Markov Decision Processes

CMPUT 261: Introduction to Artificial Intelligence

S&B §3.0-3.5

Lecture Outline

1. Recap & Logistics
2. Markov Decision Processes
3. Returns & Episodes
4. Policies & Value Functions
5. Bellman Equations

After this lecture, you should be able to:

- define a Markov decision process
- represent a problem as a Markov decision process
- define a policy
- explain whether a task is episodic or continuing
- give expressions for the state-value function and the action-value function
- state the Bellman optimality equations
- give expressions for episodic and discounted continuing returns

Assignment #3

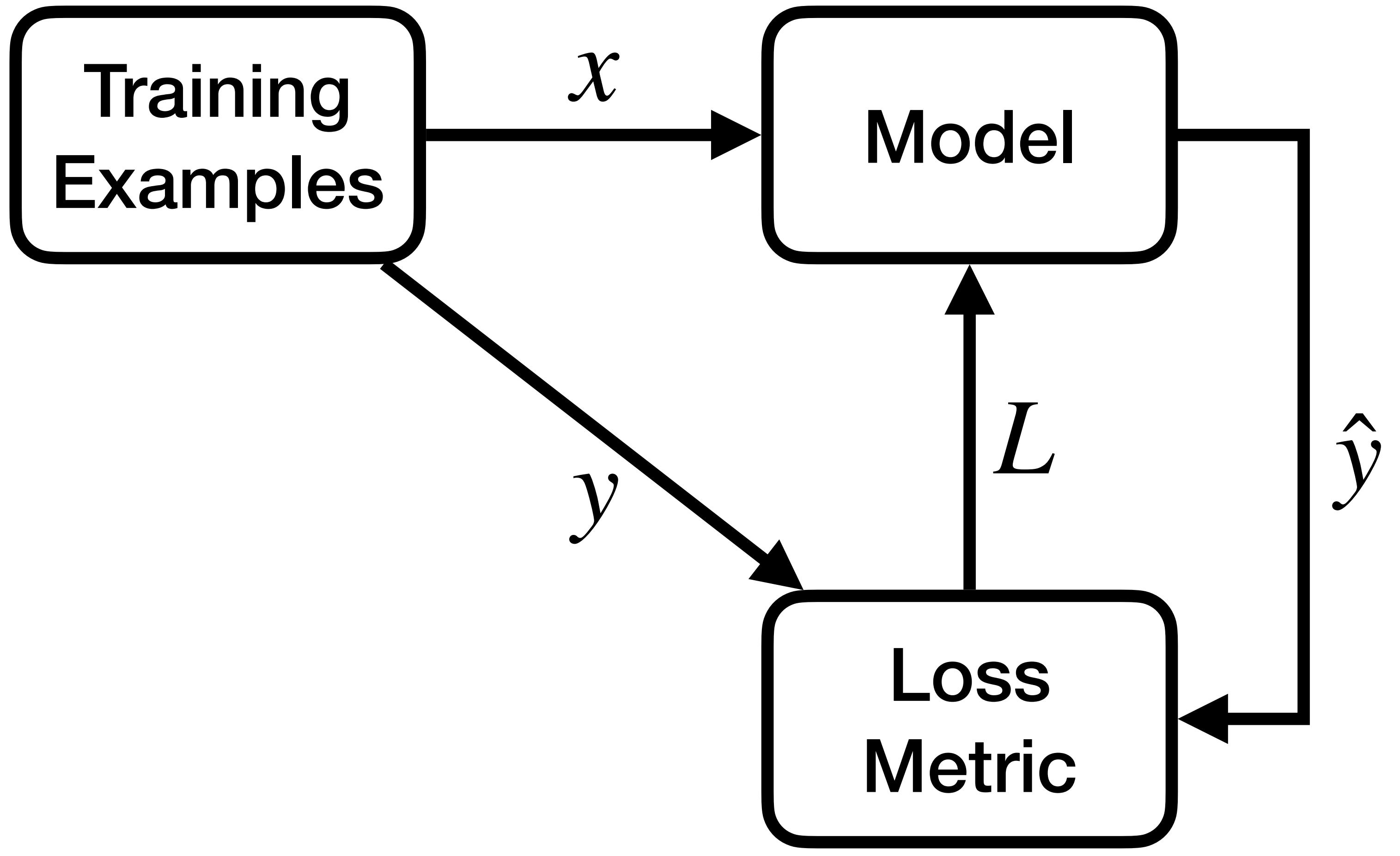
- **Assignment 3** is due **tonight**, 11:59pm
  - Late submissions until **Monday night** (March 27, 11:59pm) with 20% deduction

Recap: Deep Learning

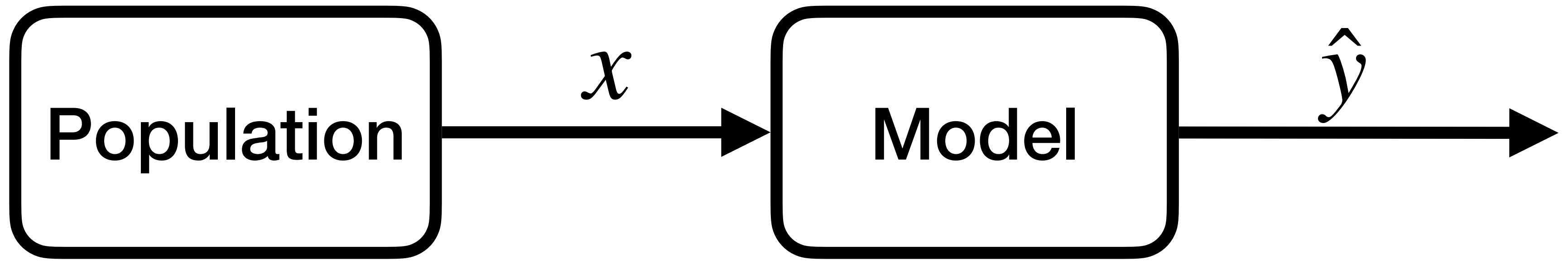
- **Feedforward neural networks** are **extremely flexible** parametric models that can be trained by gradient descent
- **Convolutional neural networks** add **pooling** and **convolution** operations
  - Vastly more efficient to train on **vision** tasks, due to **fewer parameters** and domain-appropriate **invariances**
- **Recurrent neural networks** process elements of a **sequence** **one at a time**, while maintaining **state**
  - Same function with **same parameters** applied to each (element + state)
- **Transformers** process elements of a **sequence** in **parallel**
  - Each output element depends on **weighed sum** of transformed input elements, using same parameters
  - Weights are dot product of input element's **key** and output element's **query**
  - Keys and queries are computed using the **same parameters** for all elements

Recap: Supervised Learning

Neural networks are typically used to solve **supervised learning** tasks: Selecting a **hypothesis** $h : X \to Y$ that maps from **input** features to **target** features.



Training time



Test time

Example: CanBot

- CanBot's job is to find and recycle empty cans
- At any given time, its battery charge is either **high** or **low**
- It can do three actions: **search** for cans, **wait**, or **recharge**
- *Goal:* Find cans efficiently without running out of battery charge

Questions:

1. Is this an instance of a **supervised learning** problem?
2. Is this an instance of a **search** problem?

Reinforcement Learning

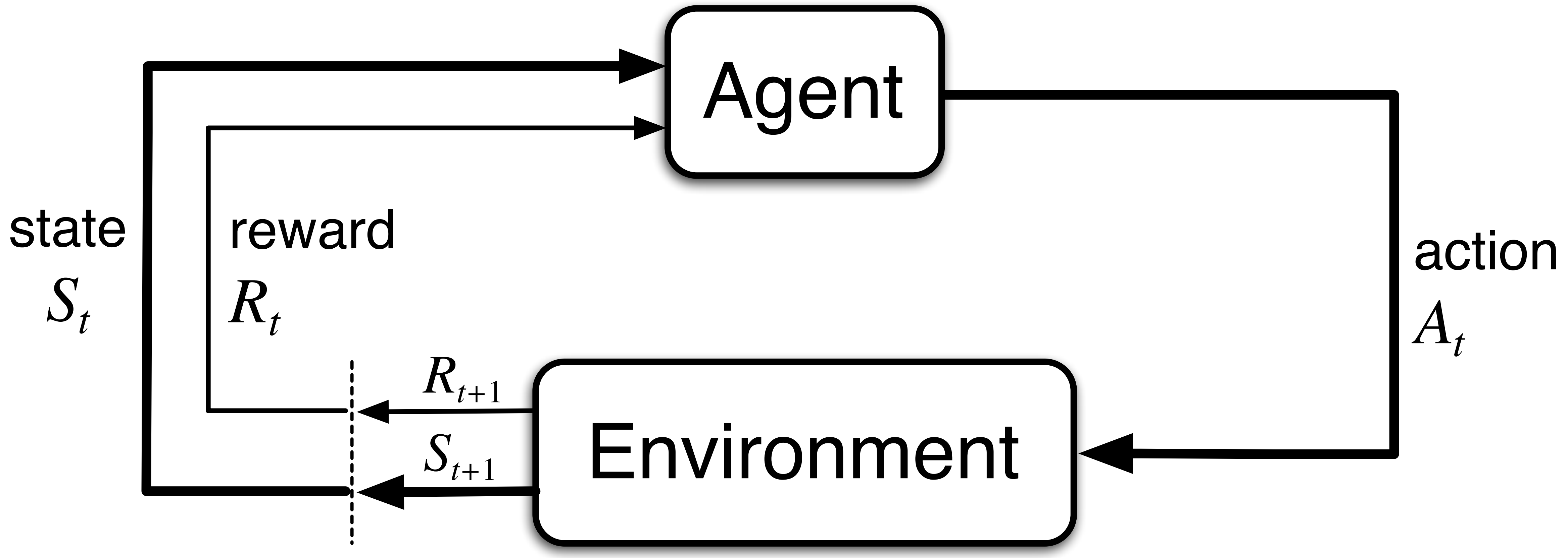
In a **reinforcement learning** task, an agent learns how to **act** based on feedback from the **environment**.

- The agent's actions may change the environment
- The "right answer" is not known
- The task may be either **episodic** or **continuing**
- The agent makes decisions **online**: determines how to act while interacting with the environment

Interacting with the Environment

At each time $t = 1,2,3,\ldots$

1. Agent receives input denoting **current state** S_t
2. Agent chooses **action** A_t
3. Next time step, agent receives **reward** R_{t+1} and **new state** S_{t+1}, chosen according to a distribution $p(s', r \mid s, a)$



This interaction between agent and environment produces a **trajectory**:
$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

Markov Decision Process

Definition:

A **Markov decision process** is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p)$, where

- $\mathcal{S}$ is a set of **states**,
- $\mathcal{A}$ is a set of **actions**,
- $\mathcal{R} \in \mathbb{R}$ is a set of **rewards**,
- $p(s', r \mid s, a) \in [0,1]$ defines the **dynamics** of the process, and
- the probabilities from p **completely** characterize the environment's dynamics

Dynamics

The four-argument dynamics function returns the probability of every **state transition**:

$$p(s', r \mid s, a) \doteq \Pr(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$$

It is often convenient to use **shorthand notation** rather than the full four-argument dynamics function:

$$p(s' \mid s, a) \doteq \Pr(S_t = s' \mid S_{t-1} = s, A_{t-1} = a) = \sum_{r \in \mathcal{R}} p(s', r \mid s, a)$$

$$r(s, a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r \mid s, a)$$

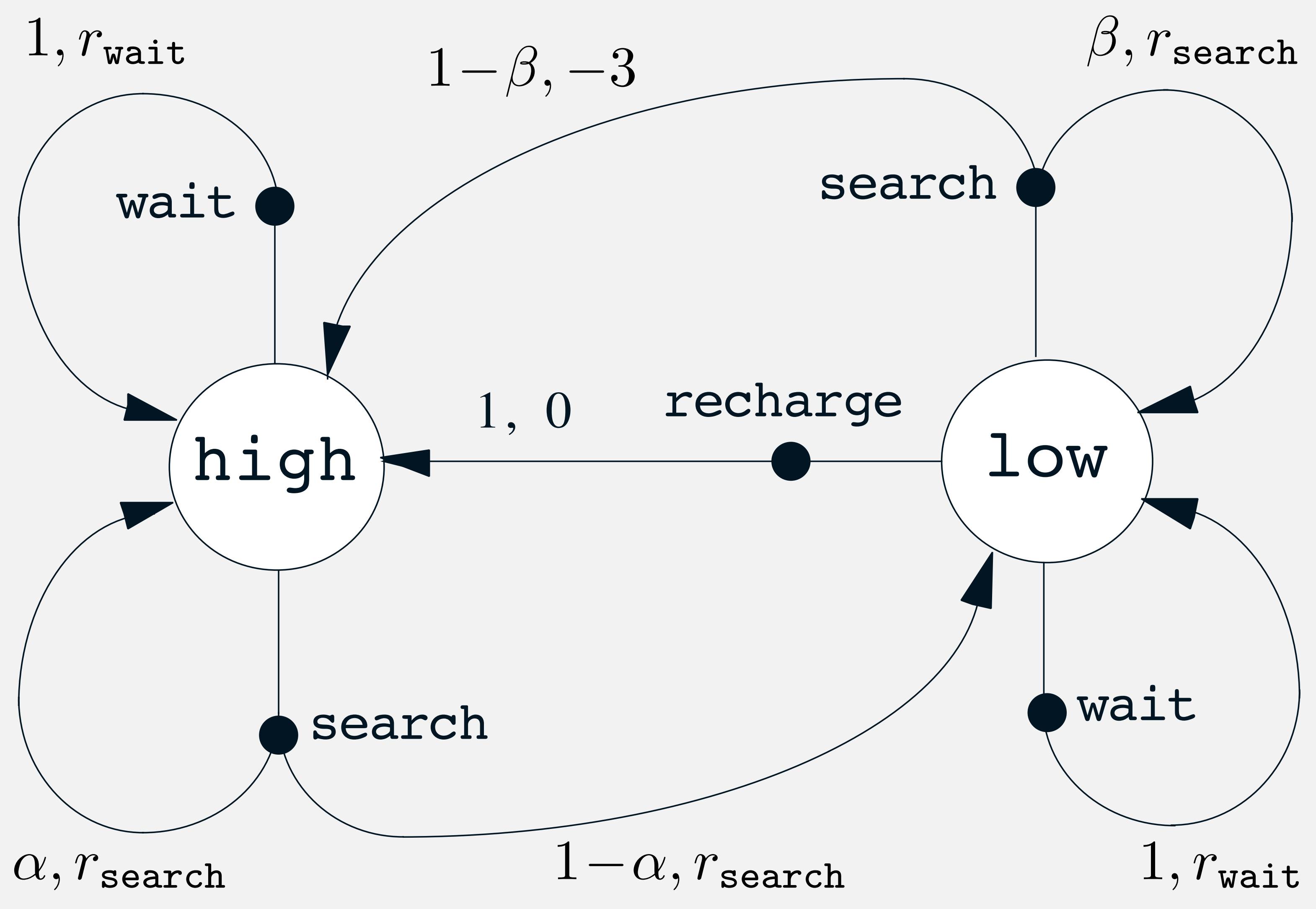
$$r(s, a, s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r \mid s, a)}{p(s' \mid s, a)}$$

CanBot as a Reinforcement Learning Agent

Question: How can we represent CanBot as a **reinforcement learning** agent?

- Need to define **states**, **actions**, **rewards**, and **dynamics**

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	$r_{\texttt{search}}$
high	search	low	$1 - \alpha$	$r_{\texttt{search}}$
low	search	high	$1 - \beta$	-3
low	search	low	β	$r_{\texttt{search}}$
high	wait	high	1	$r_{\texttt{wait}}$
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	$r_{\texttt{wait}}$
low	recharge	high	1	0
low	recharge	low	0	-



(Image: Sutton & Barto, 2018)

Reward Hypothesis

Definition: *Reward hypothesis*

An agent's goals and purposes can be entirely represented as the maximization of the **expected value** of the **cumulative sum** of a **scalar signal**.

Returns for Episodic Tasks

Question:
What does "maximize the expected value of the cumulative sum of rewards" *mean*?

Definition: A task is **episodic** if it ends after some **finite number** T of time steps in a special **terminal state** S_T.

Definition: The **return** G_t after time t is the sum of rewards received after time t: $G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$.

Answer: The return G_t is a **random variable**. In an episodic task, we want to maximize its **expected value** $\mathbb{E}[G_t]$.

Returns for Continuing Tasks

Definition: A task is **continuing** if it does not end (i.e., $T = \infty$).

- In a continuing task, we can't just maximize the sum of rewards (**why?**)
- Instead, we maximize the **discounted return**:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$\gamma \leq 1$ is the **discount factor**

- Returns are **recursively** related to each other:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
$$= R_{t+1} + \gamma G_{t+1}$$

Policies

Question: How should an agent in a Markov decision process choose its **actions**?

- **Markov assumption**: The state incorporates all of the necessary information about the history up until this point
  - i.e., Probabilities of future rewards & transitions are the same from state S_t **regardless of how you got there**
- So the agent can choose its actions based **only** on S_t
- This is called a (memoryless) **policy**: $\pi(a \mid s) \in [0,1]$ is the probability of taking **action** a given that the **current state** is s

State-Value Function

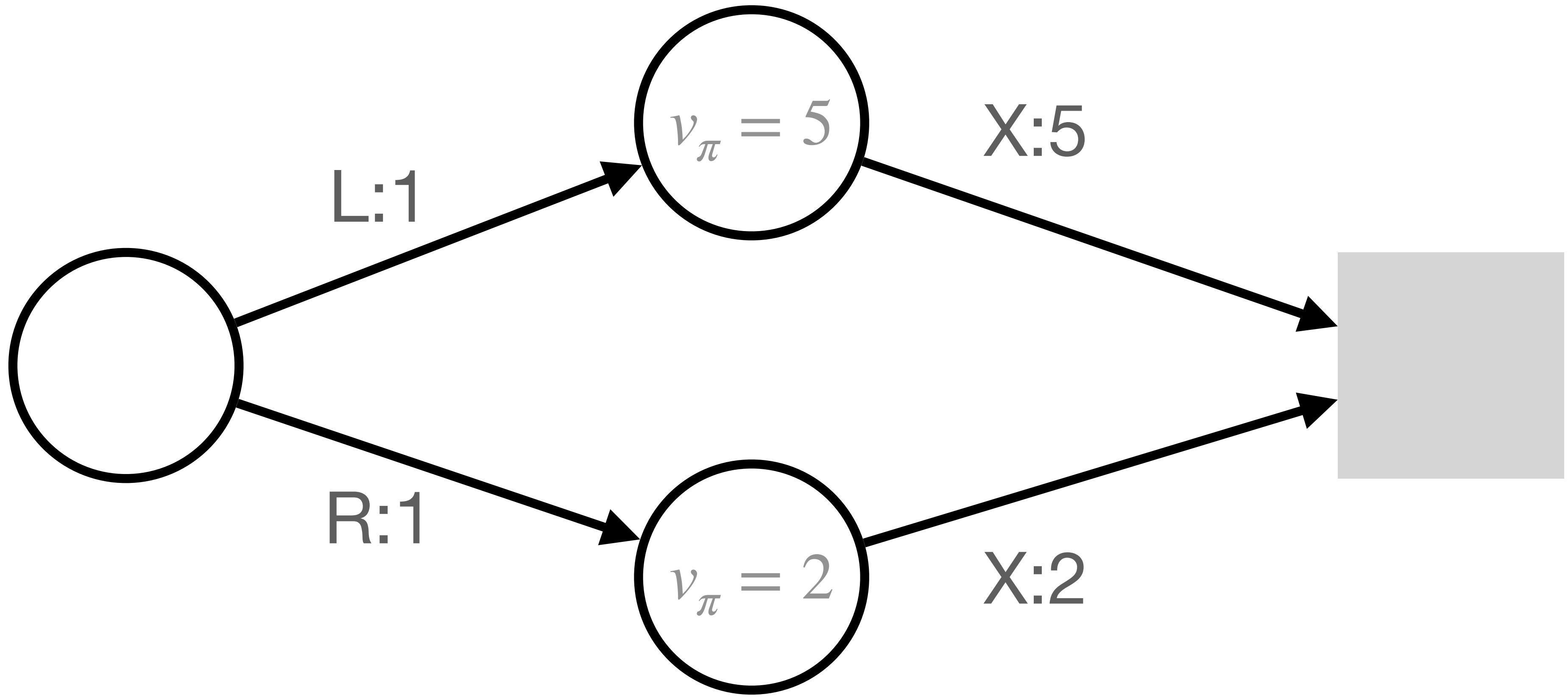
- Once you know the **policy** π and the **dynamics** p, you can compute the probability of every possible state transition starting from any given state
- It is often valuable to know the **expected return** starting from a given state s under a given policy π (**why?**)
- The **state-value function** v_π returns this quantity:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] \quad \forall t$$

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

Using State-Value Function

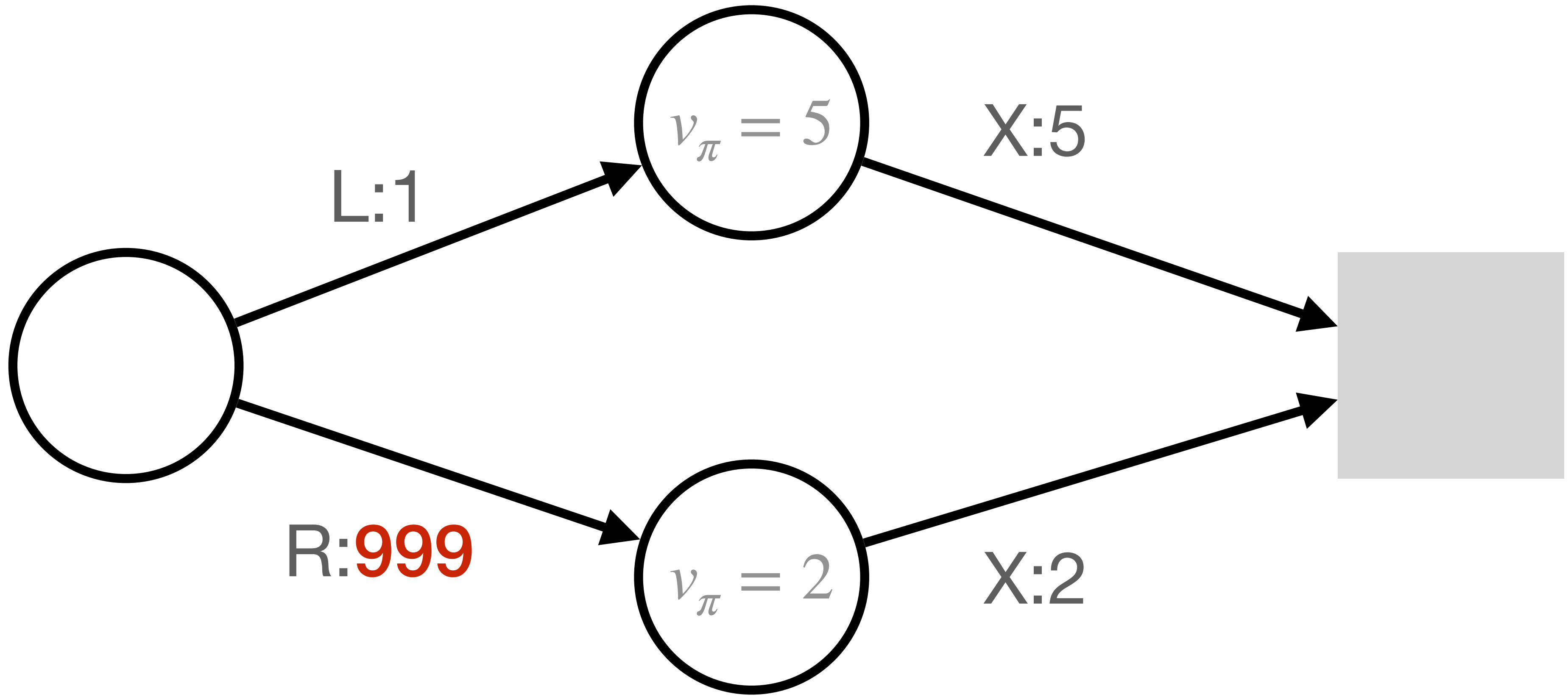
Question: Suppose state transitions are deterministic. Does it make sense to always choose the action that leads to the next state s' with the highest $v_\pi(s)$?



Using State-Value Function

Question: Suppose state transitions are deterministic. Does it make sense to always choose the action that leads to the next state s' with the highest $v_\pi(s)$?

Not always; the reward for the **transition itself** is also important!



Action-Value Function

The **action-value function** $q_\pi(s, a)$ estimates the expected return G_t starting from state s if we

1. Take action a in state $S_t = s$, *and then*
2. Follow policy π for every state S_{t+1} afterward

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

$$= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Question:
How is this any different from the **state-value** function $v_\pi(s)$?

Bellman Equations

Value functions satisfy a **recursive consistency condition** called the **Bellman equation:**

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$
$$= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \dots\right)$$
$$= R_{t+1} + \gamma G_{t+1}$$

$$\mathbb{E}[A + cB] = \mathbb{E}[A] + c\mathbb{E}[B]$$

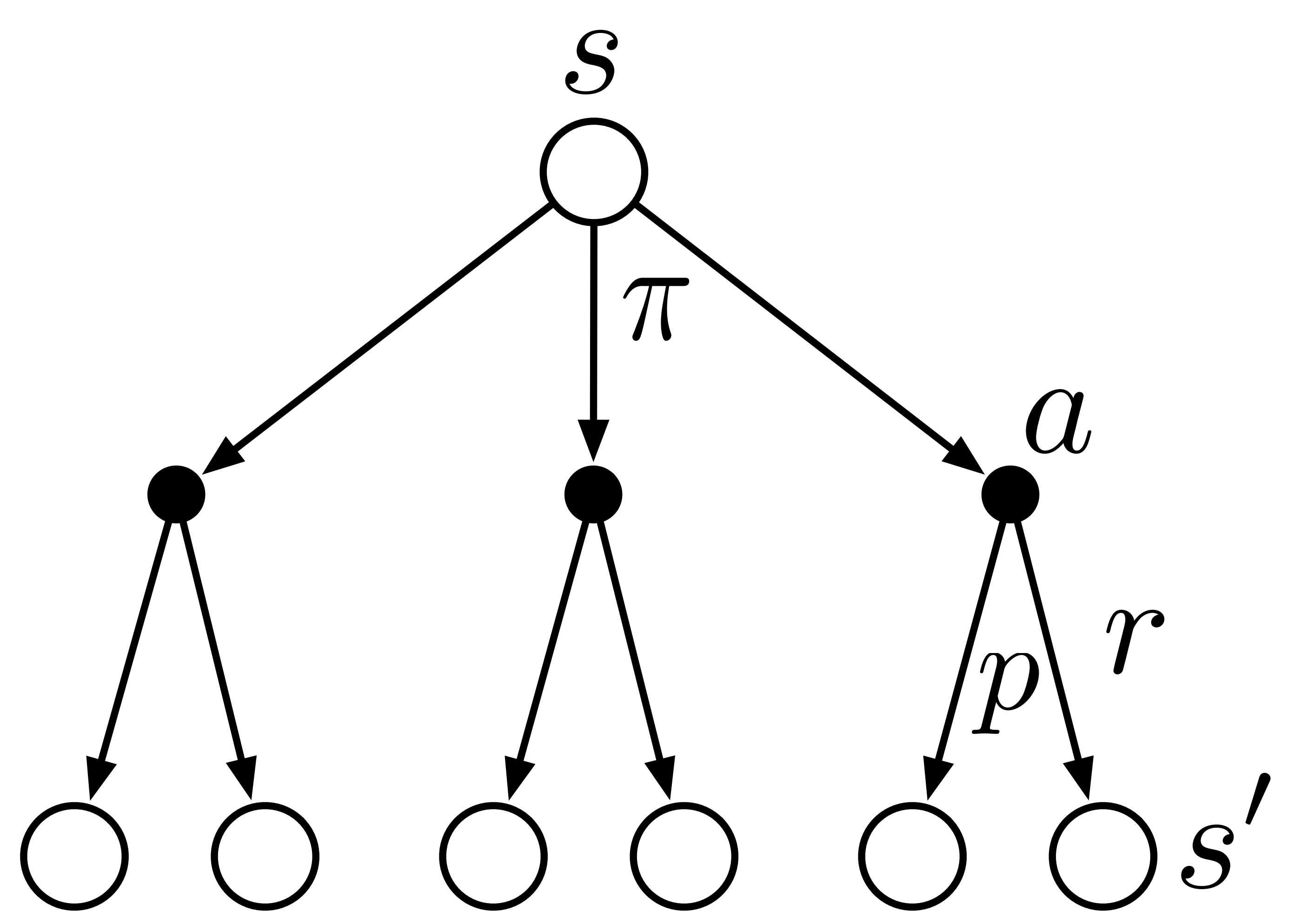
$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \,|\, S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$
$$= \mathbb{E}_\pi[R_{t+1} \mid S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_t = s]$$
$$= \sum_a \sum_{s'} \sum_r \Pr[S_{t+1} = s', R_{t+1} = r, A_t = a \mid S_t = s]\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \,|\, S_{t+1} = s']\right]$$
$$= \sum_a \sum_{s'} \sum_r \Pr[S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a] \Pr[A_t = a \mid S_t = s]\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \,|\, S_{t+1} = s']\right]$$
$$= \sum_a \pi(a \,|\, s) \sum_{s'} \sum_r p(s', r \,|\, s, a)\left[r + \gamma \mathbb{E}_\pi[G_{t+1} \,|\, S_{t+1} = s']\right]$$
$$= \sum_a \pi(a \,|\, s) \sum_{s',r} p(s', r \,|\, s, a)\left[r + \gamma v_\pi(s')\right] \quad \blacksquare$$

- v_π is the **unique solution** to π's Bellman equation
- There is also a Bellman equation for π's **action-value function**

Backup Diagrams

Backup diagrams help to visualize the flow of **information back to a state** from its successor states or action-state pairs:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$
$$= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

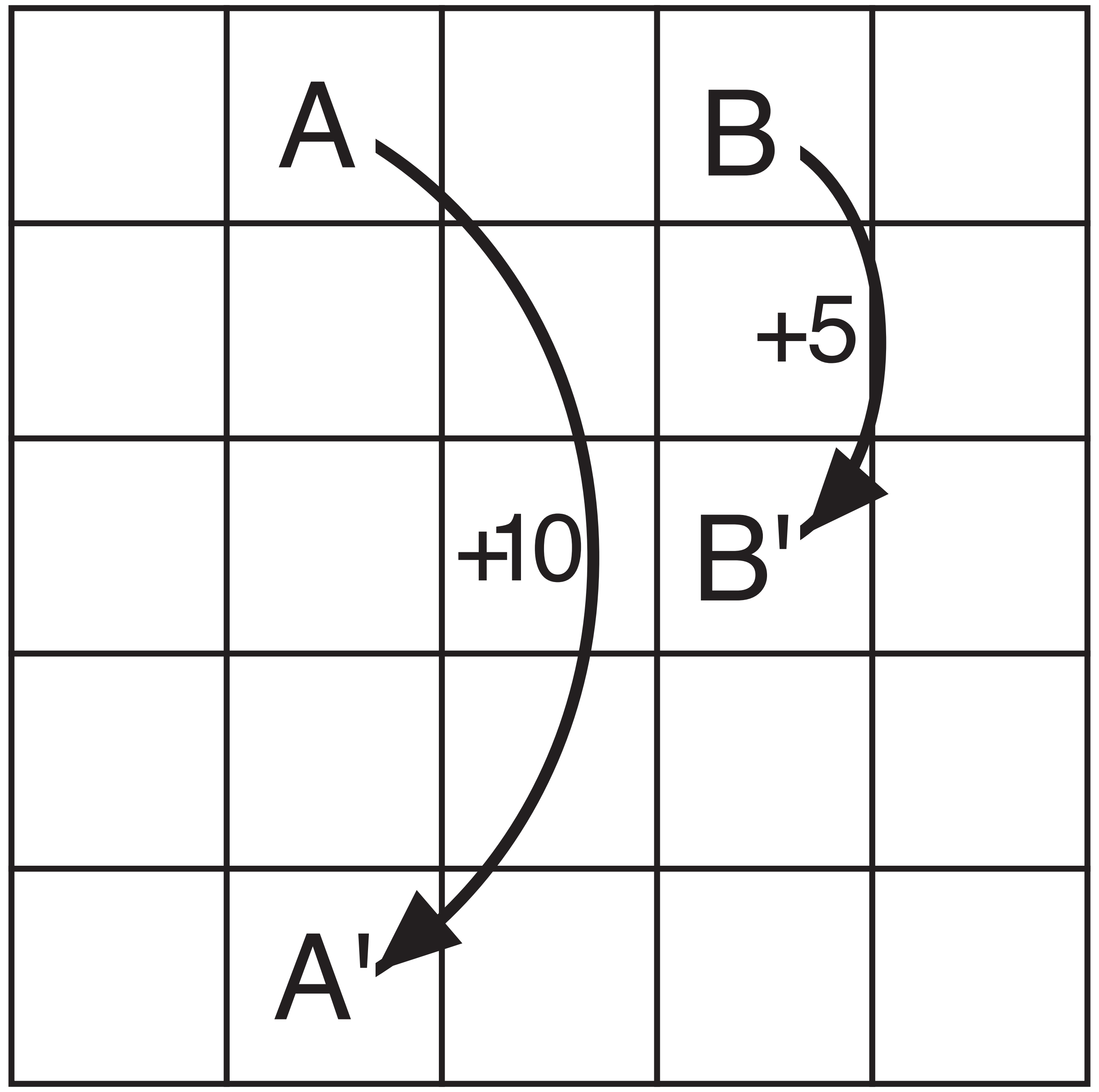


Backup diagram for v_π

(Image: Sutton & Barto, 2018)

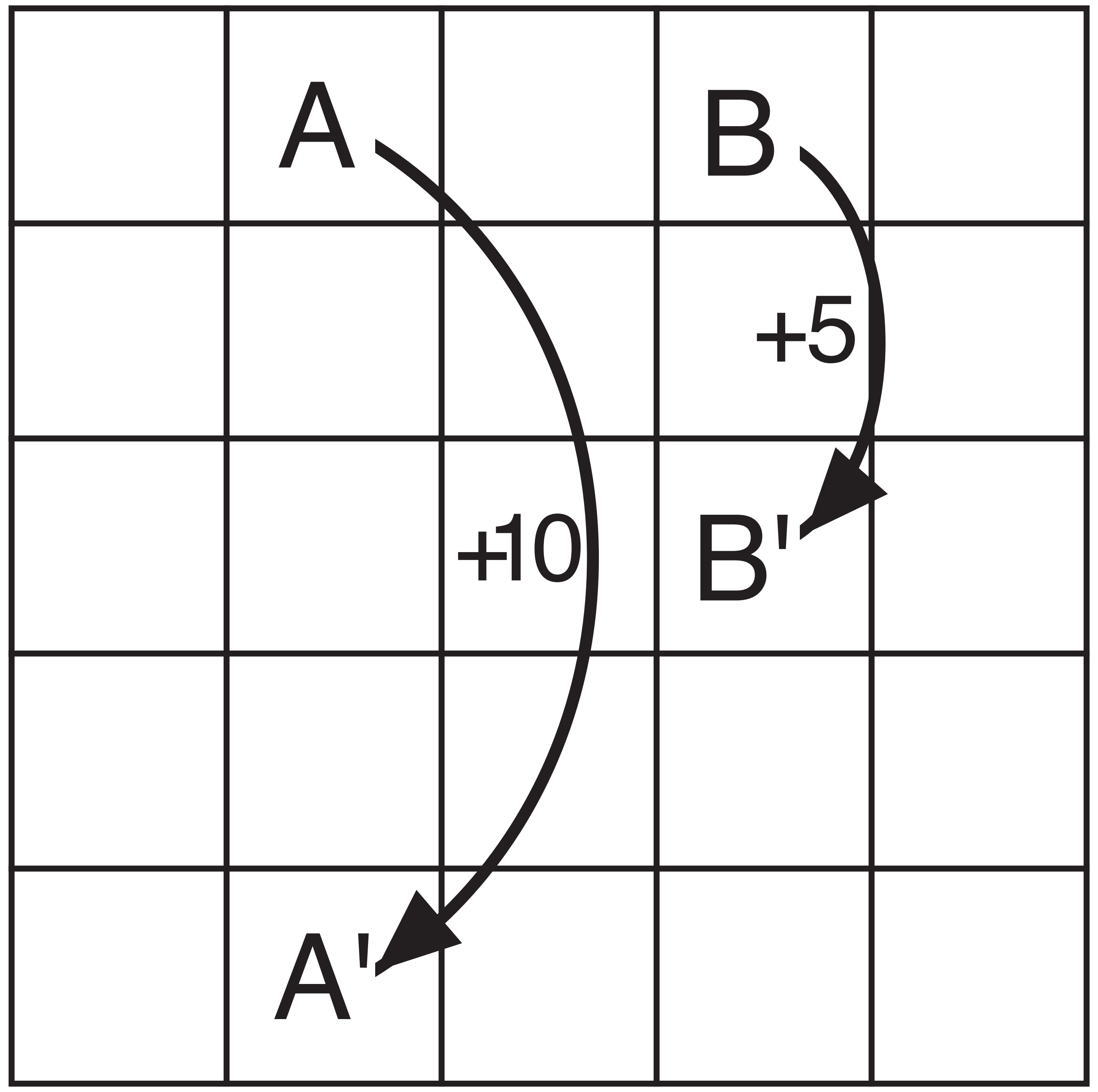
GridWorld

- At each cell, can go `north`, `south`, `east`, `west`
- Try to go off the **edge**: reward of **-1**
- Leaving state **A**: takes you to state **A′**, reward of **+10**
- Leaving state **B**: takes you to state **B′**, reward of **+5**



(Image: Sutton & Barto, 2018)

GridWorld



Reward dynamics

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

State-value function v_π for random policy

$$\pi(a \mid s) = 0.25$$

(Image: Sutton & Barto, 2018)

Summary

- **Supervised learning models** are trained offline using labelled training examples, and then make predictions
- **Reinforcement learning agents** choose their actions online, and update their behaviour based on rewards from the environment
- We can formally represent reinforcement learning environments using **Markov decision processes**, for both episodic and continuing tasks
- Reinforcement learning agents maximize expected returns
- **Policies** map states to (distribution over) actions
- Given a policy π, every state s has an expected value $v_\pi(s)$
- State-value and action-value functions satisfy the Bellman equations