香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

COMP 5212
Machine Learning
Lecture 23

Reinforcement Learning

Junxian He

May 3, 2024

Reminder

1. Programming HW is due today
2. We will have HW4 released this week, all multi-choice questions, helping you review the contents in this semester

Markov Decision Process

1. Start in some initial state s_0

2. For time step t:
   a. Agent observes state s_t
   b. Agent takes action $a_t = \pi(s_t)$ Deterministic policy
   c. Agent receives reward $r_t \sim p(r \mid s_t, a_t)$
   d. Agent transitions to state $s_{t+1} \sim p(s' \mid s_t, a_t)$

- MDPs make the ***Markov assumption***: the reward and next state only depend on the current state and action.

RL Example - gridworld

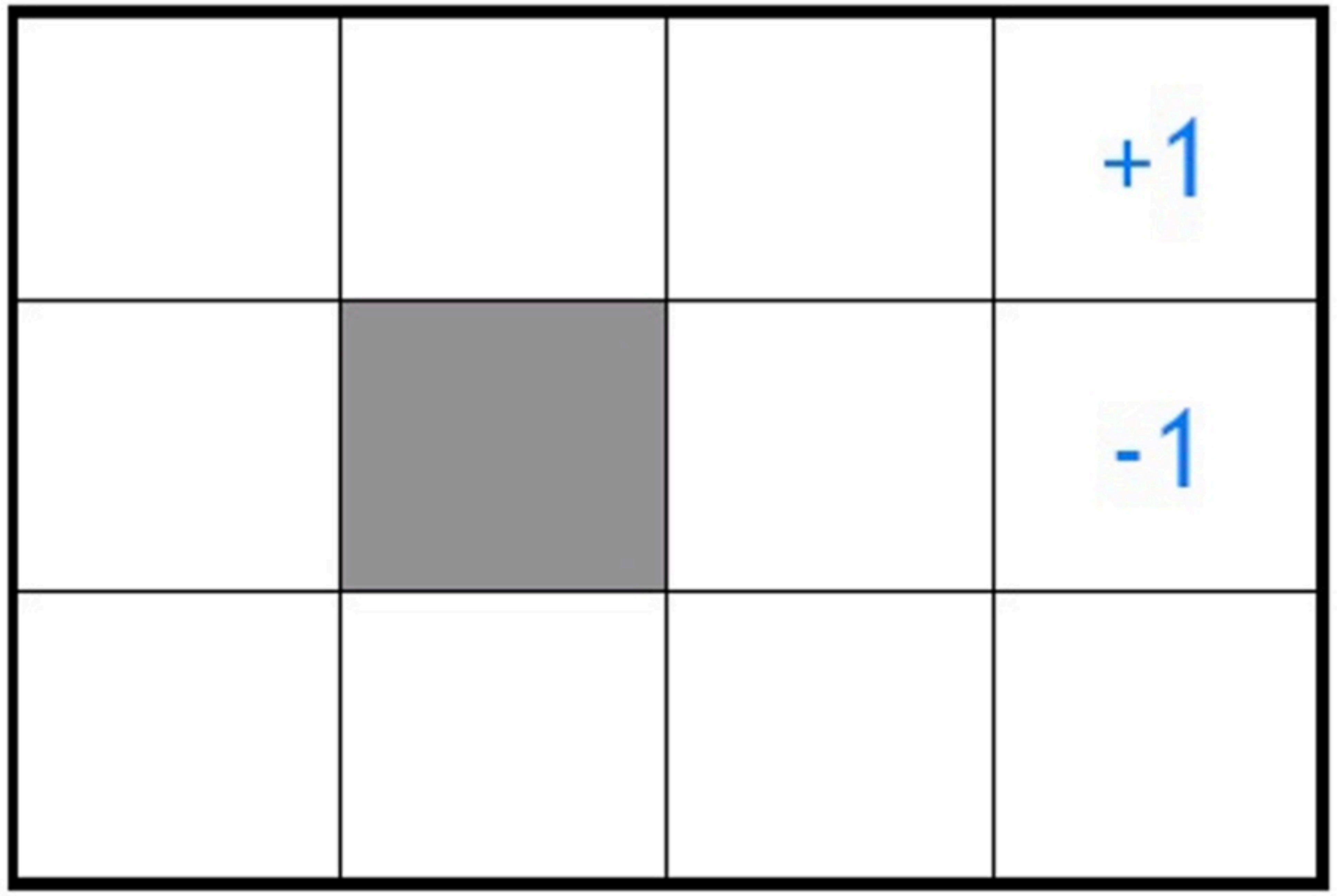
$\mathcal{S}$ = all empty squares in the grid

$\mathcal{A}$ = {up, down, left, right}

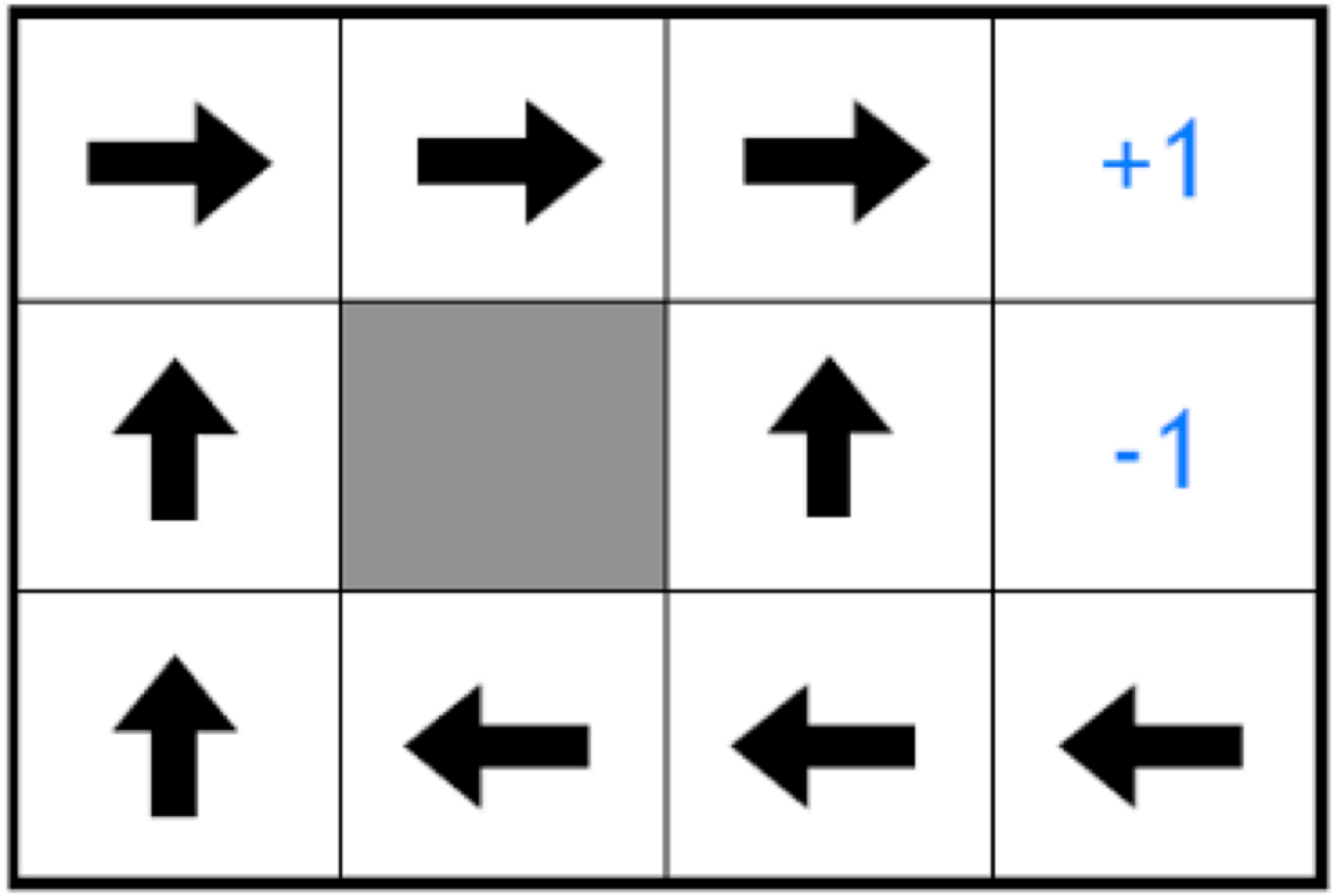
Deterministic transitions

Rewards of +1 and -1 for entering the labelled squares

Terminate after receiving either reward



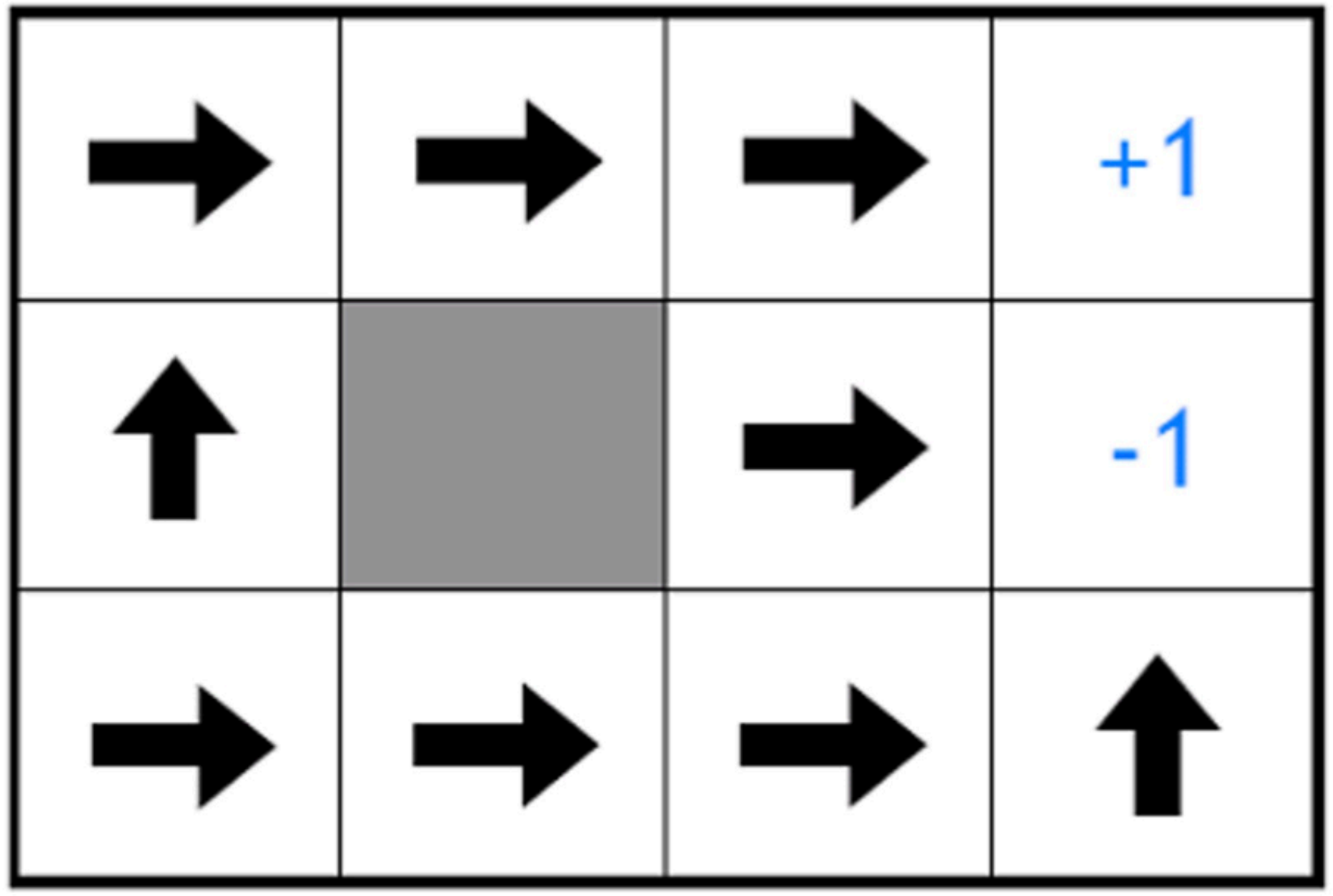
RL Example - gridworld



Is this policy optimal?

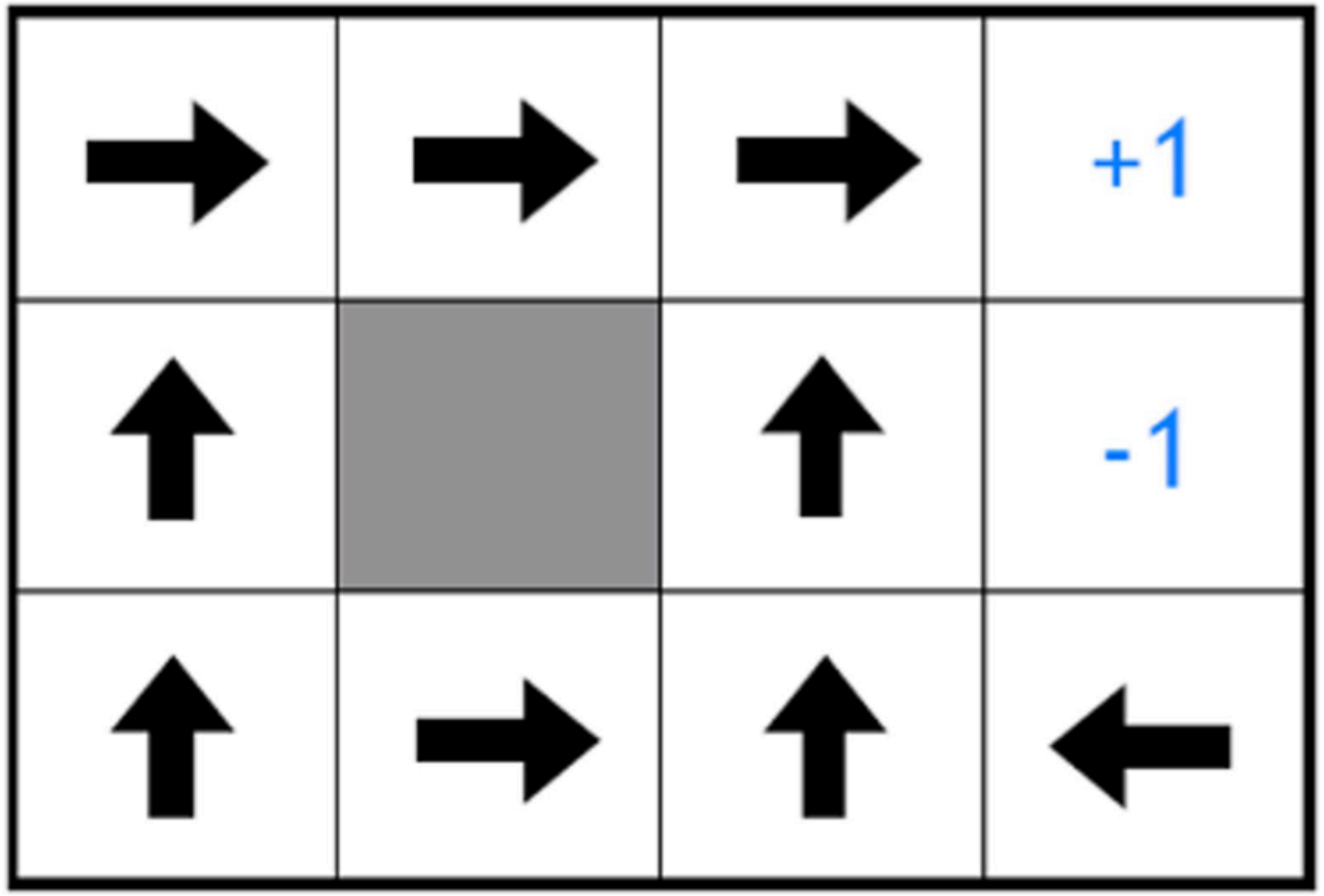
RL Example - gridworld

Optimal policy given a reward of -2 per step



RL Example - gridworld

Optimal policy given a reward of -0.5 per step



What would be the algorithm to find the optimal policy automatically?

Discounted Reward

Total reward is $\sum_{t=0}^{\infty} \gamma^t r_t = r_0 + \gamma\, r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ \ldots$

where $0 < \gamma < 1$ is some discount factor for future rewards

Why discount?

- Mathematically tractable – total reward doesn't explode

  $1 + 1 + 1 + \ldots = \infty$ but $1 + 0.8*1 + (0.8)^2*1 + \ldots = 5$

- Actions don't have lasting impact

Key Challenges

- The algorithm has to gather its own training data

- The outcome of taking some action is often stochastic or unknown until after the fact

- Decisions can have a delayed effect on future outcomes (exploration-exploitation tradeoff)

explore decisions whose reward is uncertain

exploit decisions which give high reward

RL: Objective function

- Find a policy $\pi^* = \underset{\pi}{\operatorname{argmax}}\ V^\pi(s)\ \forall\, s \in \mathcal{S}$
- $V^\pi(s) = \mathbb{E}[$*discounted* total reward of starting in state s and executing policy π forever$]$

$$= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[R\left(s_t, \pi(s_t)\right)\right]$$

where $0 < \gamma < 1$ is some discount factor for future rewards

Value Function

Bellman equations

$$V^{\pi}(s) = \underbrace{R\big(s, \pi(s)\big)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s_1 \in \mathcal{S}} p\big(s_1 \mid s, \pi(s)\big) V^{\pi}(s_1)}_{\text{Expected (Discounted) Future reward}}$$

Recursive form

Solve Value Function

$$V^{\pi}(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s_1 \in \mathcal{S}} p\big(s_1 \mid s, \pi(s)\big) V^{\pi}(s_1)$$

Given R, transition function p, and policy $\pi(s)$, we can utilize this equation to solve V(s) for any s

How?

Suppose the state size is finite |S|, you have |S| *linear* equations with |S| variables

Optimal value function and policy

- Optimal value function:

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')\right]$$

  - System of $|\mathcal{S}|$ equations and $|\mathcal{S}|$ variables – nonlinear!

- Optimal policy:

$$\pi^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\ \underbrace{R(s,a)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^*(s')}_{\text{Expected (Discounted) Future reward}}$$

- Insight: if you know the optimal value function, you can solve for the optimal policy!

Value Iteration

- Inputs: $R(s,a)$, $p(s' \mid s,a)$, $0 < \gamma < 1$
- Initialize $V^{(0)}(s) = 0 \;\forall\, s \in \mathcal{S}$ (or randomly) and set $t = 0$
- While not converged, do:
  - For $s \in \mathcal{S}$

$$V^{(t+1)}(s) \leftarrow \max_{a \in \mathcal{A}} \underbrace{\left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')\right]}_{Q(s,a)}$$

  - $t = t + 1$

$$\pi^*(s) \leftarrow \operatorname*{argmax}_{a \in \mathcal{A}} \left[R(s,a) + \gamma \textstyle\sum_{s' \in \mathcal{S}} p(s' \mid s,a) V^{(t)}(s')\right]$$

After finding the optimal value function, we can find the optimal policy

Value Iteration: Convergence

Theorem 1: Value function convergence

V will converge to V^* if each state is "visited" infinitely often (Bertsekas, 1989)

Policy Iteration

- Inputs: $R(s, a), p(s' \mid s, a), 0 < \gamma < 1$
- Initialize π randomly
- While not converged, do:
  - Solve the Bellman equations defined by policy π Linear equation system

$$V^\pi(s) = R\big(s, \pi(s)\big) + \gamma \sum_{s' \in \mathcal{S}} p\big(s' \mid s, \pi(s)\big) V^\pi(s')$$

  - Update π

$$\pi(s) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}} \; R(s, a) + \gamma \sum_{s' \in \mathcal{S}} p(s' \mid s, a) V^\pi(s')$$

Both value iteration and policy iteration are standard algorithms for solving MDPs, there isn’t universal agreement over which is better

Learning a Model for an MDP

State transition $p(s' \mid s, a)$ and reward function R(s, a) are unknown in practice

Suppose we have a number of trials:

$$
\begin{aligned}
& s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \xrightarrow{a_2^{(1)}} s_3^{(1)} \xrightarrow{a_3^{(1)}} \dots \\
& s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} s_2^{(2)} \xrightarrow{a_2^{(2)}} s_3^{(2)} \xrightarrow{a_3^{(2)}} \dots \\
& \dots
\end{aligned}
$$

$$P_{sa}(s') = \frac{\#\text{times took we action } a \text{ in state } s \text{ and got to } s'}{\#\text{times we took action a in state } s}$$

Similarly we can estimate R(s, a) to be the the average reward observed at state s with action a

Finding the Optimal Policy

1. Initialize π randomly.

2. Repeat {

   (a) Execute π in the MDP for some number of trials.

   (b) Using the accumulated experience in the MDP, update our estimates for P_{sa} (and R, if applicable).

   (c) Apply value iteration with the estimated state transition probabilities and rewards to get a new estimated value function V.

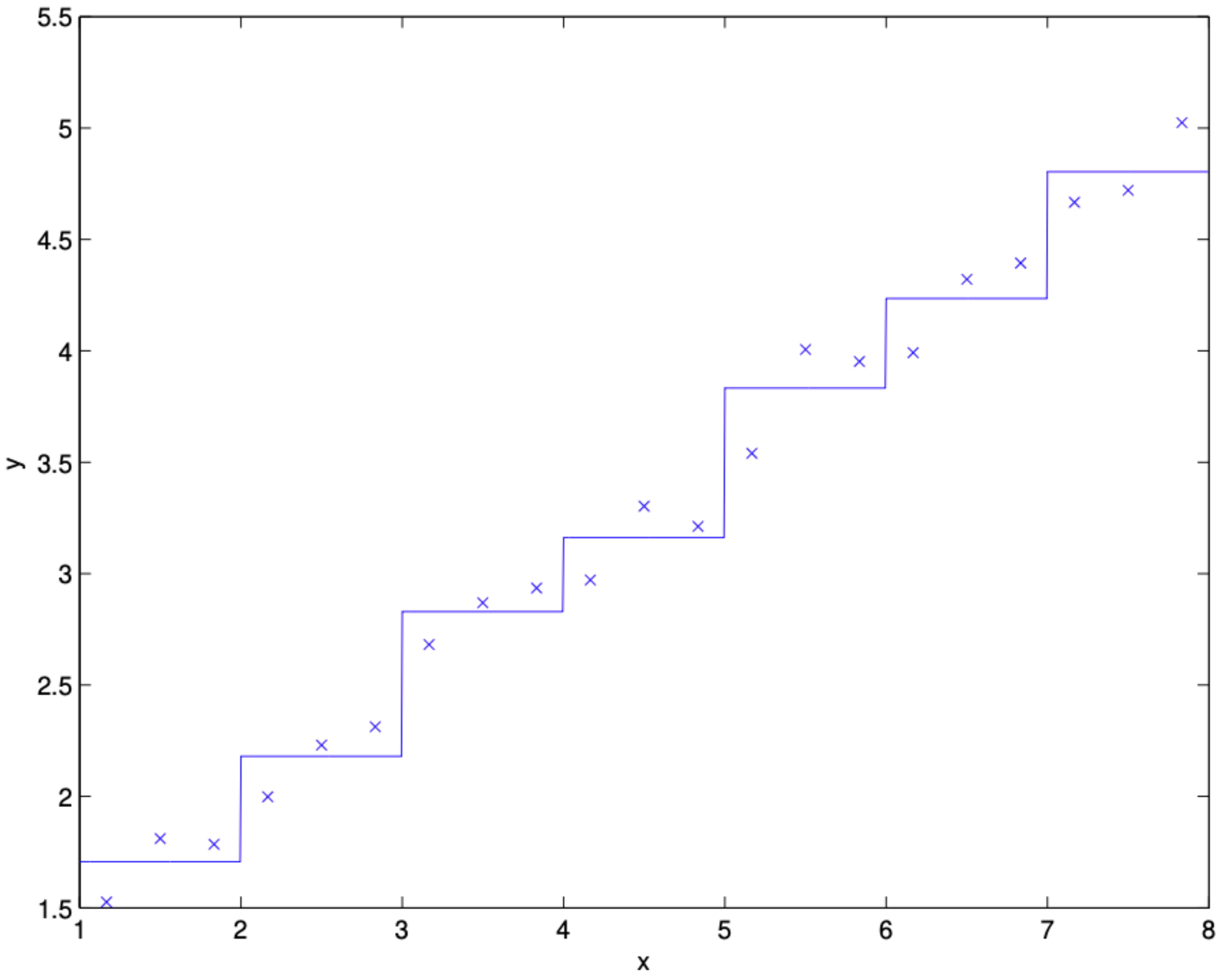
   (d) Update π to be the greedy policy with respect to V.

   }

Continuous State

Continuous states are common, e.g., using (x, y) coordinates and velocity to express the state of a car

Discretization



Dimensions are high when we have several states

Value Function Approximation

Learn a function $f_\theta(s) : s \rightarrow V(s)$

Similar to supervised learning, $f_\theta(s)$ could be a neural network

1. Randomly sample n states $s^{(1)}, s^{(2)}, s^{(3)} \ldots$

Want to compose a training dataset, next is to estimate V(s) for these n points

2. For every state, repeat value iteration to approximate $V(s^{(i)})$

3. Now we have a supervised dataset to train $f_\theta(s)$

Learning a Proxy Model

Policy is a function parameterized by θ: π_θ

$\tau = (s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T)$ is the trajectory from π_θ

τ is a random variable

Total payoff for the policy is:

$$\eta_\theta = \mathbb{E}_{\tau \sim p_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)]$$

p_θ contains the policy and the transition p(s' | s, a)

We want to optimize θ to maximize η

Policy Gradient

$$\eta_\theta = \mathbb{E}_{\tau \sim p_\theta}[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)]$$

We define $f(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$

$$\eta(\theta) = \mathrm{E}_{\tau \sim P_\theta}\left[f(\tau)\right]$$

Connection to VAE? Reparameterization trick?

Policy Gradient

$$
\begin{aligned}
\nabla_\theta \mathrm{E}_{\tau \sim P_\theta}[f(\tau)] &= \nabla_\theta \int P_\theta(\tau) f(\tau) d\tau \\
&= \int \nabla_\theta (P_\theta(\tau) f(\tau)) d\tau \quad \text{(swap integration with gradient)} \\
&= \int (\nabla_\theta P_\theta(\tau)) f(\tau) d\tau \quad \text{(becaue } f \text{ does not depend on } \theta) \\
&= \int P_\theta(\tau) (\nabla_\theta \log P_\theta(\tau)) f(\tau) d\tau
\end{aligned}
$$

$$
= \mathrm{E}_{\tau \sim P_\theta}[(\nabla_\theta \log P_\theta(\tau)) f(\tau)]
$$

Can be approximated using MC sampling

Policy gradient is a commonly used method to propagate gradients through discrete variables

Policy Gradient

$$\mathrm{E}_{\tau\sim P_\theta}\left[(\nabla_\theta \log P_\theta(\tau))f(\tau)\right]$$

What is $\nabla_\theta \log P_\theta(\tau)$

$$P_\theta(\tau) = \mu(s_0)\pi_\theta(a_0|s_0)P_{s_0a_0}(s_1)\pi_\theta(a_1|s_1)P_{s_1a_1}(s_2)\cdots P_{s_{T-1}a_{T-1}}(s_T)$$

$$\nabla_\theta \log P_\theta(\tau) = \nabla_\theta \log \pi_\theta(a_0|s_0) + \nabla_\theta \log \pi_\theta(a_1|s_1) + \cdots + \nabla_\theta \log \pi_\theta(a_{T-1}|s_{T-1})$$

$$\begin{aligned}\nabla_\theta\eta(\theta) = \nabla_\theta \mathrm{E}_{\tau\sim P_\theta}\left[f(\tau)\right] &= \mathrm{E}_{\tau\sim P_\theta}\left[\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\cdot f(\tau)\right] \\ &= \mathrm{E}_{\tau\sim P_\theta}\left[\left(\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\cdot\left(\sum_{t=0}^{T-1}\gamma^t R(s_t,a_t)\right)\right]\end{aligned}$$

A Lemma

$$\mathop{\mathrm{E}}_{x \sim P_\theta} [\nabla_\theta \log P_\theta(x)] = 0.$$

$$\mathrm{E}_{a_t \sim \pi_\theta(\cdot|s_t)} \nabla_\theta \log \pi_\theta(a_t|s_t) = 0$$

$$\mathrm{E}_{\tau \sim P_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right] = 0$$

Policy Gradient

$$\begin{aligned}\nabla_\theta \eta(\theta) &= \sum_{t=0}^{T-1} \mathrm{E}_{\tau\sim P_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\cdot\left(\sum_{j=0}^{T-1}\gamma^j R(s_j,a_j)\right)\right]\\ &= \sum_{t=0}^{T-1} \mathrm{E}_{\tau\sim P_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\cdot\left(\sum_{j\geq t}^{T-1}\gamma^j R(s_j,a_j)\right)\right]\end{aligned}$$

Loss does not mean much, and you should only care about the return

Learning Policy and Value Function Together

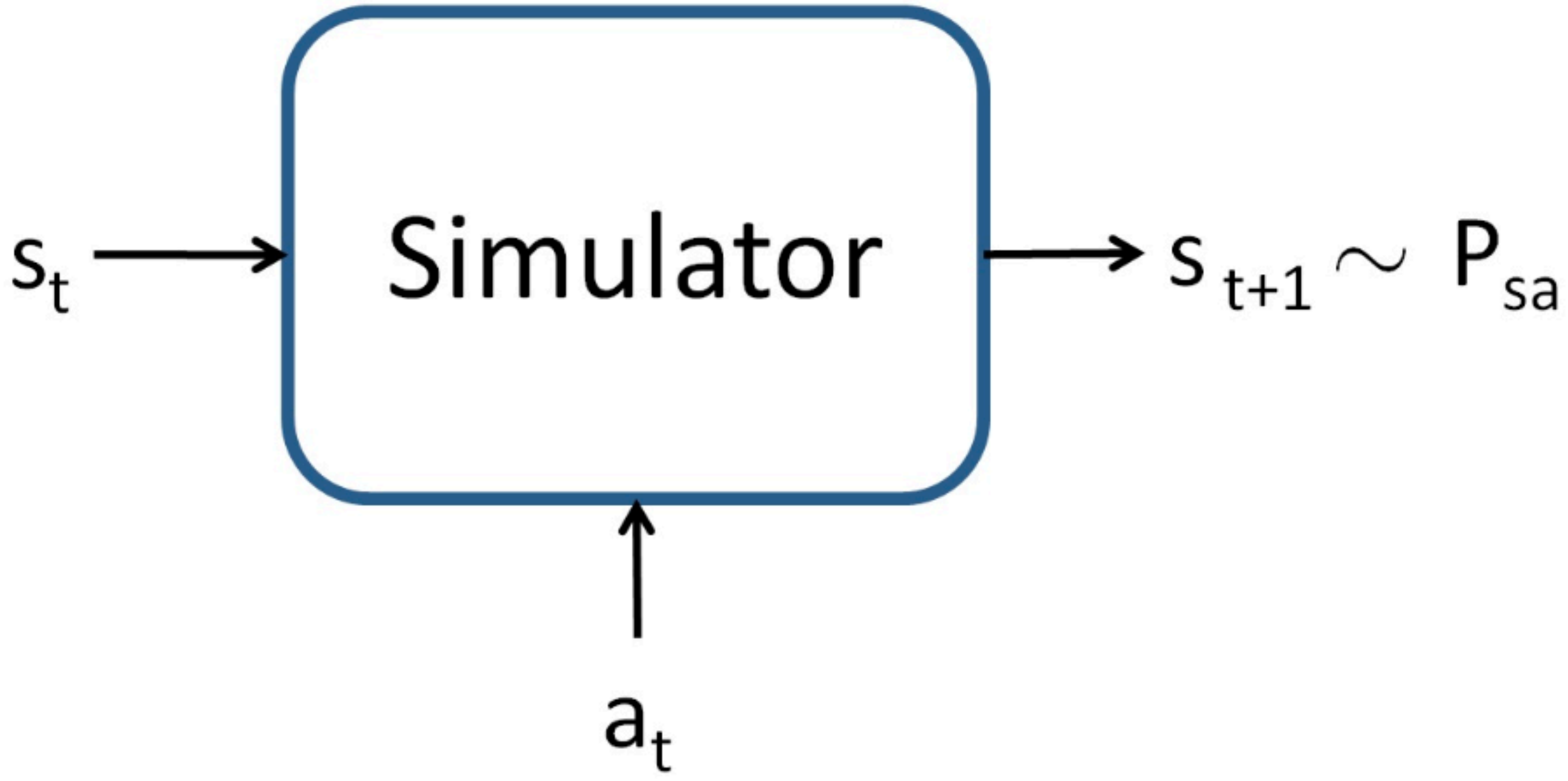
Repeat{

1. Perform a number of trials from policy π_θ to get all the trajectory
2. Update the policy with the current value function
3. Compute the expected reward for each state in the trajectories
4. Supervised training to train the value function

}

Reward models are often trained in advance

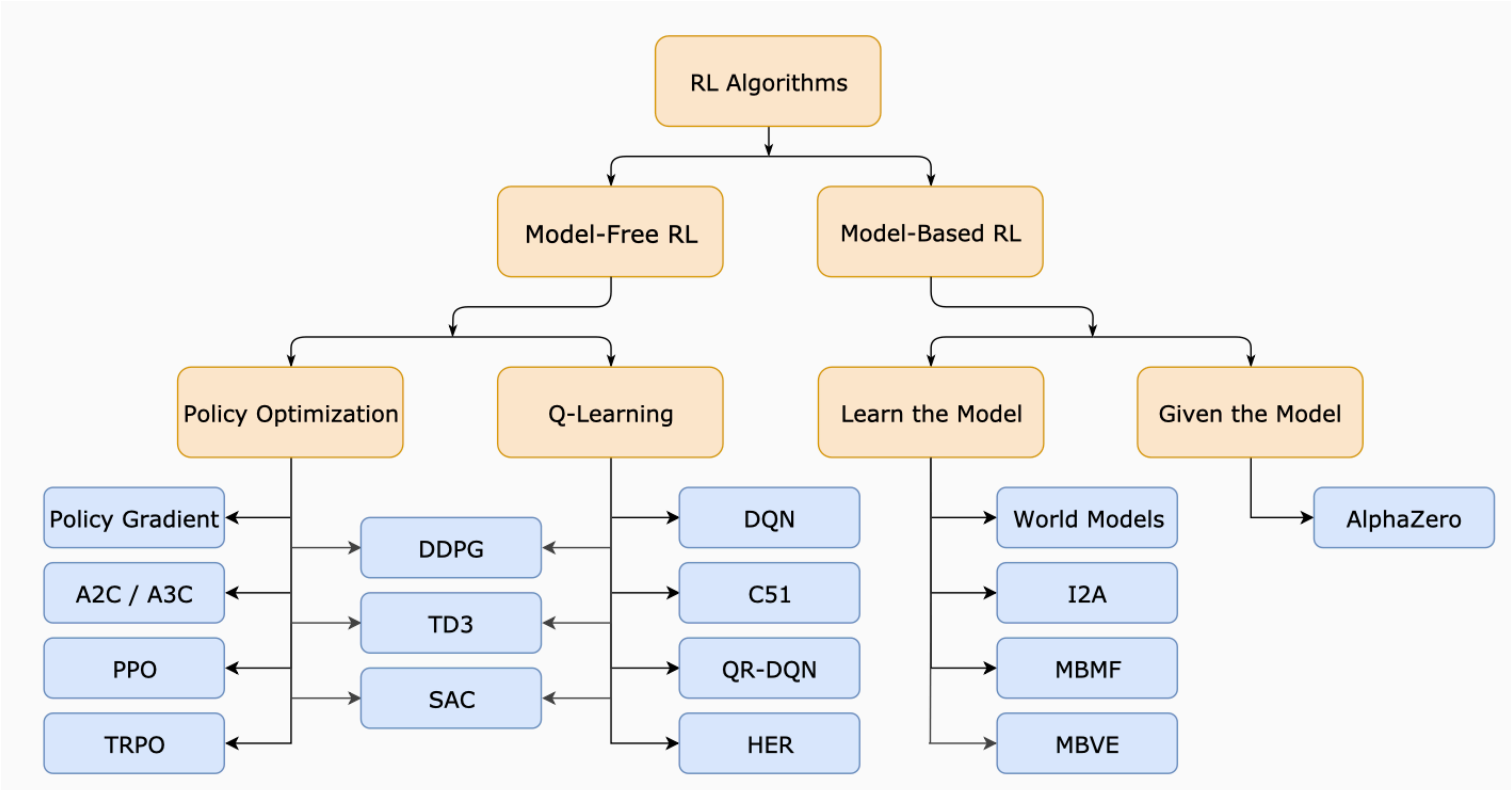
Model for the Environment



Model-based Reinforcement Learning

1. Interaction with real environment can be slow
2. Interaction with real environment can be risky

Taxonomy of RL



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#part-2-kinds-of-rl-algorithms