05 Markov Decision Processes (MDPs)

Table of contents

Introduction to Markov Decision Processes (MDPs)

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, e.g.

- Optimal control primarily deals with continuous MDPs
- Partially observable problems can be converted into MDPs

Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with *decisions*.

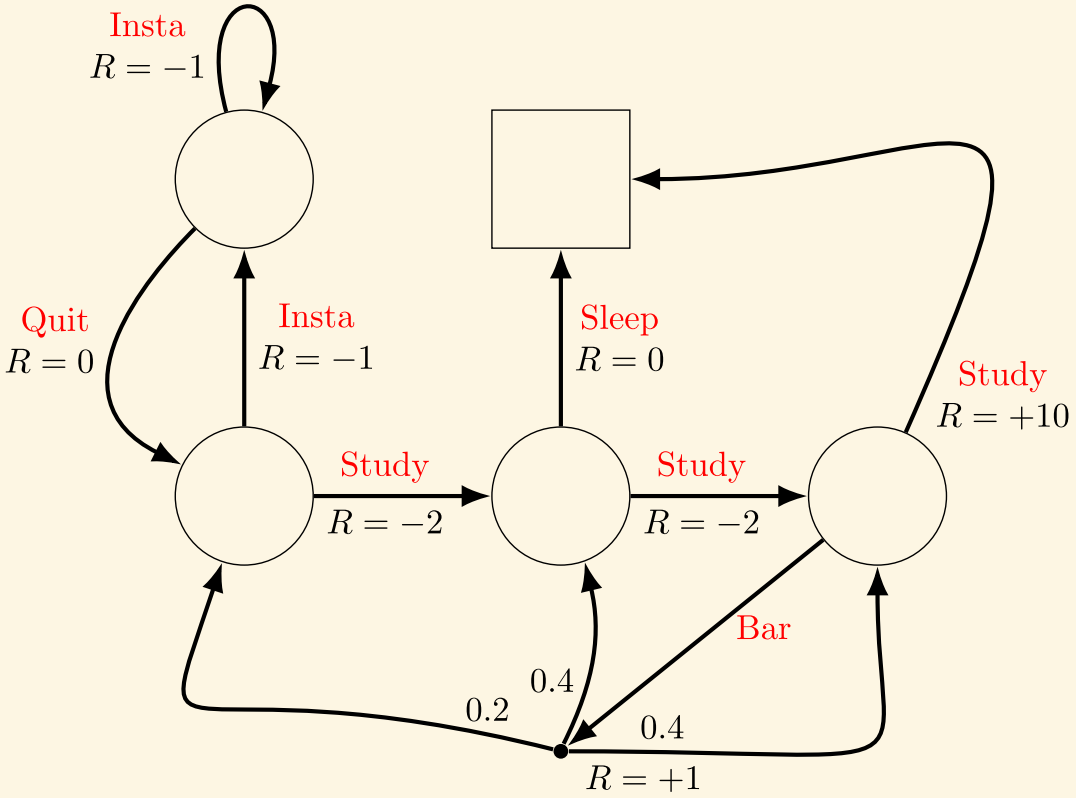
- It is an *environment* in which all states are Markov.
- We introduced agency in terms of *actions*.

Definition

A Markov Decision Process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

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

Example: Student MDP



Agent exerts control over MDP via actions, and goal is to find the best path through decision making process to maximise rewards

Policies (1)

Definition

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

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

- 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),
  $A_t \sim \pi(\cdot \mid S_t), \quad \forall t > 0$

Policies (2) - Recovering Markov reward process from MDP

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π

- The state sequence $S_1, S_2, \ldots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$, where

$$\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, \mathcal{P}^a_{ss'}$$

$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a \mid s)\, \mathcal{R}^a_s$$

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

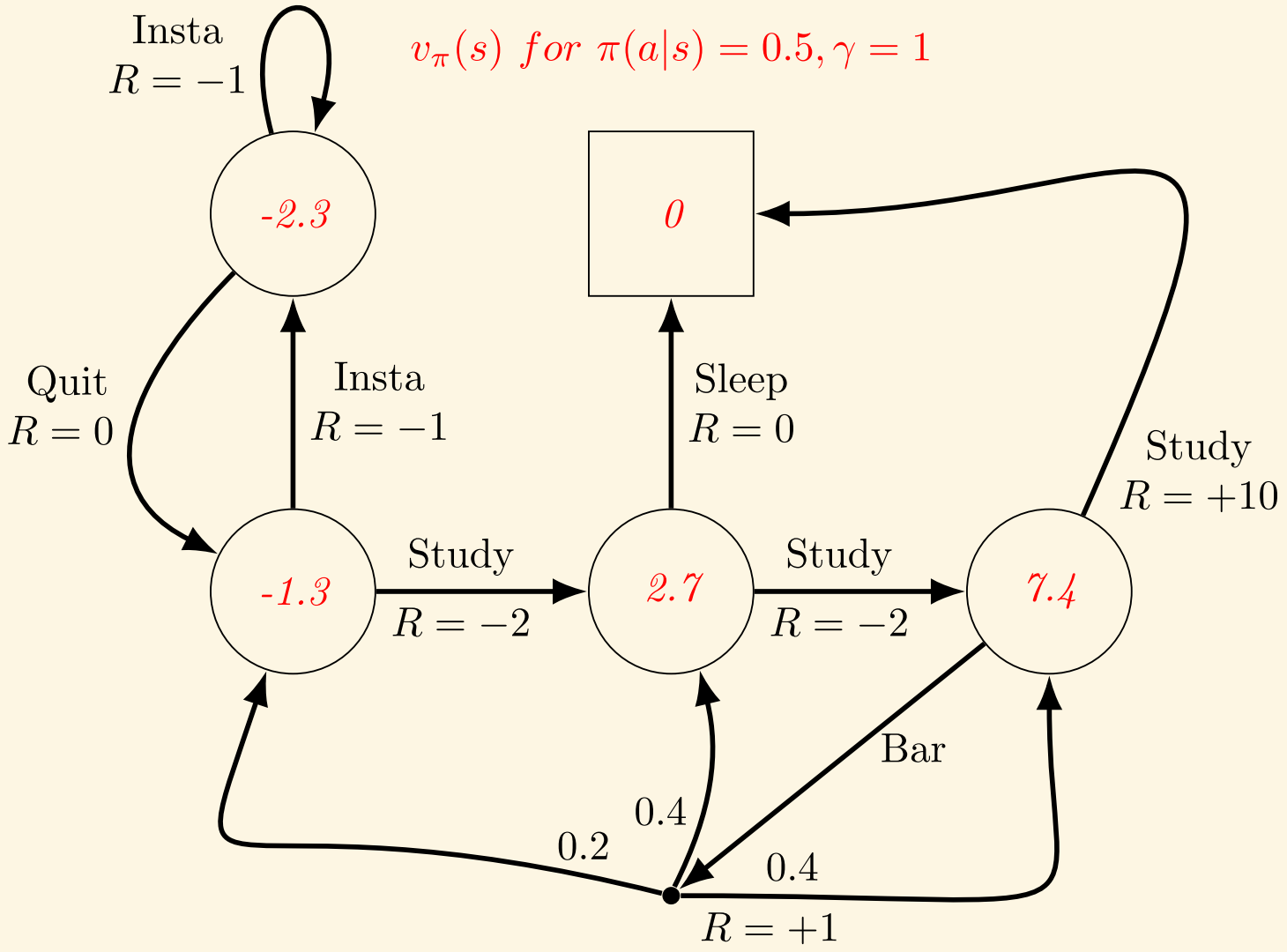
$$v_\pi(s) = \mathbb{E}_\pi[\, G_t \mid S_t = s \,]$$

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

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

Example: State-Value Function for Student MDP



Bellman Expectation Equation

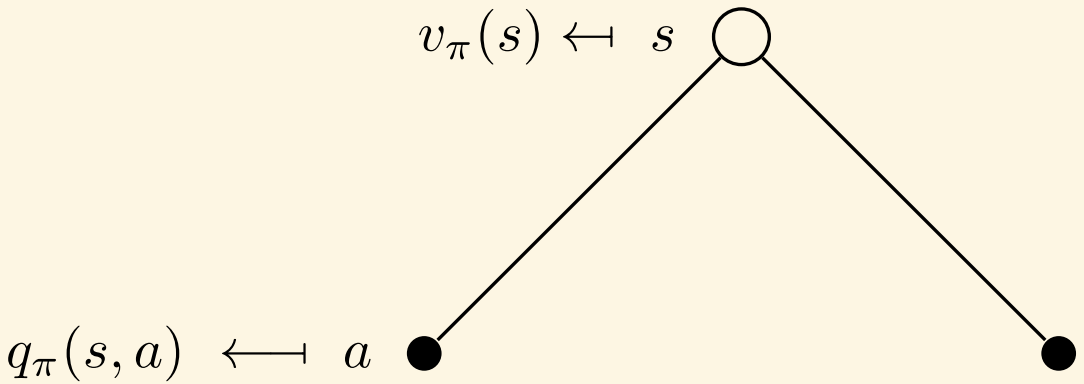
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi[\, R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \,]$$

Can do the same thing for the q values: the action-value function can similarly be decomposed,

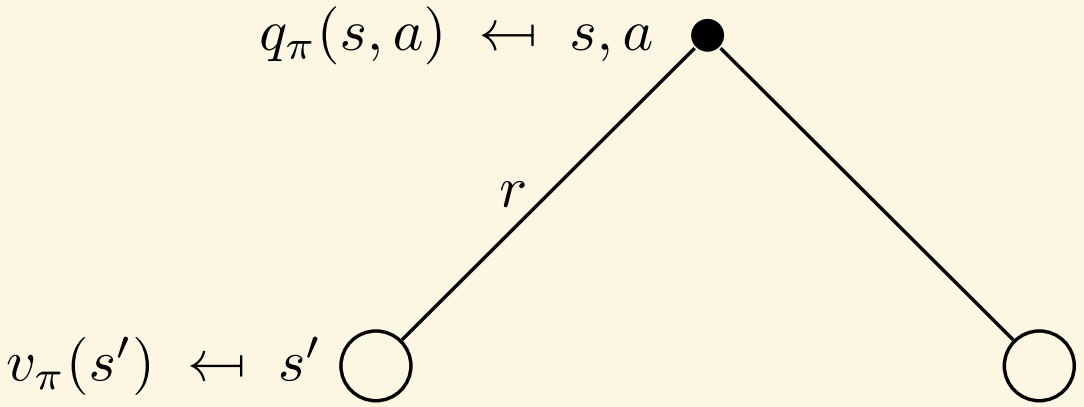
$$q_\pi(s, a) = \mathbb{E}_\pi[\, R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s,\ A_t = a \,]$$

Bellman Expectation Equation for V^π (look ahead)



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

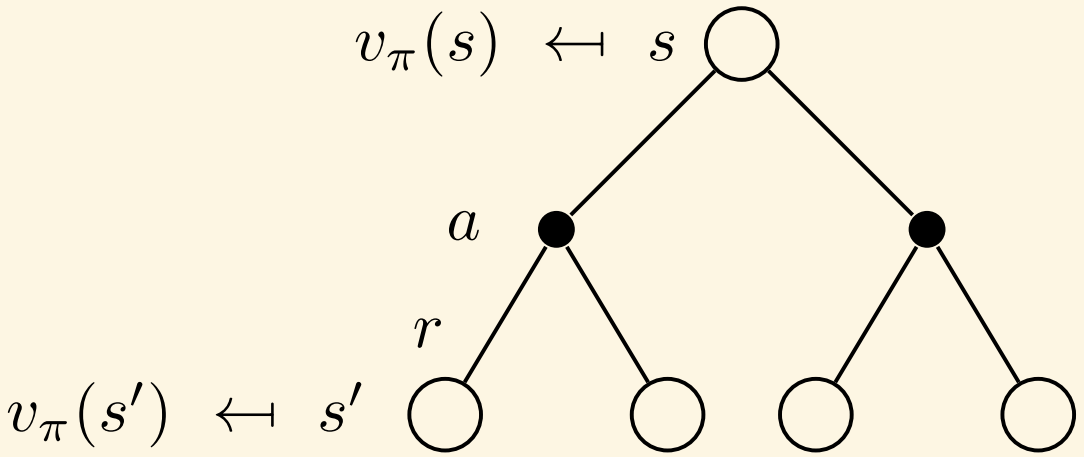
Bellman Expectation Equation for Q^π (look ahead)



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

Bellman Expectation Equation for $v_\pi(2)$

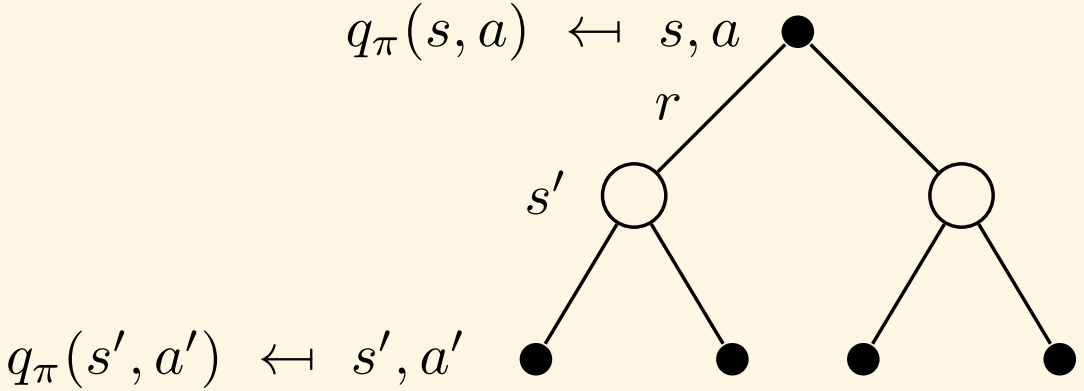
Bringing it together: agent actions (open circles), environment actions (closed circles)



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

Bellman Expectation Equation for $q_\pi(2)$

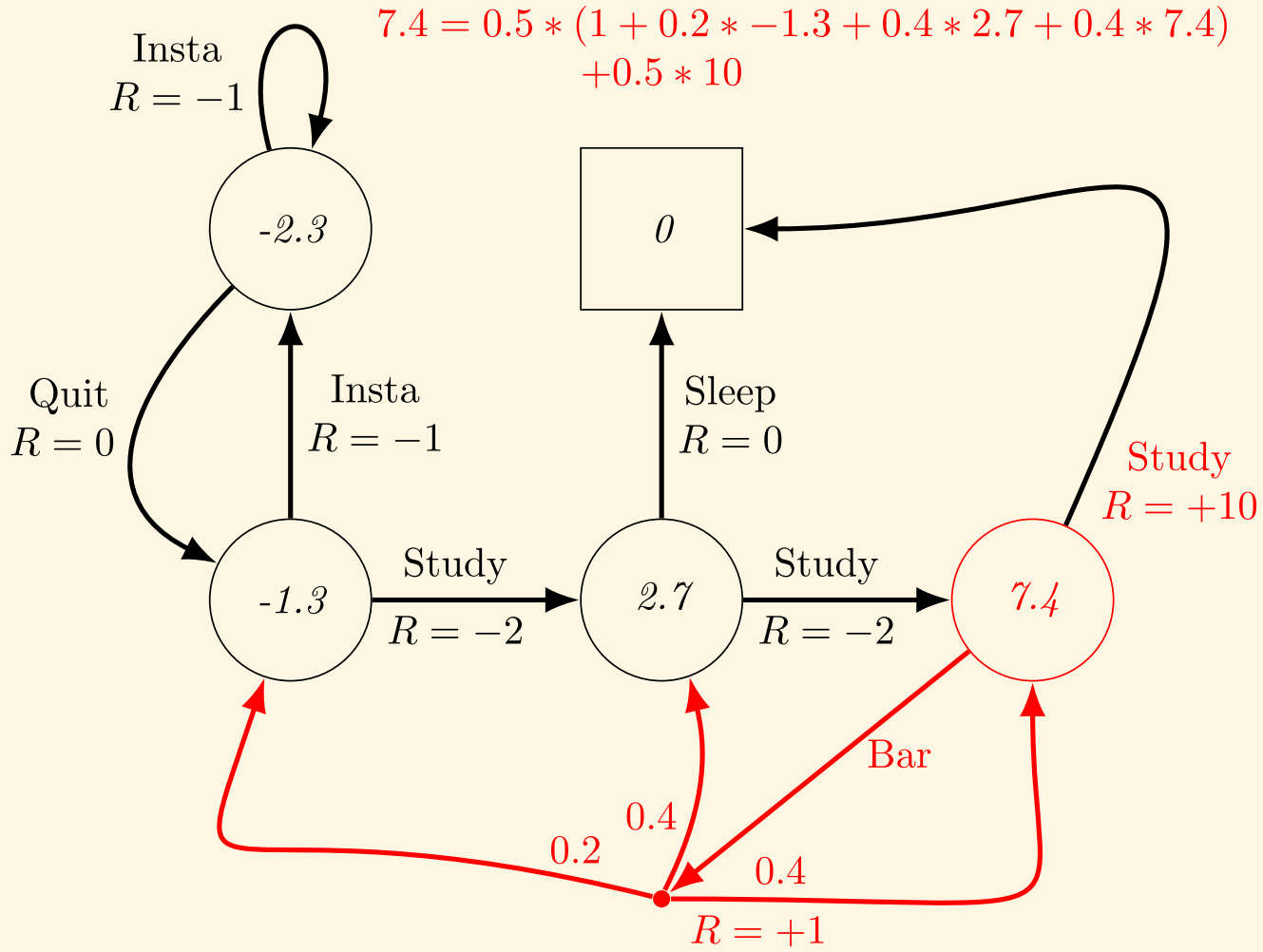
The other way around: can do same thing for action values



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

In both forms value function is (recursively) equal to reward of immediate state s + value s' (where you end up)

Example: Bellman Expectation Equation in Student MDP



Verify Bellman Equation to compute $v_\pi(s)$ for $s = C3$

Bellman Expectation Equation (Matrix Form)

The Bellman expectation equation can be expressed concisely using the induced MRP (as before),

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

with direct solution

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

- Bellman equation gives us description of system can solve
- Essentially *averaging* then computing inverse, although inefficient!

Optimal Value Function

Optimal Value Function (Finding the best behaviour)

Definition

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

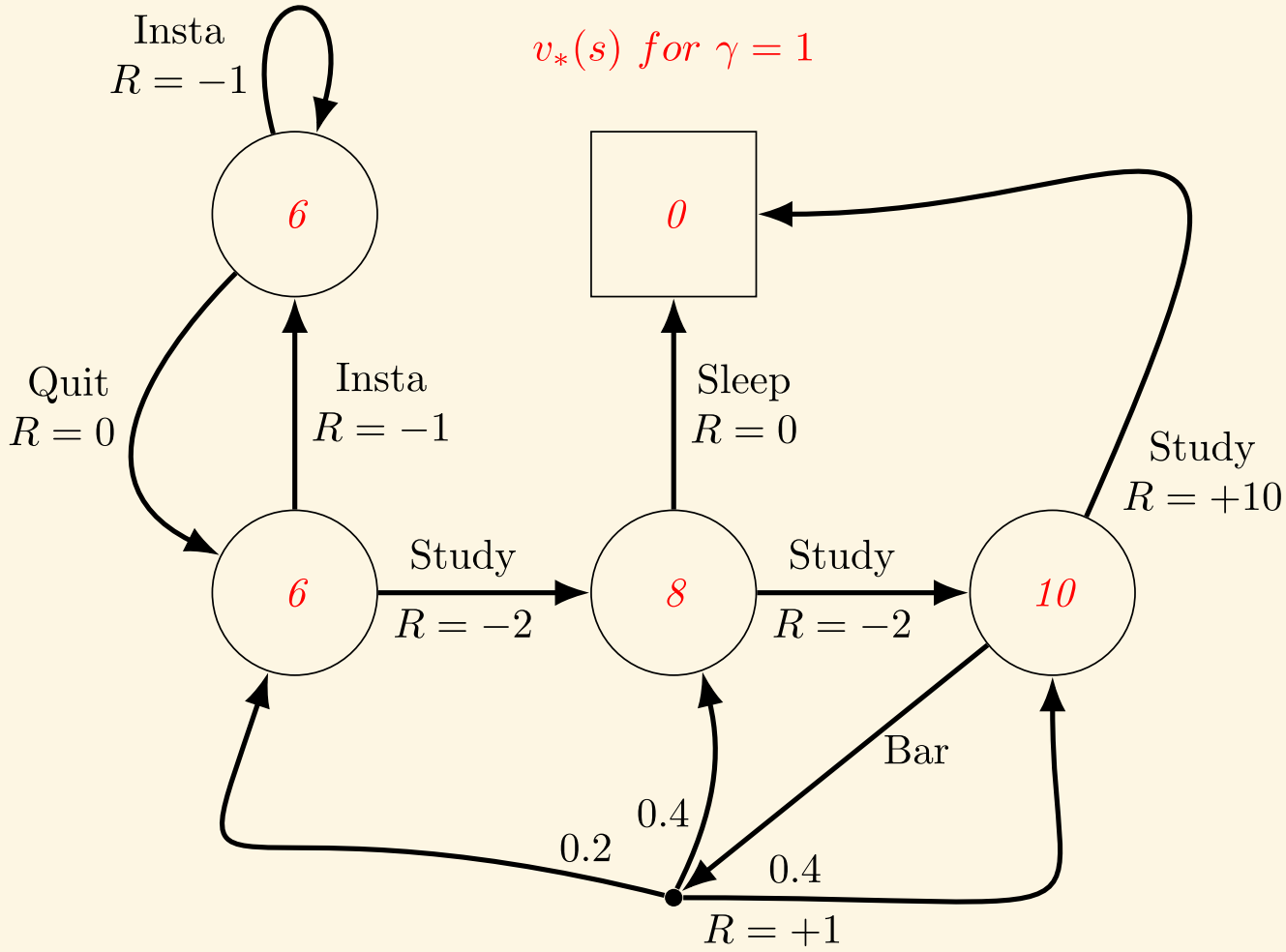
$$v_*(s) = \max_\pi v_\pi(s)$$

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

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

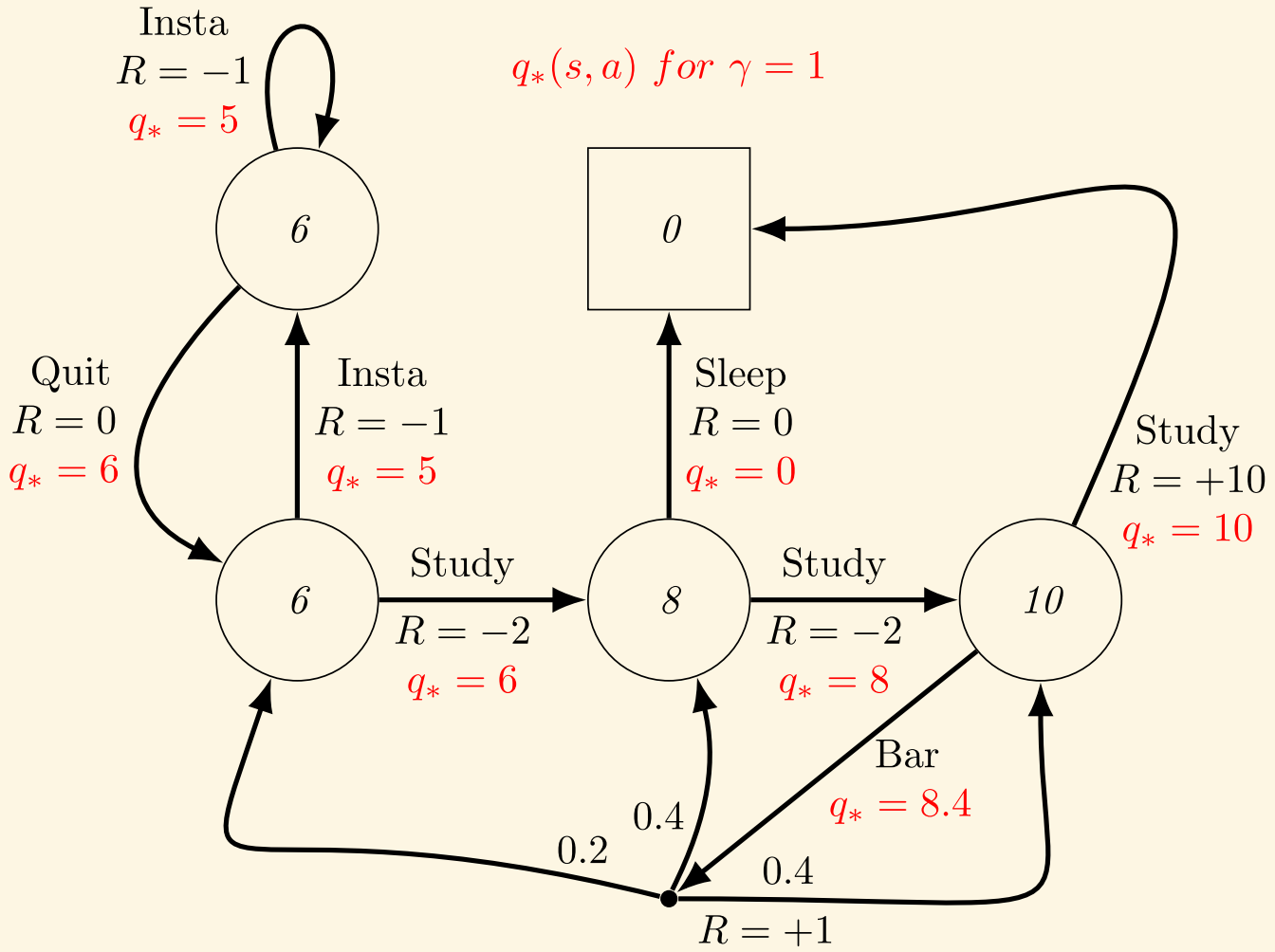
- The optimal value function specifies the best possible performance in the MDP.
- An MDP is “solved” when we know the optimal value function.
- If you know q_*, you have the optimal value function
- So solving means finding q_*

Example: Optimal Value Function for Student MDP



Gives us value function for each state s (not how to behave)

Example: Optimal Action-Value Function for Student MDP



Gives us best action, a, for each state s (can choose)

Optimal Policy

Define a *partial ordering* over policies

$$\pi \geq \pi' \quad \text{if } v_\pi(s) \geq v_{\pi'}(s), \ \forall s$$

Theorem For any Markov Decision Process

- There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \ \forall \pi$
- All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

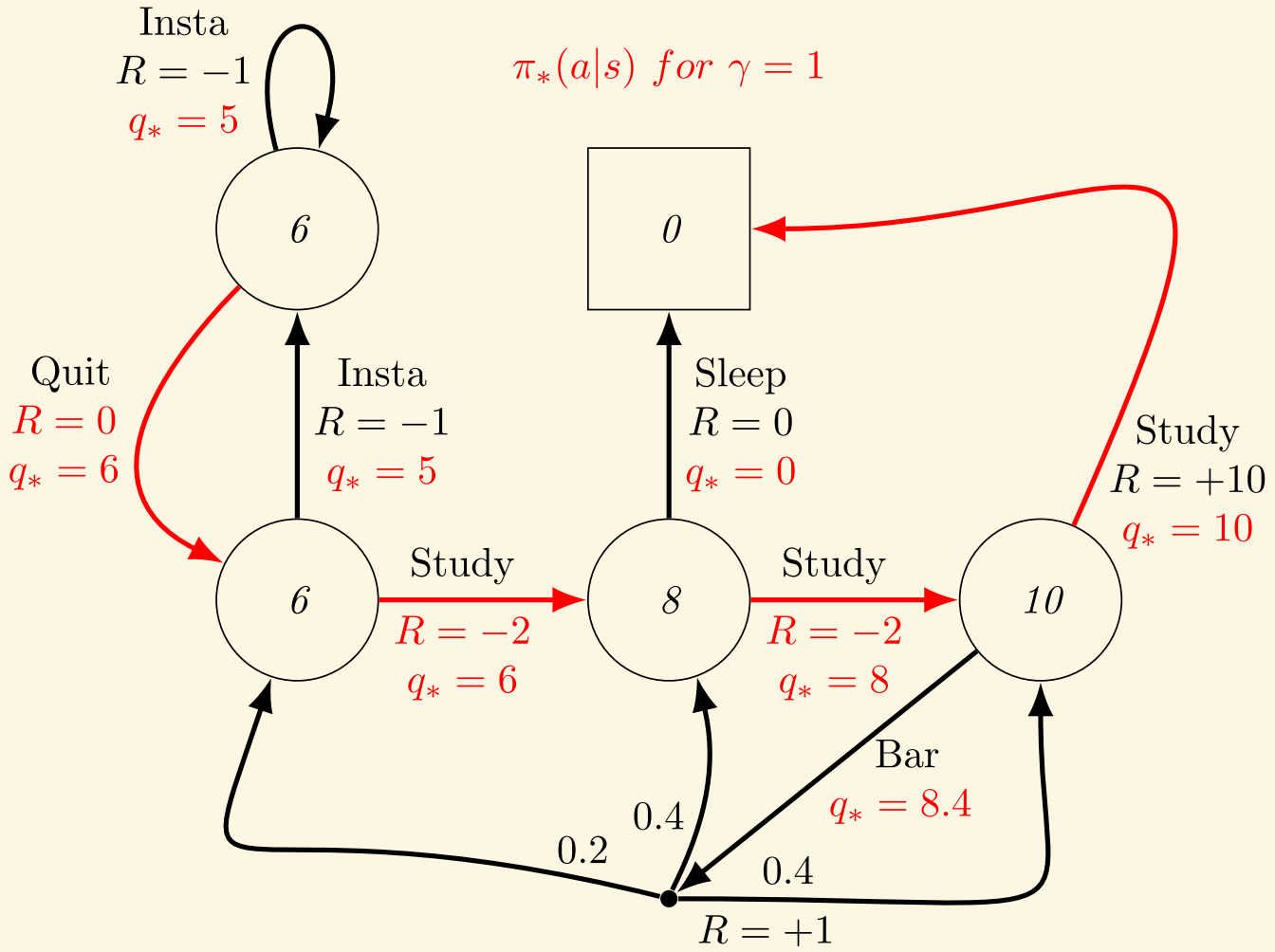
Finding an Optimal Policy

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

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

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

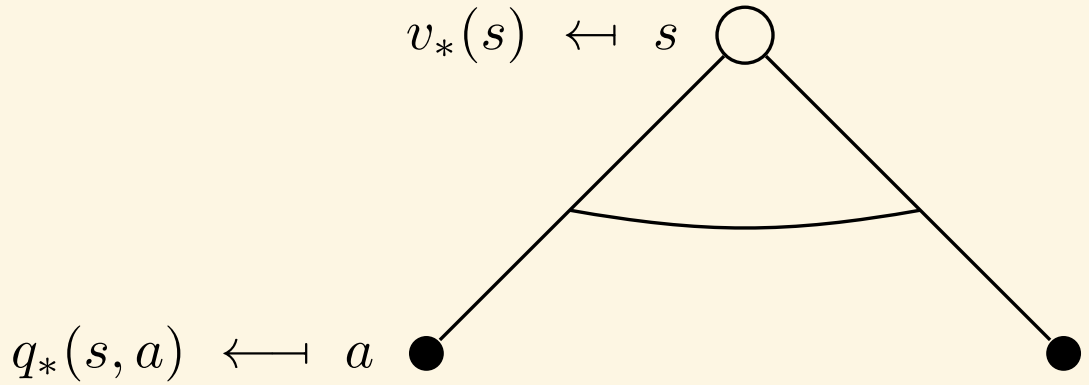
Example: Optimal Policy for Student MDP



Red arcs (actions) represent optimal policy: picks highest q_*

Bellman Optimality Equation for v_* (look ahead)

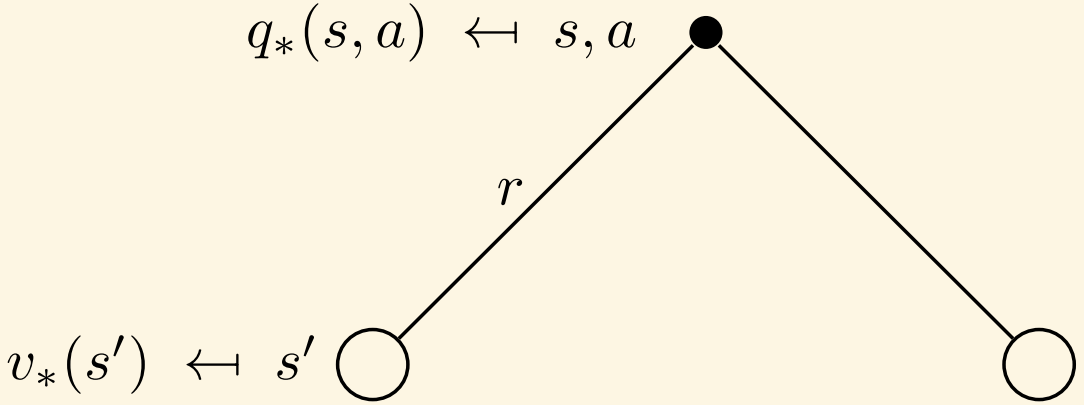
The optimal value functions are recursively related by the Bellman optimality equations:



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

Working backwards using backup diagrams we get $v_*(s)$ - best action over all policies taking *max* instead of average

Bellman Optimality Equation for Q^* (look ahead)

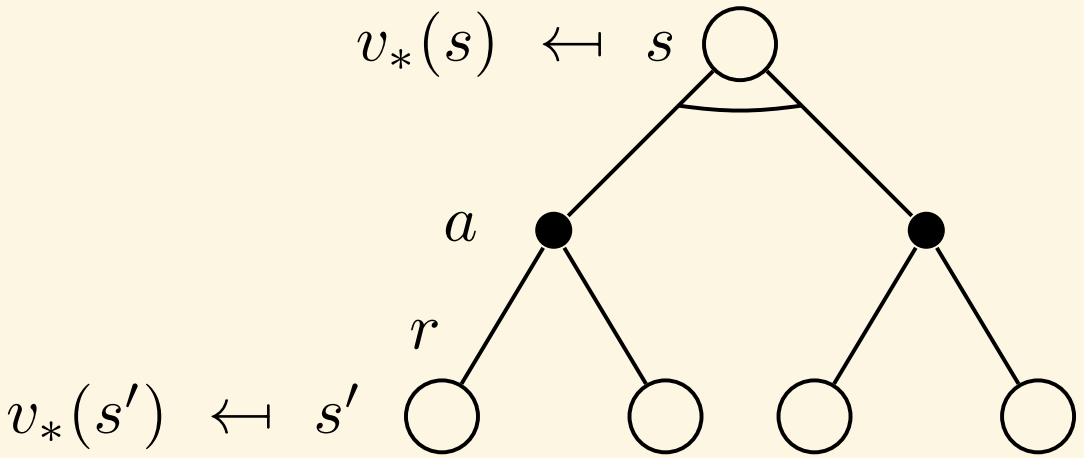


$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v_*(s')$$

Considers where the environment might take us by averaging (looking ahead) and backing up (inductively)

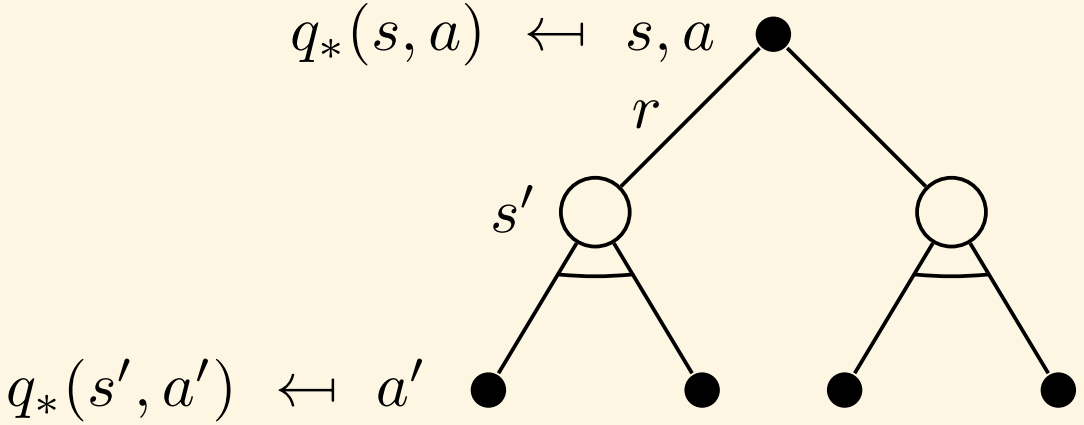
Bellman Optimality Equation for $V^*(2)$

Bringing it together (two-step look ahead): agent actions (open circles), environment actions (closed circles)



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

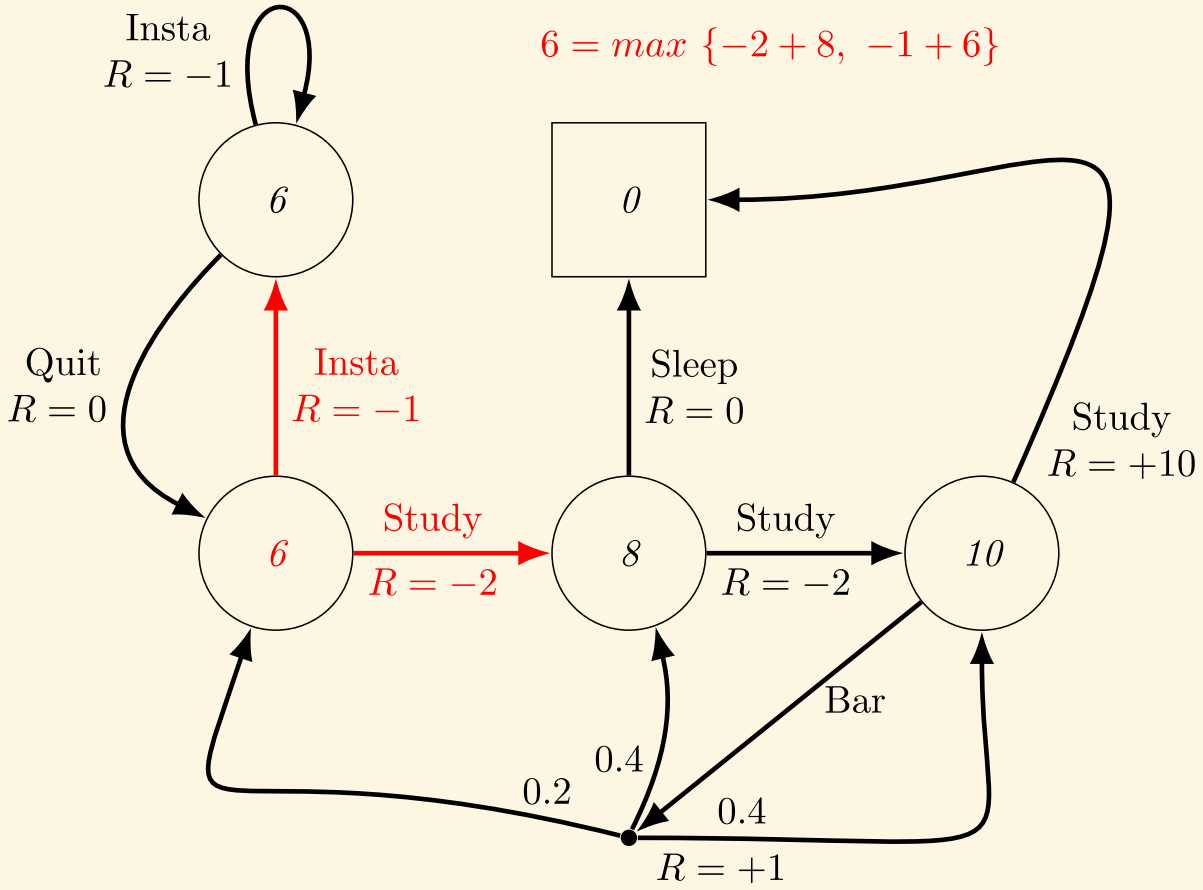
Bellman Optimality Equation for $Q^*(2)$



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

Determines Q^* reordering from environments perspective

Example: Bellman Optimality Equation in Student MDP



Compute $v_*(s)$ for $s = C1$ looking one step ahead (no environment actions in $C1$)

Solving the Bellman Optimality Equation

Bellman Optimality Equation is non-linear

- No closed form solution (in general)

Many iterative solution methods

- Value Iteration (not covered in this subject)
- Policy Iteration (not covered in this subject)
- Q-learning (covered in Module 9)
- SARSA (covered in Module 9)

Model-Based Reinforcement Learning

Last Module: Learning & Relxation

This Module: learn model directly from experience

- . . . and use planning to construct a value function or policy

Integrates learning and planning into a single architecture

Model-Based Reinforcement Learning

Model-Based and Model-Free RL

Model-Free RL

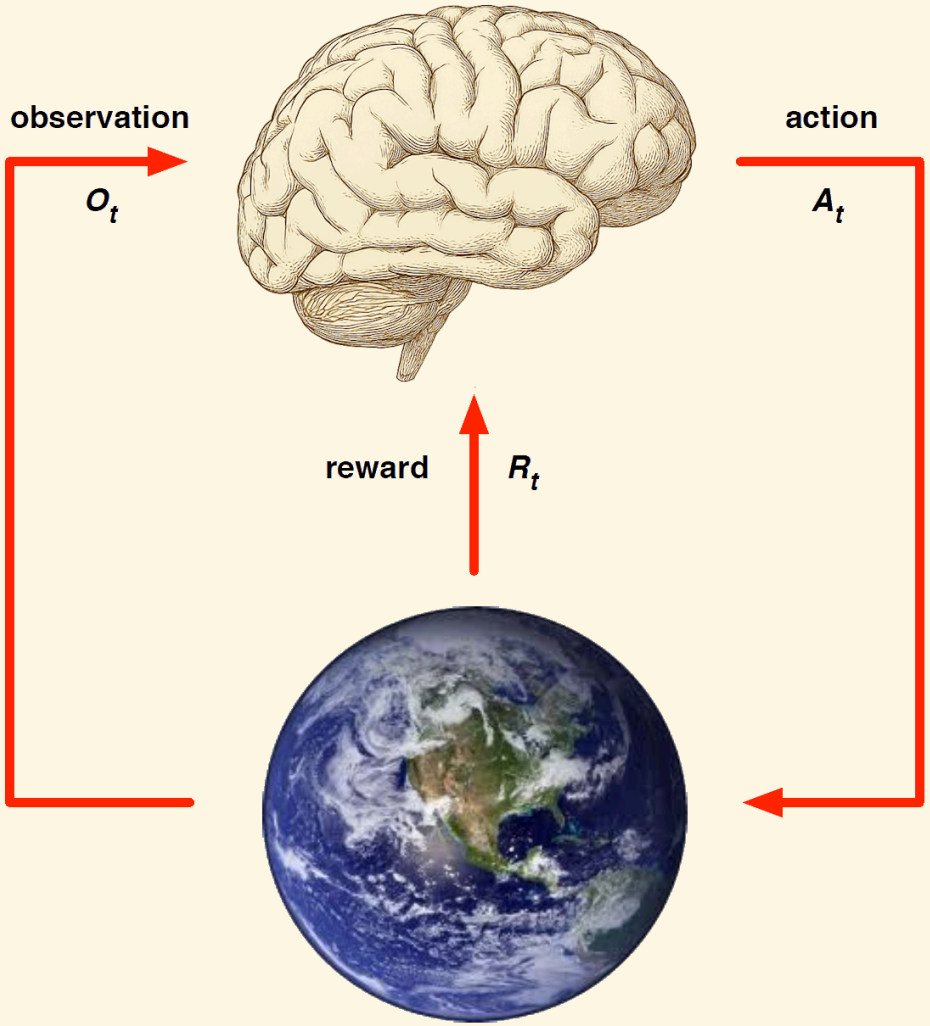
- No model
- Learn value function (and/or policy) from experience

Model-Based RL

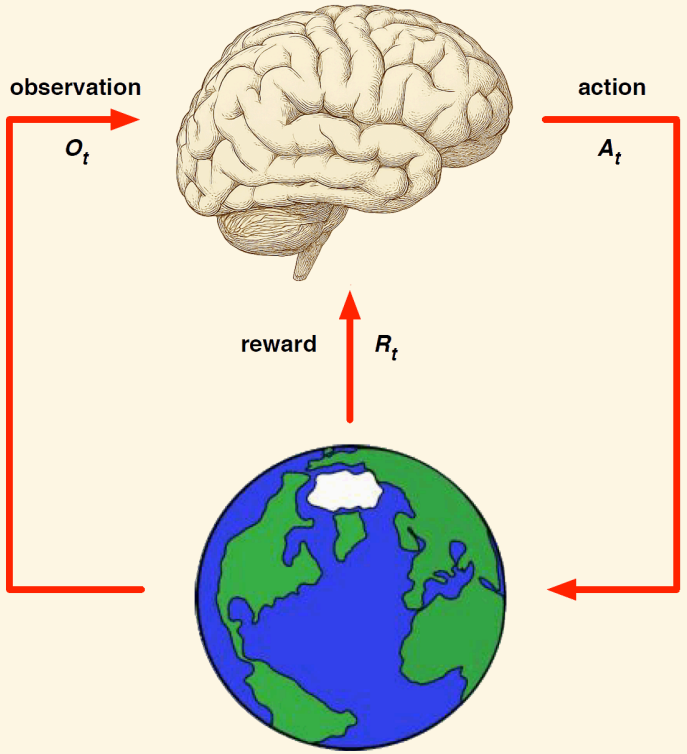
- Learn a model from experience
- Plan value function (and/or policy) from model

Lookahead by planning (*or thinking*) about what the value function will be

Model-Free RL



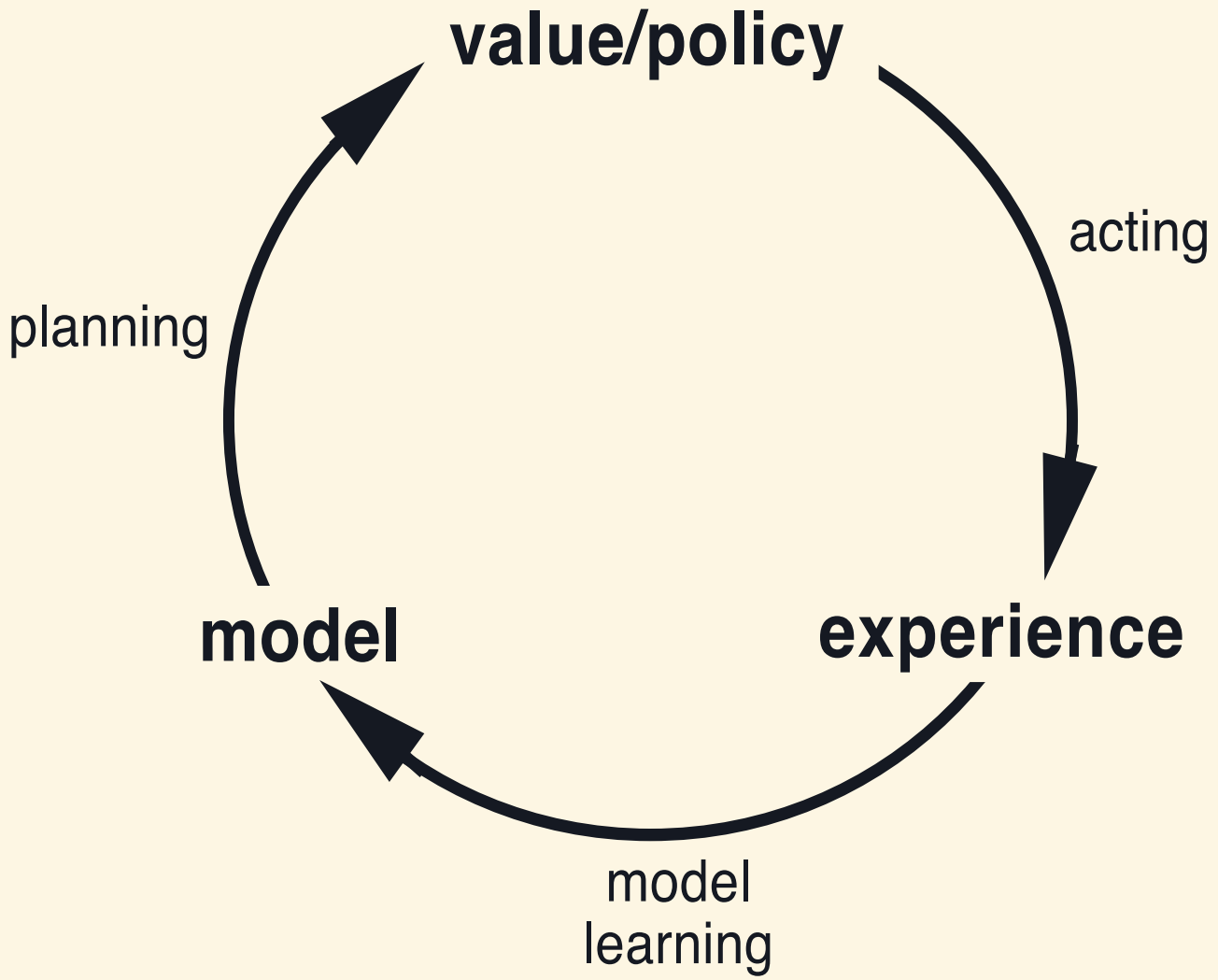
Model-Based RL



Replace real world with the agent's (simulated) *model* of the environment

- Supports rollouts (lookaheads) under imagined actions to reason about what value function will be, without further environment interaction

Model-Based RL (2)



Advantages of Model-Based RL

Advantages:

- Can efficiently learn model by supervised learning methods
- Can reason about model uncertainty, and even take actions to reduce uncertainty

Disadvantages:

- First learn a model, then construct a value function

  $\Rightarrow$ two sources of approximation error

Learning a Model

What is a Model?

A *model* $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, *parameterised* by η

- We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known

So a model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} \mid S_t, A_t)$$

$$R_{t+1} = \mathcal{R}_\eta(R_{t+1} \mid S_t, A_t)$$

Typically assume conditional independence between state transitions and rewards

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

Note you can learn from each (one-step) transition, treating the following step as the supervisor for the prior step.

Model Learning

Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, \ldots, S_T\}$

This is a supervised learning problem

$$
\begin{aligned}
S_1, A_1 &\rightarrow R_2, S_2 \\
S_2, A_2 &\rightarrow R_3, S_3 \\
&\vdots \\
S_{T-1}, A_{T-1} &\rightarrow R_T, S_T
\end{aligned}
$$

- Learning $s, a \rightarrow r$ is a *regression* problem
- Learning $s, a \rightarrow s'$ is a *density estimation* problem
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimise empirical loss

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- · · · almost any supervised learning model

Table Lookup Model

Model is an explicit MDP, $\hat{\mathcal{P}}, \hat{\mathcal{R}}$

- Count visits $N(s, a)$ to each state–action pair (parametric approach)

$$\hat{\mathcal{P}}^{a}_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

$$\hat{\mathcal{R}}^{a}_{s} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a)\, R_t$$

- Alternatively (a simple non-parametric approach)
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

AB Example (Revisited) - Building a Model

Two states A, B; no discounting; 8 episodes of experience

$A, 0, B, 0$

$B, 1$

$B, 1$

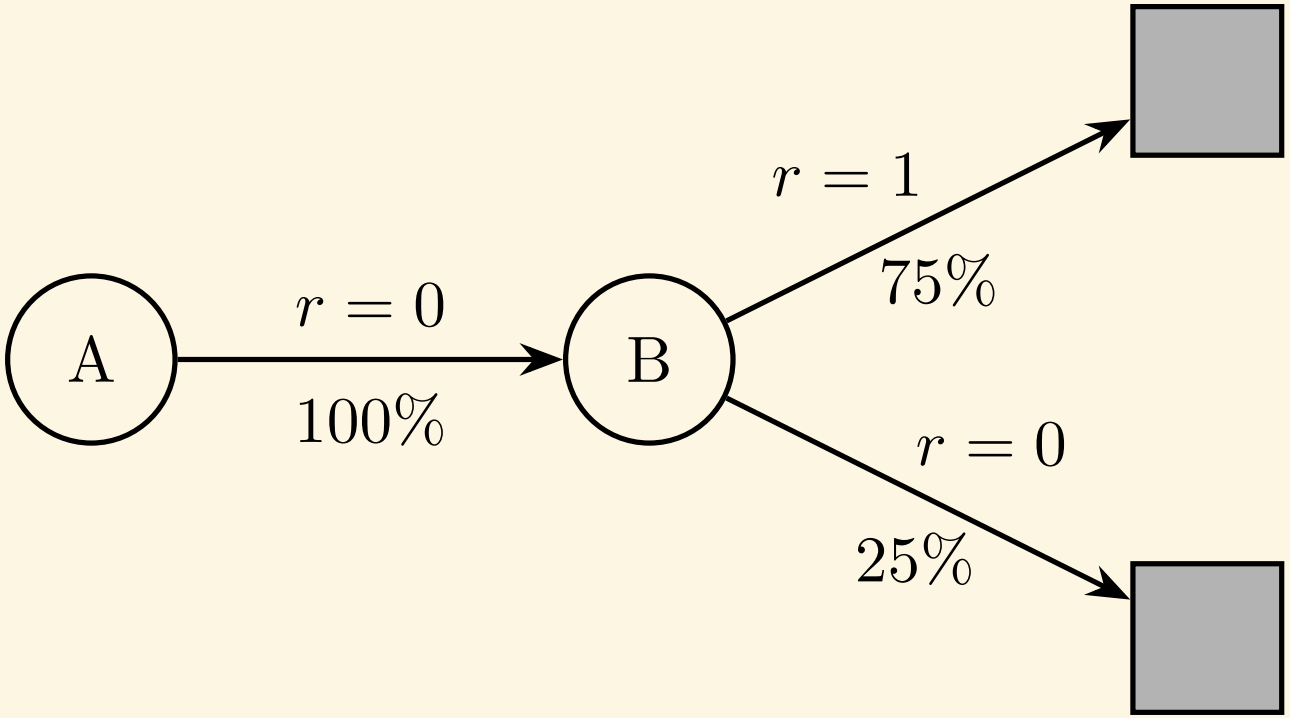
$B, 1$

$B, 1$

$B, 1$

$B, 1$

$B, 0$



We have constructed a table lookup model from the experience