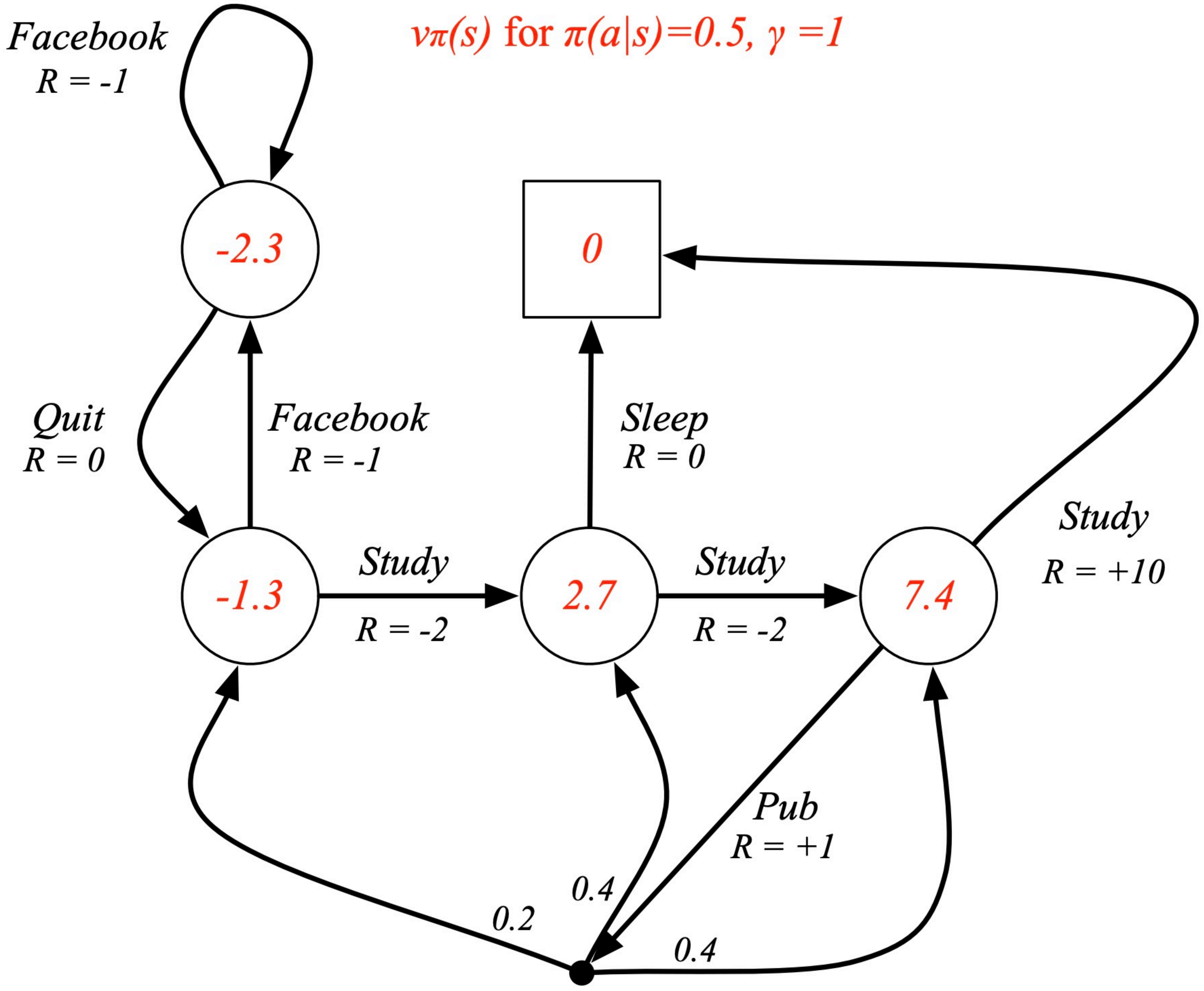
Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Markov Decision Processes

Example



Markov Decision Processes

Bellman Expectation Equation

- The state-value function can again be decomposed into immediate reward plus discounted value of successor state

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\Big] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \\
&= \mathbb{E}_\pi\left[r + \gamma v_\pi(s') \mid S_t = s\right].
\end{aligned}
$$

Markov Decision Processes

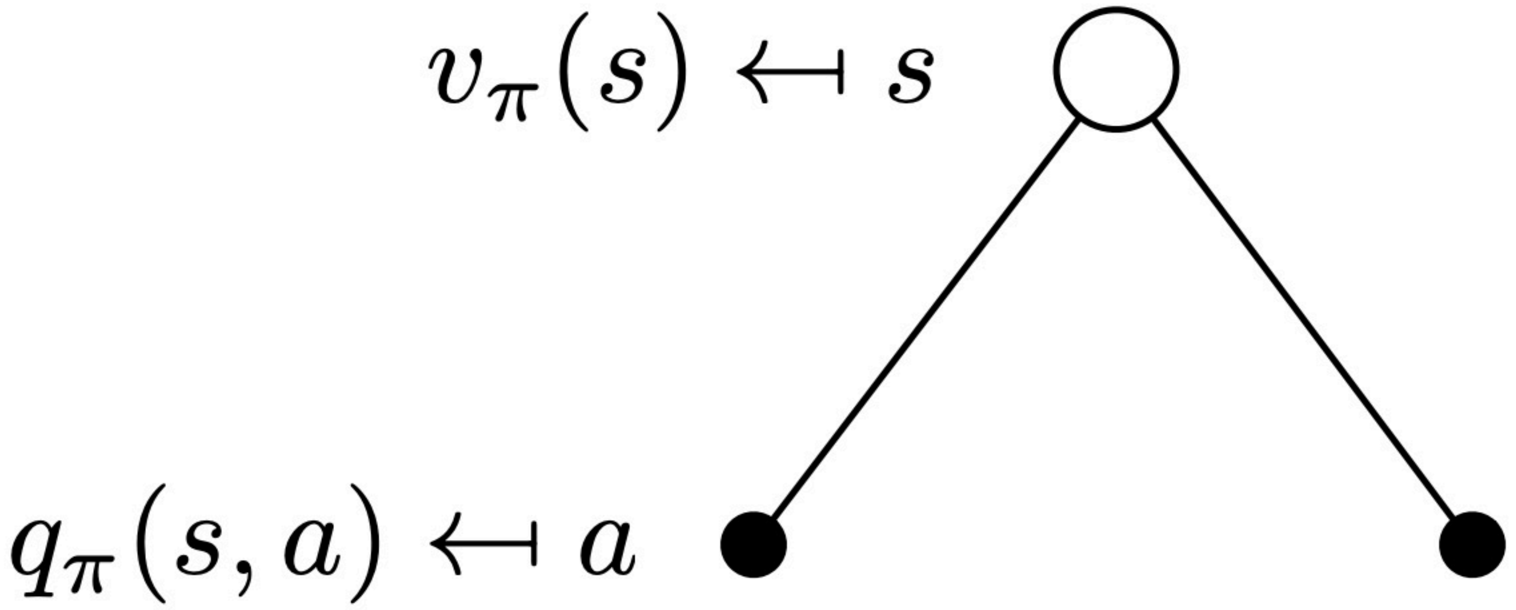
Bellman Expectation Equation

- The action-value function can similarly be decomposed,

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

Markov Decision Processes

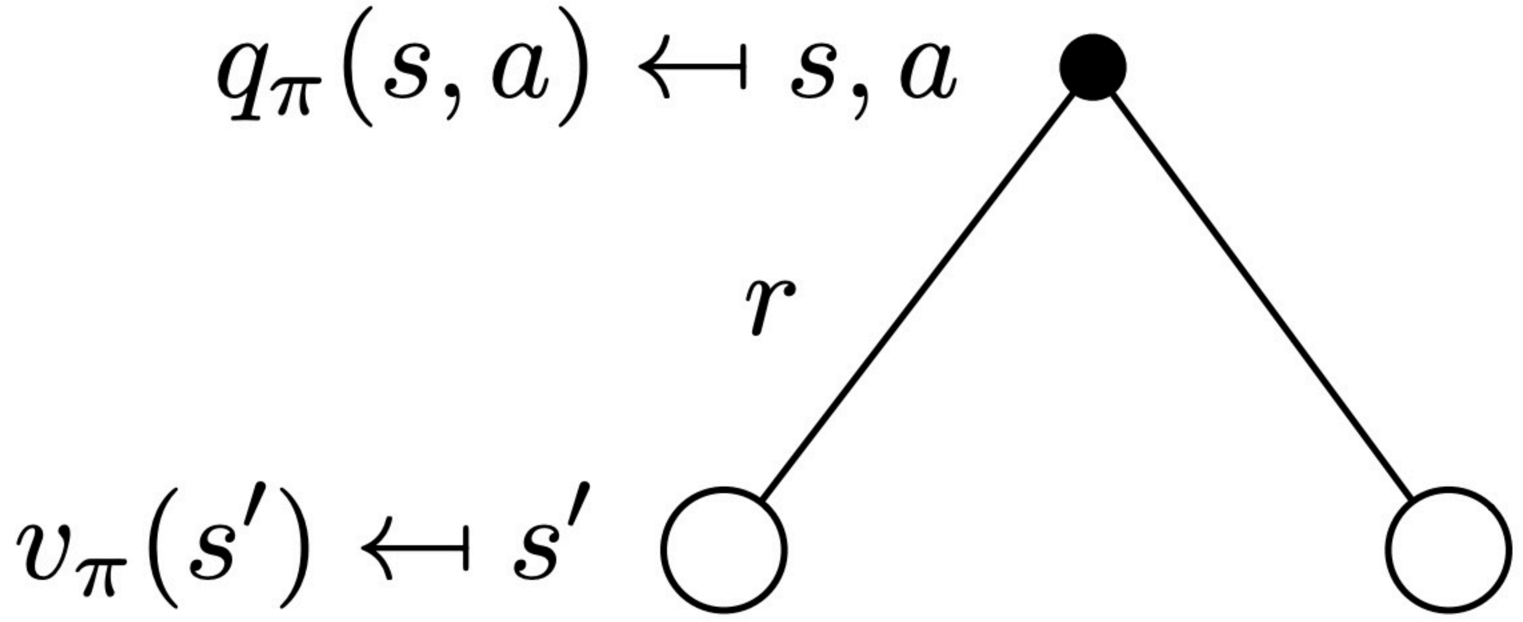
Bellman Expectation Equation (Backup)



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Markov Decision Processes

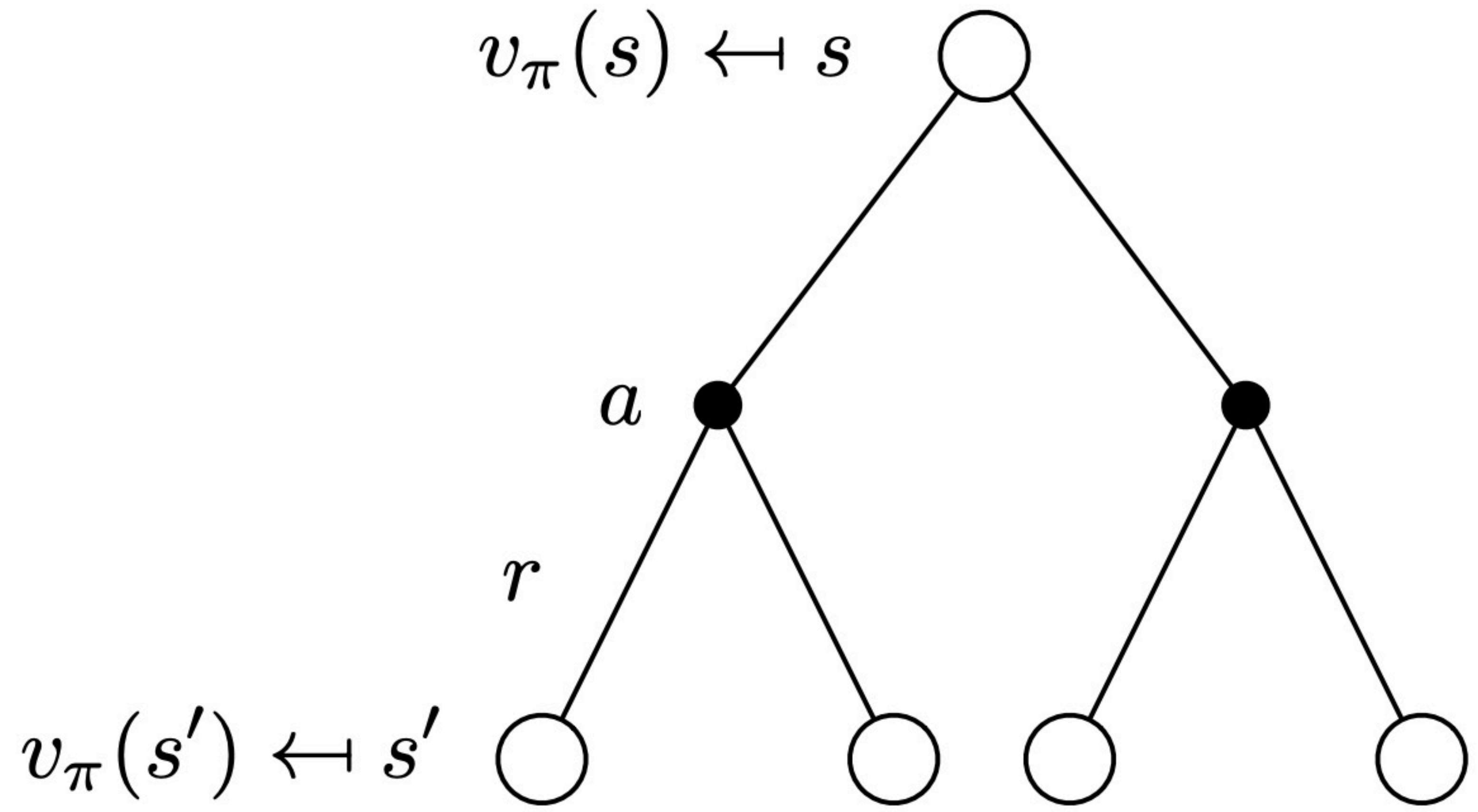
Bellman Expectation Equation (Backup)



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

Markov Decision Processes

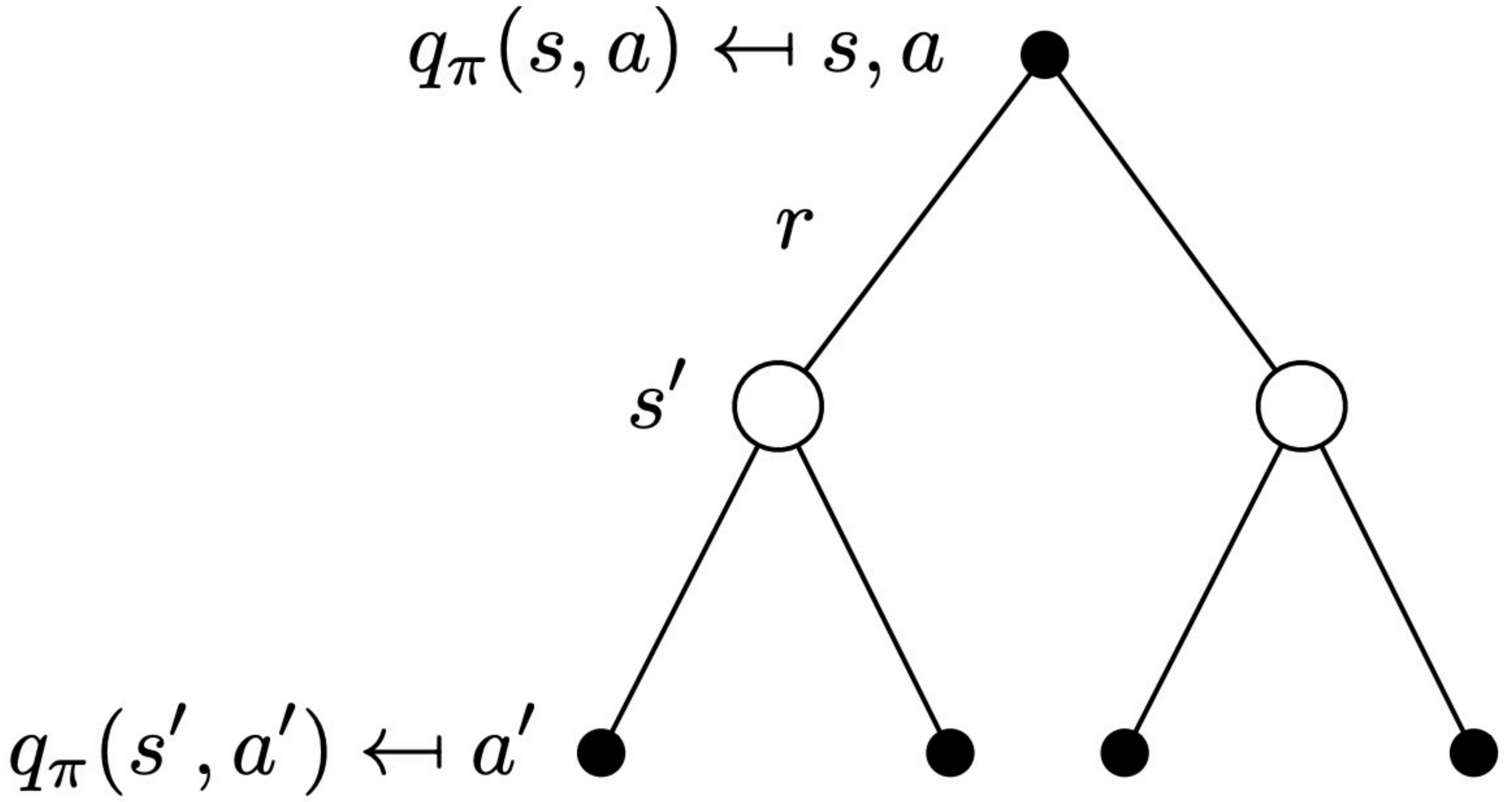
Bellman Expectation Equation (Backup)



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Markov Decision Processes

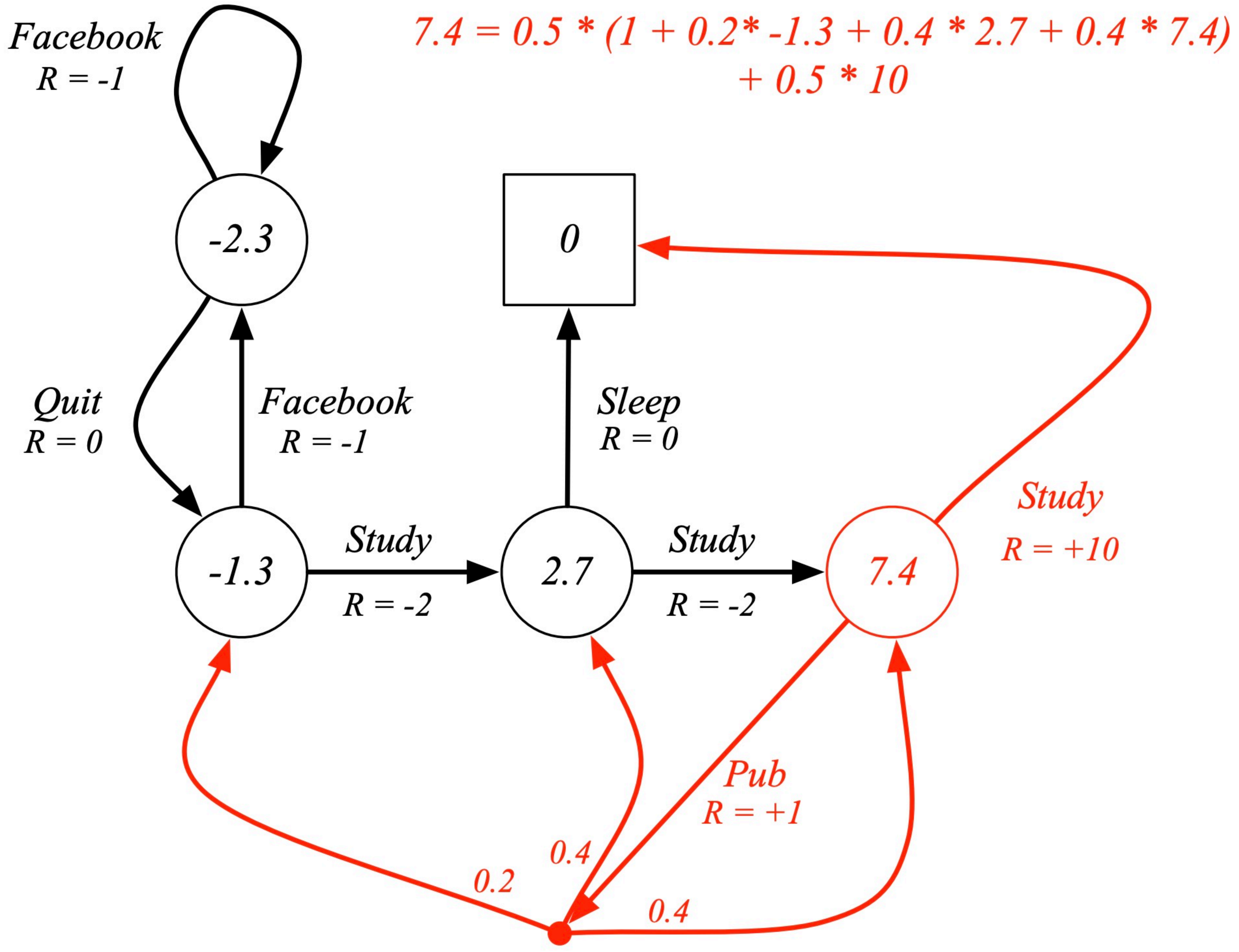
Bellman Expectation Equation (Backup)



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

Markov Decision Processes

Bellman Expectation Equation (Backup)



Markov Decision Processes

Bellman Expectation Equation (matrix form)

- The Bellman expectation equation can be expressed concisely in matrix form:

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi.$$

where v_π is a column vector with one entry per state, i.e.

$$\begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}^\pi_1 \\ \vdots \\ \mathcal{R}^\pi_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}^\pi_{11} & \cdots & \mathcal{P}^\pi_{1n} \\ \vdots & & \vdots \\ \mathcal{P}^\pi_{n1} & \cdots & \mathcal{P}^\pi_{nn} \end{bmatrix} \begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix}.$$

- It is a linear equation which can be solved directly as follows:

$$\begin{aligned} v_\pi &= \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi \\ (\mathbf{I} - \gamma \mathcal{P}^\pi) v_\pi &= \mathcal{R}^\pi \\ v_\pi &= (\mathbf{I} - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi. \end{aligned}$$

Markov Decision Processes

Bellman optimality

- The optimal state-value function $v_*(s)$ is the maximum value function over all policies:

$$v_*(s) = \max_\pi v_\pi(s).$$

- The optimal action-value function $q_*(s, a)$ is the maximum action-value function over all policies:

$$q_*(s, a) = \max_\pi q_\pi(s, a).$$

- The optimal state-value or action-value function specifies the best possible performance in a given MDP.

Markov Decision Processes

Bellman optimality

- We can define a partial ordering over all policies: $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \ \forall s.$
- For any MDP:
  - There always exists an optimal policy π_*:

$$\pi_* \geq \pi, \quad \forall \pi.$$

  - All optimal policies achieve the optimal state-value function:

$$v_{\pi_*}(s) = v_*(s).$$

  - All optimal policies achieve the optimal action-value function:

$$q_{\pi_*}(s, a) = q_*(s, a).$$

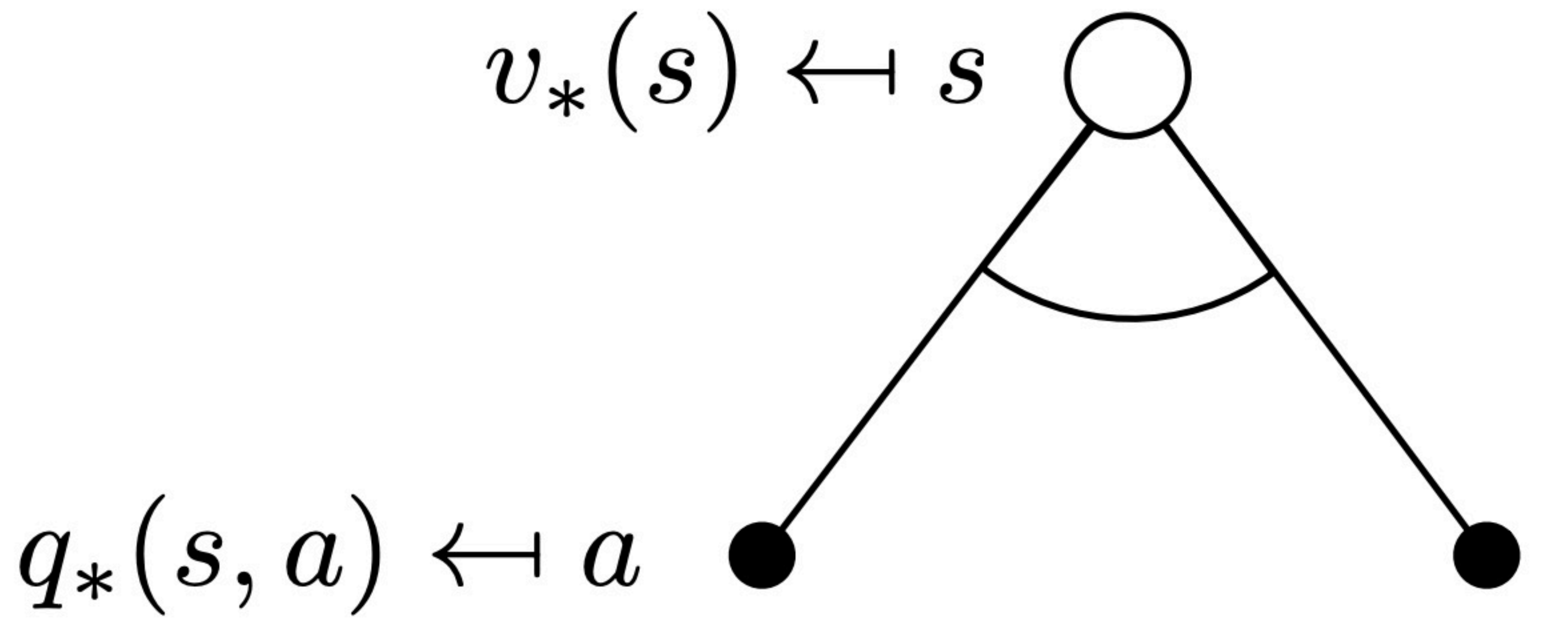
- An optimal policy can be found by maximizing over $q_*(s, a)$:

$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{otherwise.} \end{cases}$$

Markov Decision Processes

Bellman optimality

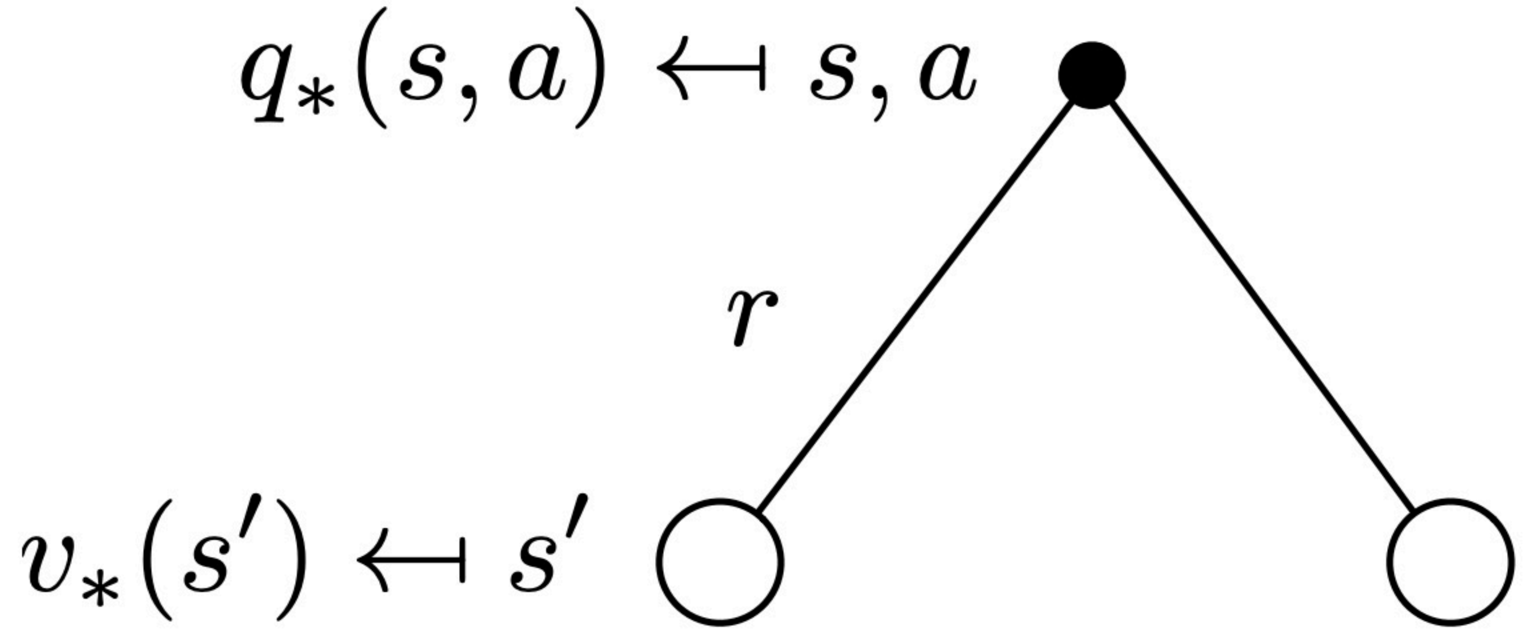
The optimal value functions are recursively related by the Bellman optimality equations:



$$v_*(s) = \max_a q_*(s, a)$$

Markov Decision Processes

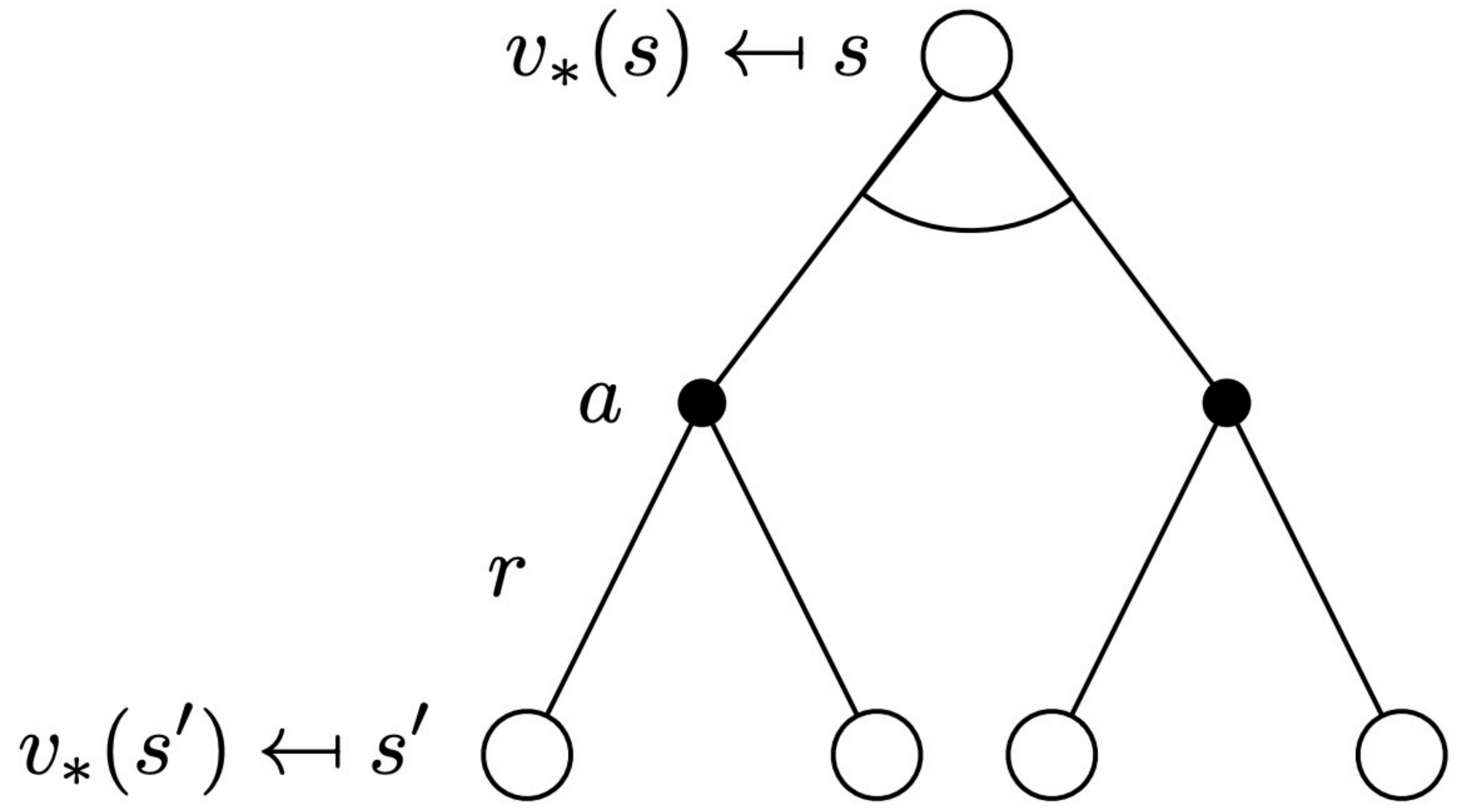
Bellman optimality



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

Markov Decision Processes

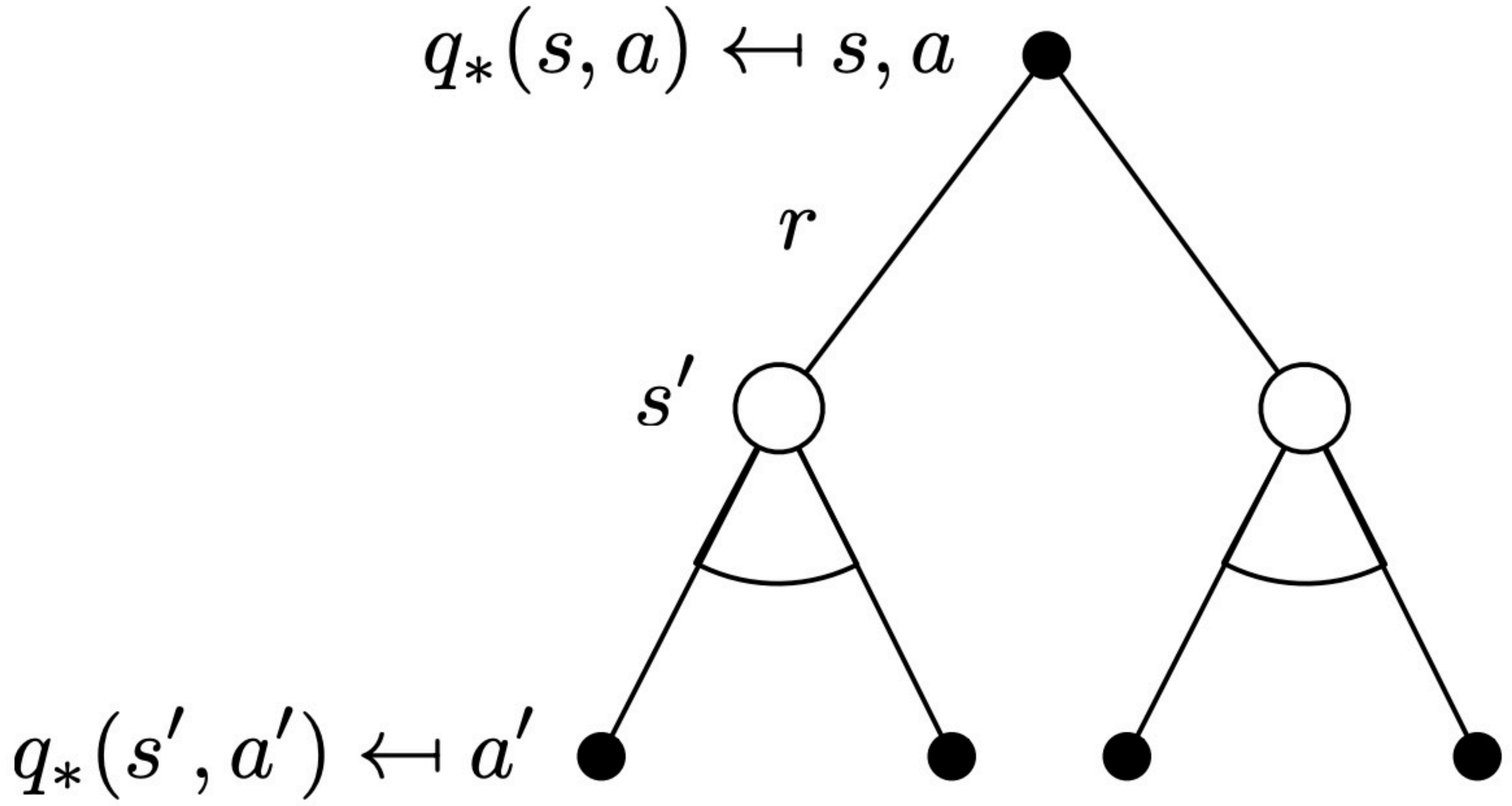
Bellman optimality



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

Markov Decision Processes

Bellman optimality



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

Markov Decision Processes

Solving the bellman optimality equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
    - Value Iteration
    - Policy Iteration
    - Q-learning
    - Sarsa

Markov Decision Processes

Iteration by Dynamic programming

- Classical dynamic programming (DP) algorithms compute optimal policies, assuming full knowledge of the underlying MDP and the existence of sufficient computational resources.
- Note that this setting is of limited utility in most realistic reinforcement learning problems, but it is important theoretically and provides useful insights for more practical reinforcement learning algorithms.
- The key idea of DP, and of reinforcement learning in general, is the use of value functions to organize and structure the search for good policies.
- The optimal policies can be obtained after finding the optimal value functions v_* or q_*:

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

$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a').$$

Markov Decision Processes

Iteration by Dynamic programming

- DP algorithms make use of update rules obtained by turning the equality $(=)$ in the Bellman optimality equations into assignment $(\leftarrow)$, which aims to iteratively improve approximations of the desired value functions.
- For example, we want to construct a sequence $\{v_k\}$ that converges asymptotically to v_* based on the following update rule:

$$v_{k+1}(s) \leftarrow \max_a \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v_k(s') \right\} .$$

- This update rule is used in the value iteration algorithm.

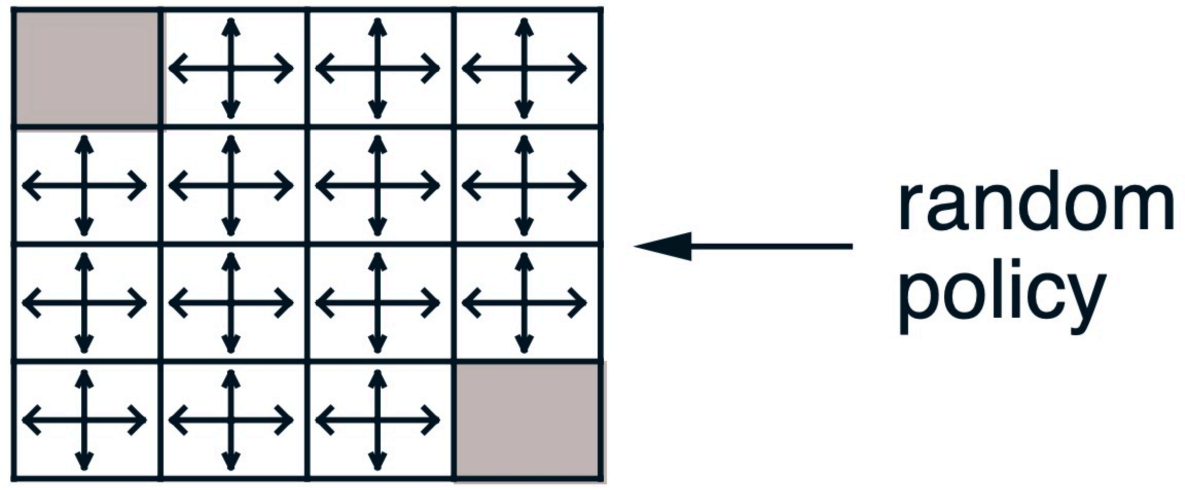
Markov Decision Processes

Value Iteration by Dynamic programming

v_k for the Random Policy | Greedy Policy w.r.t. v_k

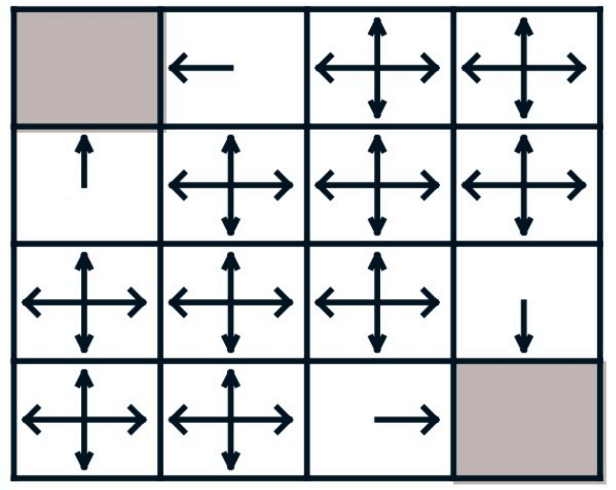
$k = 0$

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



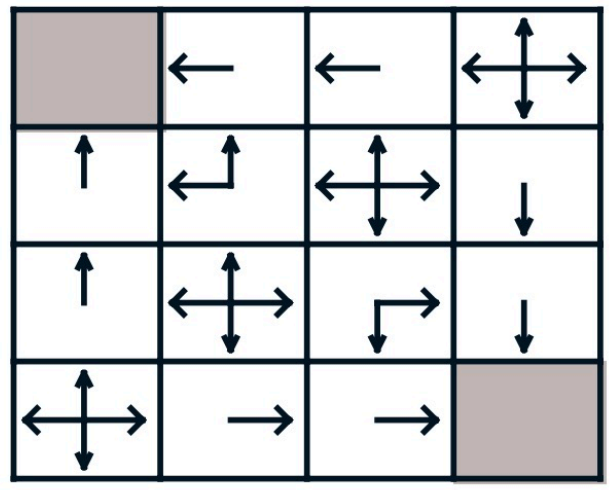
$k = 1$

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



Markov Decision Processes

Value Iteration by Dynamic programming

$k = 3$

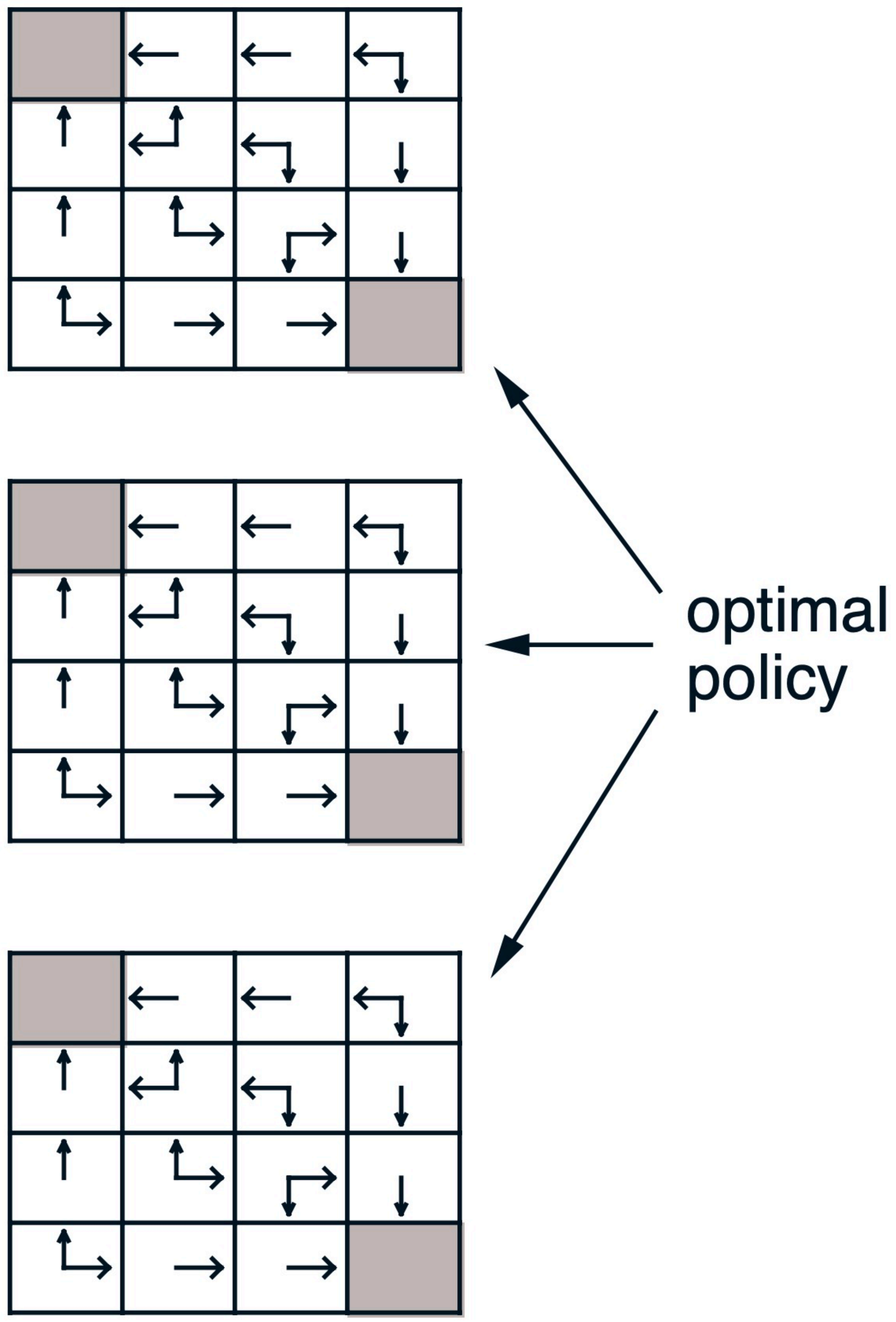
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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

$k = \infty$

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



Markov Decision Processes

Value Iteration by Dynamic programming

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
$v(s) \leftarrow 0$ for all states s
repeat
  $\delta \leftarrow 0$
  for each state s **do**
    $v_{prev} \leftarrow v(s)$
    $v(s) \leftarrow \max_a \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v(s') \right\}$
    $\delta \leftarrow \max(\delta, |v_{prev} - v(s)|)$
  end for
until $\delta < \theta$
Output a deterministic policy $\pi \approx \pi_*$ s.t. $\pi(s) = \arg\max_a \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v(s') \right\}$

Markov Decision Processes

Value Iteration by Dynamic programming

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
$q(s, a) \leftarrow 0$ for all state-action pairs (s, a)
repeat
  $\delta \leftarrow 0$
  for each state-action pair (s, a) **do**
    $q_{prev} \leftarrow q(s, a)$
    $q(s, a) \leftarrow \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q(s', a')$
    $\delta \leftarrow \max(\delta, |q_{prev} - q(s, a)|)$
  end for
until $\delta < \theta$
Output a deterministic policy $\pi \approx \pi_*$ s.t. $\pi(s) = \arg\max_a q(s, a)$

- Note that determining the optimal policy from q is easier than v.

Model-Free Reinforcement Learning

- Unlike DP algorithms for solving MDPs, model-free reinforcement learning algorithms assume no knowledge of $\mathcal{P}$ and $\mathcal{R}$.
- For some problems, the MDP is actually known but is too big to apply DP algorithms.
- Model-free reinforcement learning algorithms aim to learn directly from episodes of experience interacting with the environment, requiring sufficient exploration in addition to exploitation.
- We will consider both tabular methods for discrete states and actions and function approximation methods for the continuous extension.

Model-Free Reinforcement Learning

Q-Learning

$q(s, a) \leftarrow 0$ for all state-action pairs (s, a)
repeat
  Initialize s
  repeat
    Choose action a at state s according to some strategy
    Take action a at state s, then observe s' and r
    $q(s, a) \leftarrow q(s, a) + \alpha\left[r + \gamma \max_{a'} q(s', a') - q(s, a)\right]$
    $s \leftarrow s'$
  until s is a terminal state
until convergence

Model-Free Reinforcement Learning

Q-Learning

- Q-learning target based on a one-step look-ahead:

$$r + \gamma \max_{a'} q(s', a').$$

- Update rule based on temporal-difference (TD) learning:

$$q(s, a) \leftarrow q(s, a) + \alpha \left[r + \gamma \max_{a'} q(s', a') - q(s, a) \right],$$

  where α is the learning rate.

- It has been shown that the Q-learning update rule converges to the optimal action-value function, i.e., $q(s, a) \to q_*(s, a)$.