CSE 478 Robot Autonomy
Reinforcement Learning

Siddhartha Srinivasa (siddh@)
Abhishek Gupta (abhgupta@)

TAs:
Rohan Baijal (rbaijal@)
Sidhartha Talia (sidtalia@)
Christopher Tan (tan7271@)
Helen Wang (yiruwang@)

Lecture Outline

Reinforcement Learning: What it is, where it's used

↓

Recap: A Formalism for Sequential Decision Making

↓

Value Functions and the Bellman Equation

↓

Planning with a known model: value iteration

↓

Q-learning, exploration, and modern RL

Lecture Outline

Reinforcement Learning: What it is, where it's used

↓

Recap: A Formalism for Sequential Decision Making

↓

Value Functions and the Bellman Equation

↓

Planning with a known model: value iteration

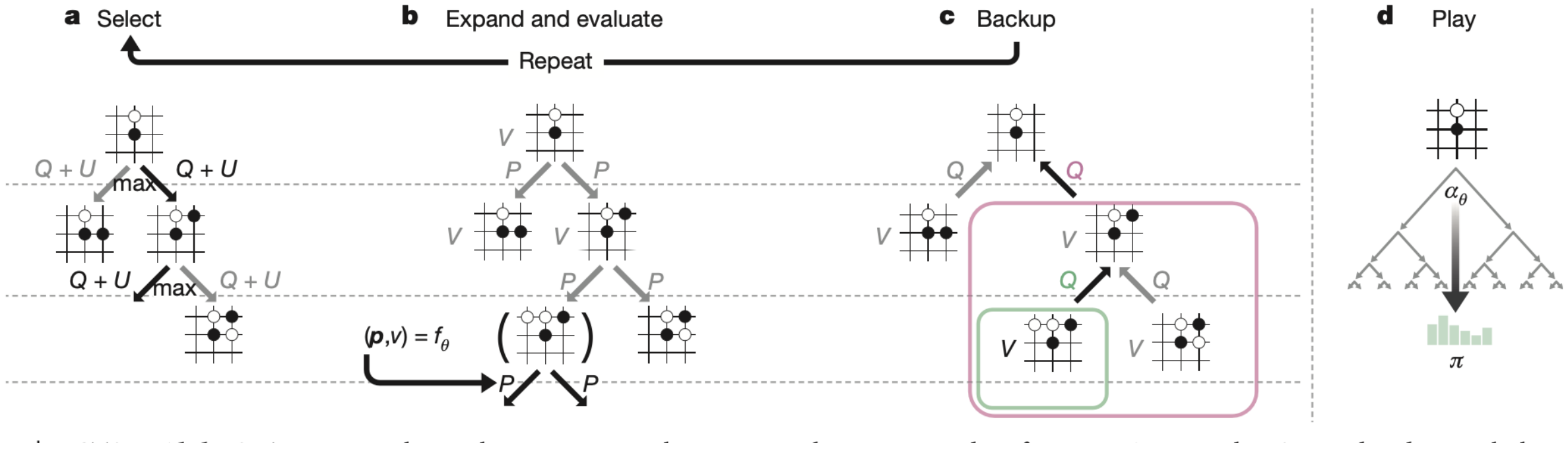
↓

Q-learning, exploration, and modern RL

What is Reinforcement Learning?

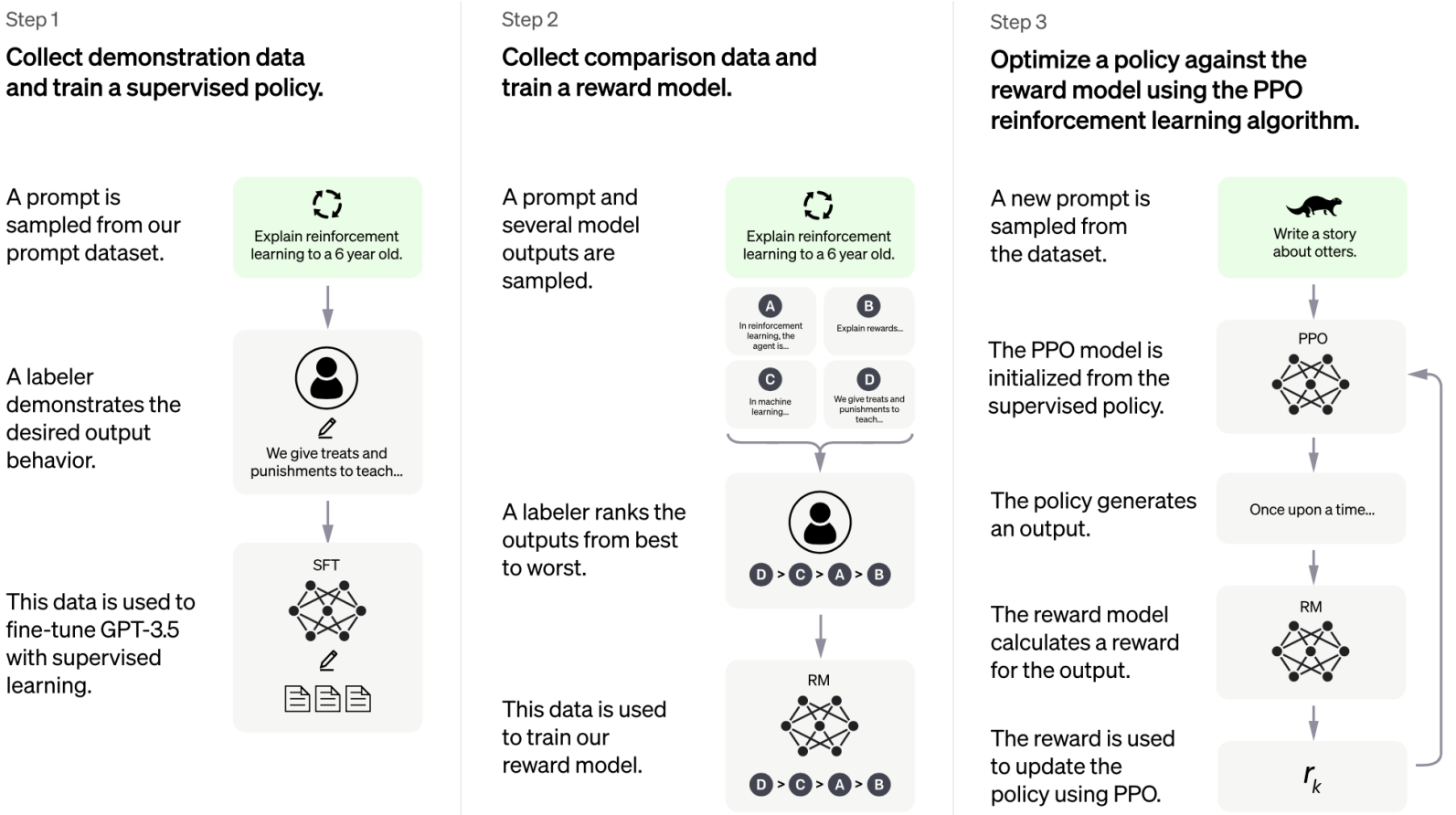
- *Learning through experience/data to make good decisions under uncertainty.*

Board Game Go



AlphaGo (2016): RL mastered Go — more legal board positions than atoms in the universe — defeating 18-time world champion Lee Sedol 4–1.

ChatGPT



RLHF: human feedback becomes the reward signal that fine-tunes language models toward preferred responses.

What RL Generally Involves

- Optimization
- Delayed consequences
- Exploration
- Generalization

Optimization

- Goal is to find an optimal way to make decisions
  - Yielding best outcomes or at least very good outcomes
- Explicit notion of decision utility
- Example: finding minimum distance route between two cities given network of roads

Delayed Consequences

- Decisions can impact things much later
- Introduces two challenges:
  - When planning: decisions involve reasoning about not just immediate benefit of decision, but also longer-term implications
  - When learning: temporal credit assignment is hard (what caused later high or low rewards?)

Exploration

- Learning about the world by making decisions
- Decisions impact what we learn about
  - Only get reward for decision made
  - Don’t know what would have happened for other decision

Generalization

- A policy maps states to actions — but it must act well in states it has never seen before
- Far too many situations to memorize one-by-one (every camera image, every board position)
- So we learn a function that generalizes across states, often a neural network
- Good generalization is what lets RL scale beyond toy problems

Problems where RL is Powerful

- No examples of desired behavior
  - Goal is to go beyond human performance, or there's no existing data for a task
- Enormous search or optimization problem with delayed outcomes

Lecture Outline

Reinforcement Learning: What it is, where it's used

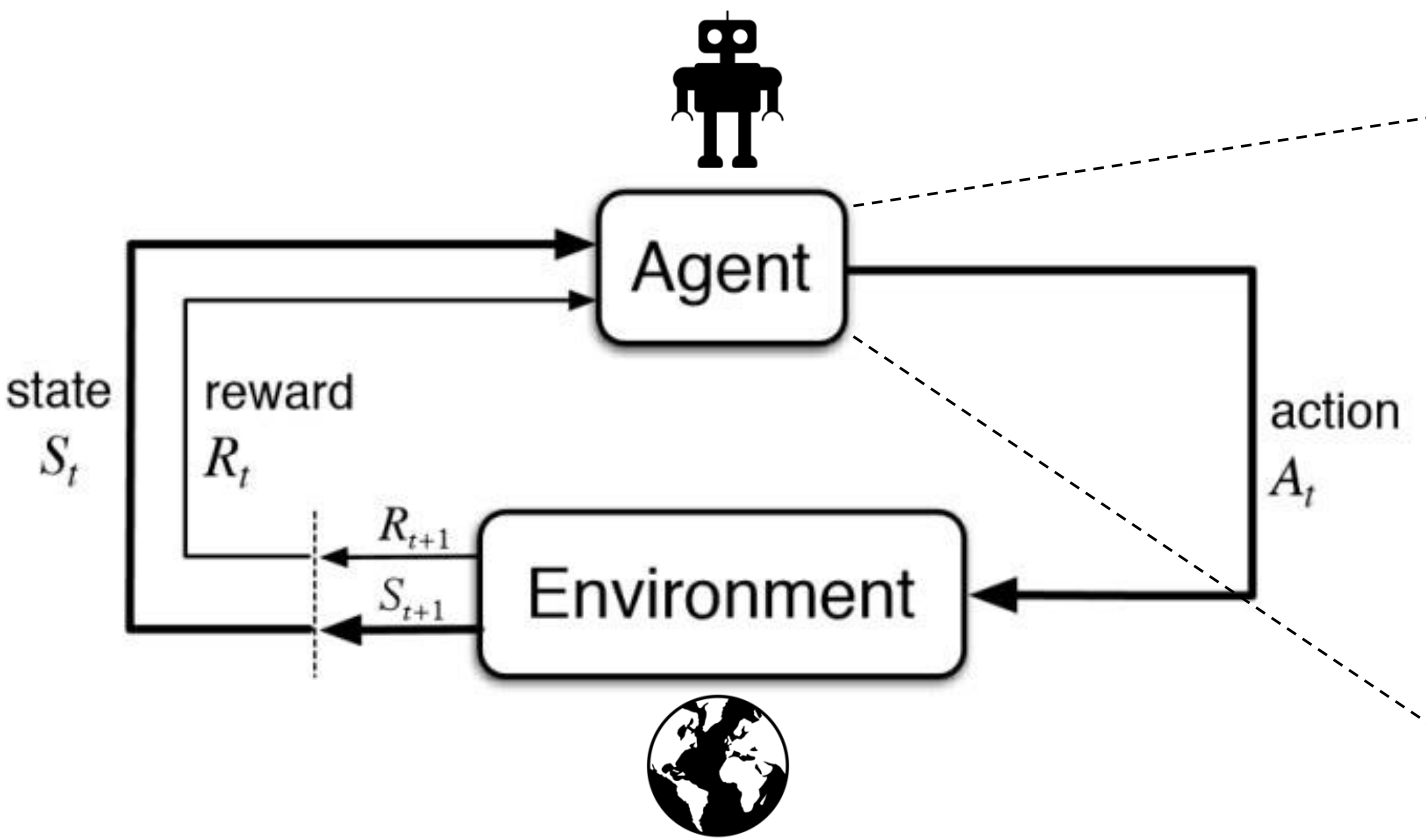
Recap: A Formalism for Sequential Decision Making

Value Functions and the Bellman Equation

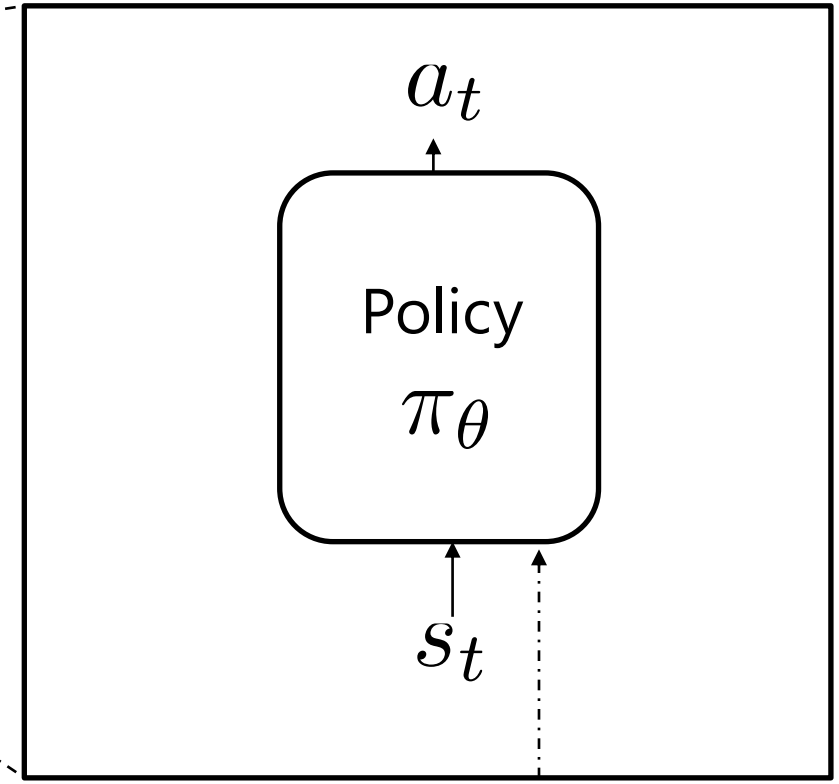
Planning with a known model: value iteration

Q-learning, exploration, and modern RL

Recap: Reinforcement Learning Formalism



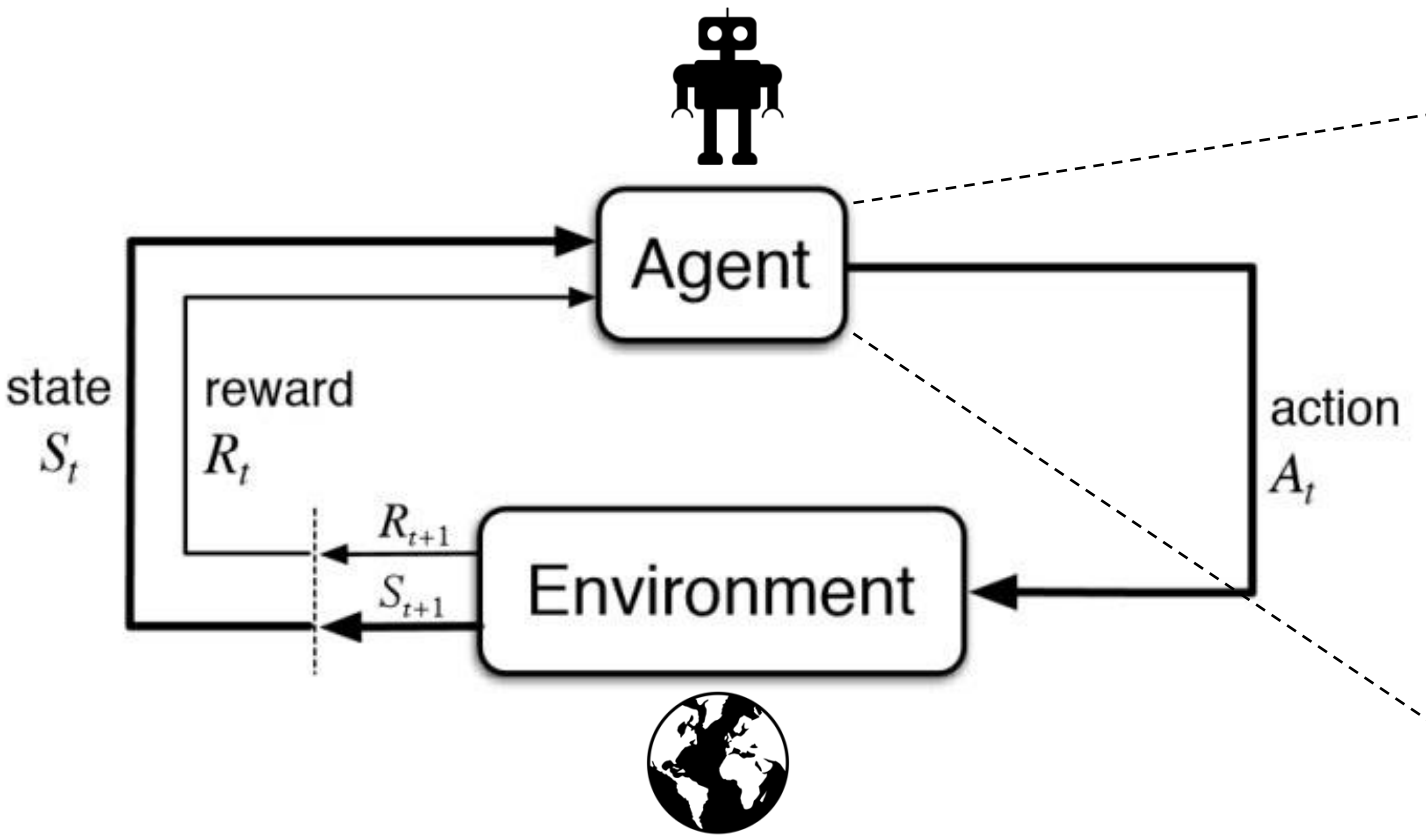
Rules for choosing actions



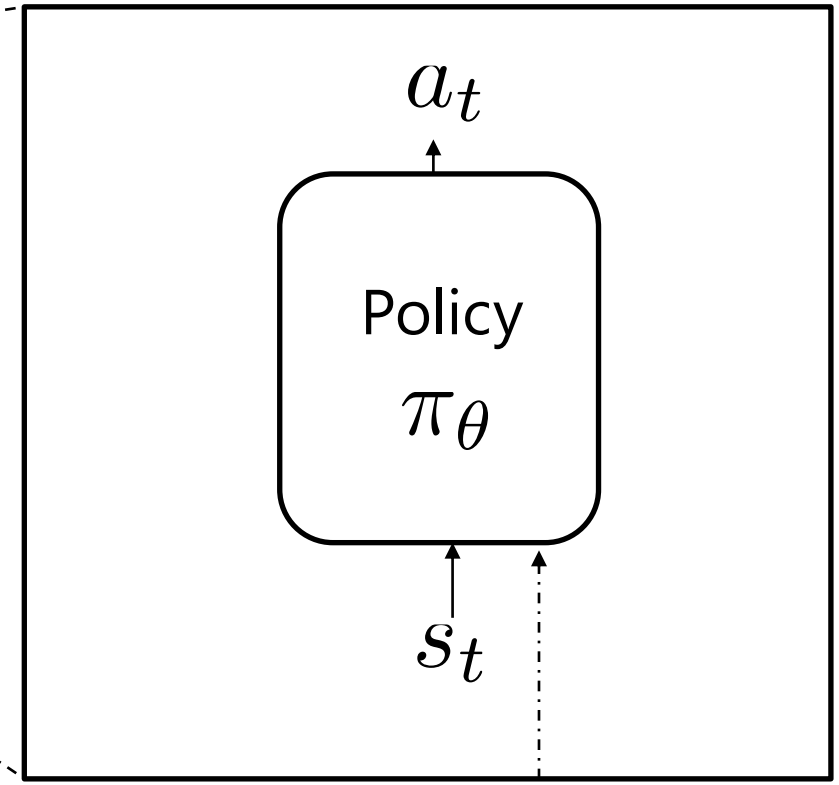
Maximize the sum of expected rewards under policy

Needs to be learned

Reinforcement Learning Formalism



Rules for choosing actions

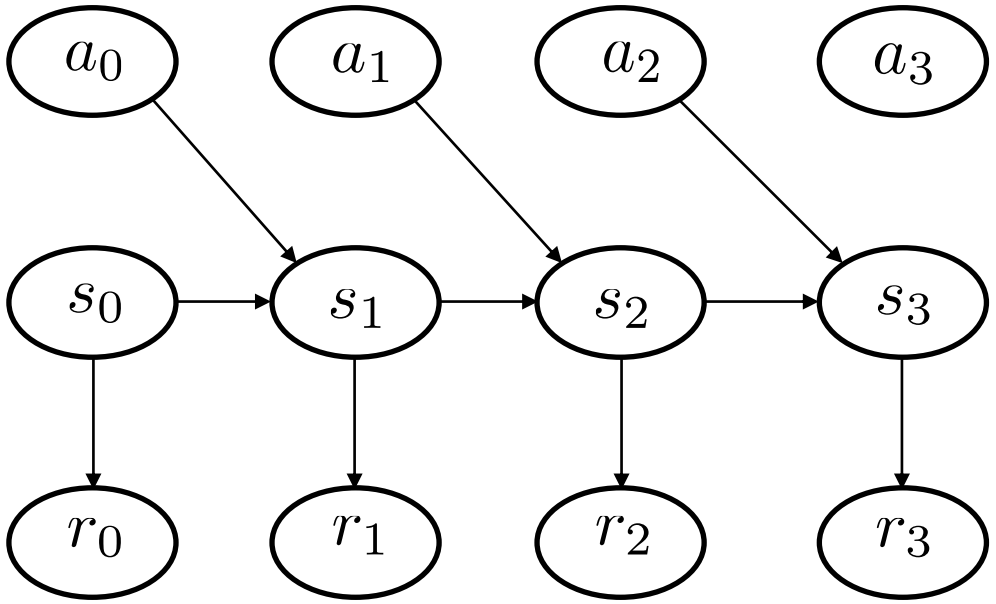


$$\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right]$$

Trajectory sampled using policy

Needs to be learned

Recap: Framework for Sequential Decision Making - Markov Decision Process



States: $\mathcal{S}$

Actions: $\mathcal{A}$

Rewards: $\mathcal{R}$

Transition Dynamics - $p(s_{t+1}|s_t, a_t)$

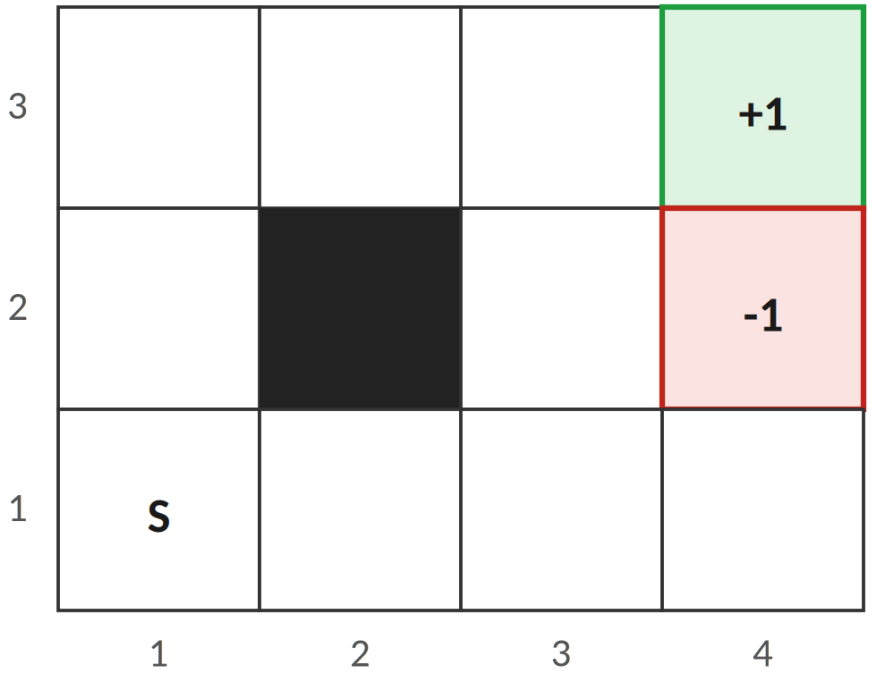
Markov property $$P(s_{t+1} \mid s_t, a_t, \ldots, s_0, a_0) = P(s_{t+1} \mid s_t, a_t)$$

Trajectory $$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T)$$

Key: MDPs obey the Markov property

The next state depends only on the current state and action, not the full history

Running Example



A mobile robot on a grid

State: which cell the robot is in

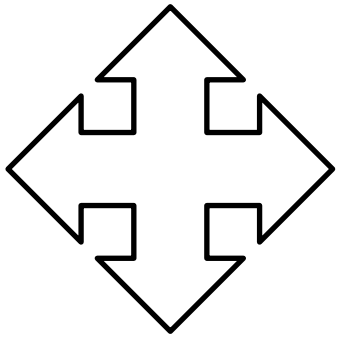
Actions: up, down, left, right

Reward: +1 at goal, -1 at hazard,
0 elsewhere

Motion is noisy

Intended direction: 80%

Slips left or right: 10% each



Policy: the Agent's Strategy

- A policy is the agent's strategy: a rule for choosing an action in every state
- **Deterministic:** $\pi(s)$ maps each state to a single action
- **Stochastic:** $\pi(a|s)$ gives a probability distribution over actions
- In our gridworld: a policy says which way to move from every cell
- The goal of RL is to find a policy that maximizes expected return — the optimal policy π^*

Lecture Outline

Reinforcement Learning: What it is, where it's used

Recap: A Formalism for Sequential Decision Making

Value Functions and the Bellman Equation

Planning with a known model: value iteration

Q-learning, exploration, and modern RL

Reward and Return

$$G_t = r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k\, r_{t+k}$$

The agent does not maximize the next reward; it maximizes the return Gt, the total reward from time t onward.

- γ in (0, 1) is the discount factor: a reward now is worth more than the same reward later, and it keeps the sum finite.

- γ near 0 is short-sighted; γ near 1 is far-sighted.

- Sparse reward (+1 only at the goal) is honest but slow to learn from; dense reward (rises as the robot nears the goal) is easier but encodes your assumptions.

Value Functions: V and Q

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\, G_t \mid s_t = s \,]$$

value of a state

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\, G_t \mid s_t = s,\ a_t = a \,]$$

value of a state-action pair

- V(s): expected return if the robot starts in cell s and follows its policy.
- Q(s, a): expected return if it takes action a in cell s, then follows its policy.
- **Q is what we act on: in each cell, compare the four actions and pick the highest Q.**
- In the gridworld, V is high near the goal and low near the hazard; Q tells you which direction is best from each cell.

Bellman Equation

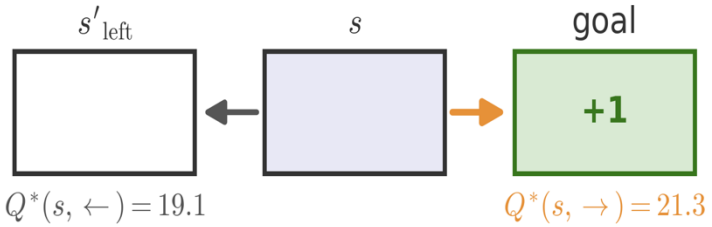
$$V^*(s) = \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right]$$

Value = one step + the rest:

the value of a state is the best action's immediate reward plus the discounted value of where you land, averaged over the noisy outcomes.

This is the same optimal-substructure principle behind Dijkstra and dynamic programming: an optimal path is built from optimal sub-paths.

Bellman optimality: act greedily w.r.t. Q^*



$$V^*(s) = \max_a Q^*(s, a) = 21.3$$

$$Q^*(s, a) = \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V^*(s') \right]$$

Lecture Outline

Reinforcement Learning: What it is, where it's used

↓

Recap: A Formalism for Sequential Decision Making

↓

Value Functions and the Bellman Equation

↓

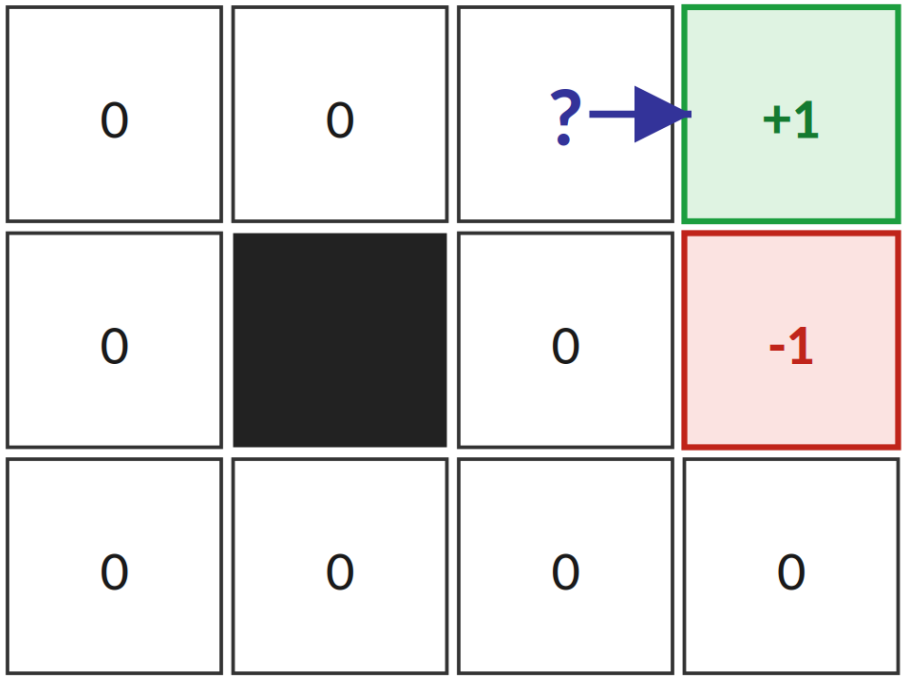
Planning with a known model: value iteration

↓

Q-learning, exploration, and modern RL

Value Iteration: One Iteration Example

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} P(s' \mid s, a) \left[R(s, a, s') + \gamma V_i(s') \right]$$



Try action "move right":

$0.8 \cdot [0 + 0.9 \times 1]$ intended

$+ 0.1 \cdot [0 + 0.9 \times 0]$ slip up

$+ 0.1 \cdot [0 + 0.9 \times 0]$ slip down

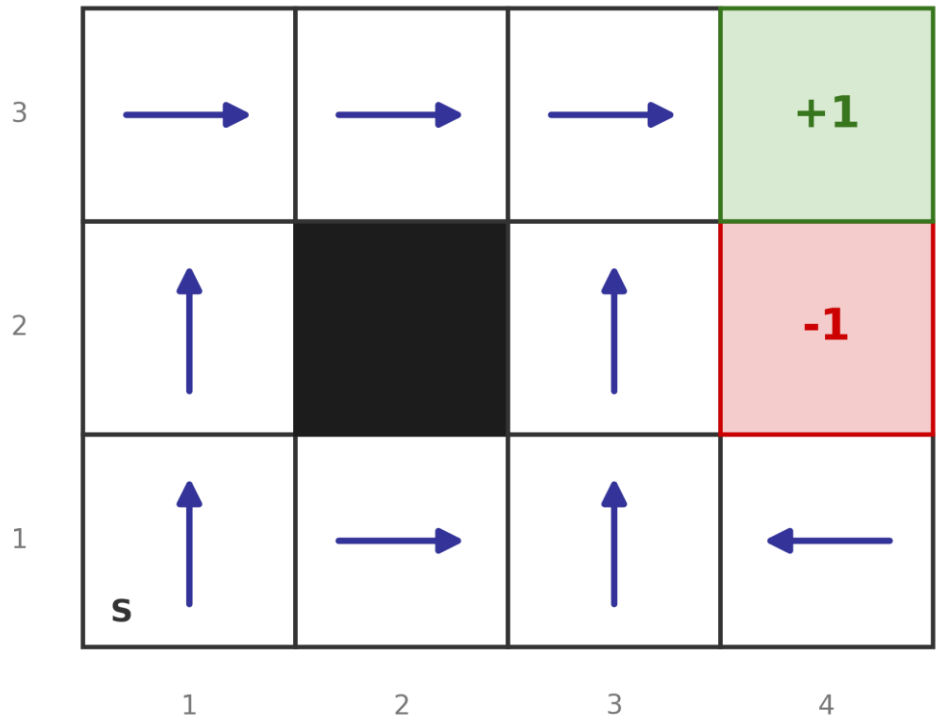
= 0.72 (noise = 0.2, γ = 0.9)

Value Iteration: Repeat till it settles

Apply the update to every cell, repeat. Value spreads outward from the goal, one ring per sweep, exactly like search expanding from a goal.

When the values stop changing, that fixed point is the optimal value V*; the optimal policy is then "step toward the best neighbor."

The Optimal Policy



- Once values converge, read off the policy greedily: from each cell, step toward the best-valued neighbor
- The arrows route around the −1 hazard, not straight through it
- The cell below the hazard points away from it — a short detour beats risking the penalty
- This policy π* is the answer we were solving for all along

Planning vs Learning

- Two ways to find a good policy — which one you use depends on what you know about the world:

Known model → Planning

Value iteration

You have P(s'|s,a) and R(s,a,s')

Compute the optimal policy directly, no real-world trial needed

Sweep the Bellman update until values converge

Unknown model → Learning

Q-learning

You do not know the dynamics or rewards in advance

Act, observe outcomes, and learn from sampled experience

Must balance exploration with exploitation

Lecture Outline

Reinforcement Learning: What it is, where it's used

Recap: A Formalism for Sequential Decision Making

Value Functions and the Bellman Equation

Planning with a known model: value iteration

Q-learning, exploration, and modern RL

What if we don’t know the model?

Value iteration needs P(s’ | s, a) and R(s, a, s’) in advance, like a search algorithm needs the graph and edge costs.

A real robot rarely has this. It does not know in advance how its noisy motion will turn out, or where the hazard is.

But it can act and observe one outcome at a time: from state s it takes action a, and sees reward r and next state s’.

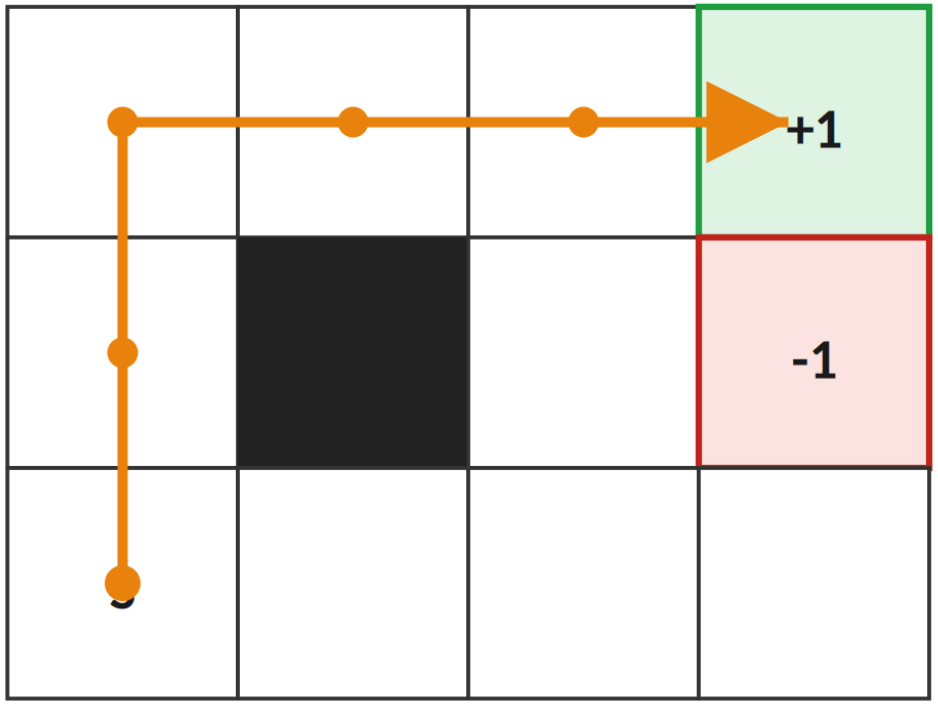
Idea: use each observed (s, a, r, s’) as a sample of what the model would have told us, and update our value estimate toward it.

Q-Learning: The Update Rule

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

- When P and R are unknown, we cannot compute the Bellman expectation — but we can ACT and observe one outcome at a time: (s, a, r, s′)
- Replace the expected value with a single sampled estimate of $r + \gamma \max_{a'} Q(s',a')$, and nudge Q(s,a) toward it
- **Step size α:** how aggressively each update moves Q
- **TD error:** the bracketed term — the gap between the new sampled estimate and the old Q
- With enough exploration of all (s,a), Q converges to Q* — no model needed

Q-Learning in the Gridworld



One episode

The robot acts, sometimes randomly, and reaches the goal (reward +1).

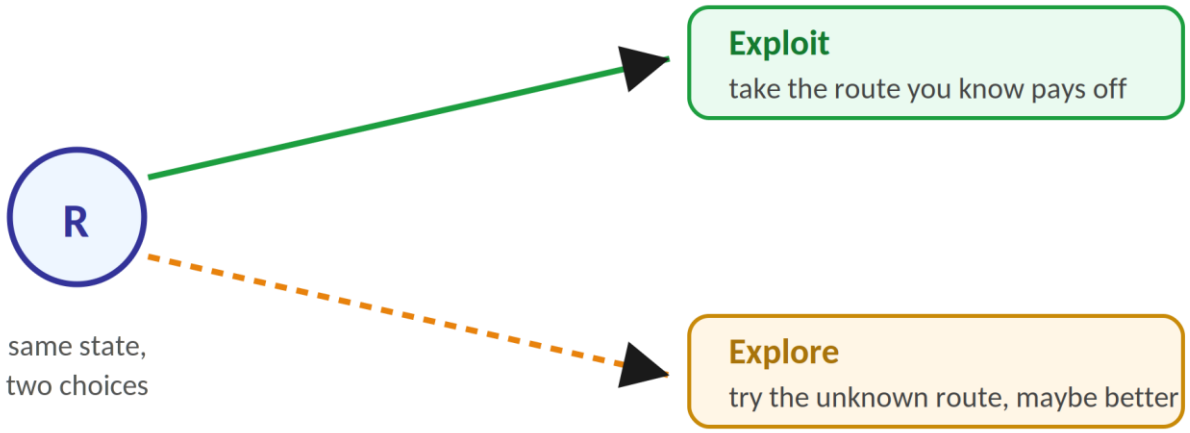
Each step it visited gets updated:

Q(cell, action) moves toward reward + best next Q.

Over many episodes, Q-values spread back from the goal, and the greedy policy becomes the shortest safe path.

No map needed: the robot learns from the outcomes of its own actions.

Exploration vs. Exploitation



$$\pi(s) = \begin{cases} \arg\max_a \hat{Q}(s,a) & \text{with prob. } 1-\epsilon \\ \text{random action} & \text{with prob. } \epsilon \end{cases}$$

If the robot only ever exploits its first lucky route, it may never discover the shorter one; if it only explores, it never commits.

- Exploit: take the action currently believed best.
- Explore: try an uncertain action to learn whether it is better.
- ε-greedy: act greedily with probability 1−ε, randomly with probability ε, and shrink ε as learning proceeds.

ε-Greedy Exploration

- **With probability ε:** take a random action (explore)
- **With probability 1 – ε:** take the greedy action argmax_a Q(s,a) (exploit)
- Typically start with high ε (e.g., 1.0) and decay over time (e.g., toward 0.05) — explore broadly early, exploit what you've learned later
- **Pros:** dead simple, no tuning beyond ε, works well in practice
- **Cons:** undirected — doesn't know which states need more exploration; can be slow in large MDPs
- This is the default exploration policy paired with Q-learning

Policy Gradient Methods

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t\right]$$

- A second major family of RL methods: skip the Q-table and parameterize the policy πθ(a|s) directly — usually as a neural network
- Define J(θ) = expected return under πθ, then climb the gradient: increase log-prob of actions that led to high return, decrease the rest (REINFORCE)
- **Why it matters:** natural fit for continuous actions, large/stochastic policies, and end-to-end training of deep networks
- **Real-world example:** RLHF (slide on ChatGPT earlier) trains language models with a policy gradient variant (PPO)

Deep Reinforcement Learning

- Real problems have huge or continuous state spaces — pixels in a game, joint angles on a robot, a board position in Go. A Q-table is hopeless.
- Replace the table with a function approximator $Q\theta(s,a)$ or $\pi\theta(a|s)$, typically a neural network, and train its parameters by gradient descent
- **Deep Q-Networks (DQN):** regress $Q\theta(s,a)$ toward the Bellman target $r + \gamma \max_{a'} Q(s',a')$; stabilized with experience replay and target networks
- **Deep policy gradients:** the same idea on the policy side — gradient methods (PPO, A3C, SAC) scaled to neural policies
- This is how RL works in practice today — Atari (DQN), Go (AlphaGo), robotics, RLHF for language models

Why RL is Hard in Practice

Sample inefficiency

Real robots cannot run millions of trials; hardware wears out and time is limited.

Sparse reward

With reward only at success, random actions almost never trigger it, so there is little to learn from.

Safety

Exploration means trying unknown actions; a real robot can damage itself or its surroundings.

Sim-to-real

Training in simulation is fast and safe, but a policy must still transfer to an imperfect real world.

These are why robotics often combines RL with simulation, with imitation as a warm start, and with carefully shaped, safe exploration.

Where this is going

- Real-world challenges: sample efficiency, safety, sim-to-real transfer, and learning from logged data are open problems.

- Legged locomotion: quadrupeds and humanoids learn robust walking and recovery in simulation, then transfer to hardware.

- Manipulation: in-hand reorientation and contact-rich tasks learned where demonstrations are hard to collect.

- **Combining paradigms: imitation to start, RL to refine, and a known model for planning where one exists.**

Summary

- **RL = learning to make good decisions under uncertainty** — it combines optimization, delayed consequences, exploration, and generalization.
- **MDPs** formalize the problem: states, actions, rewards, and transition dynamics, with the Markov property.
- **Value functions** measure expected return: V(s) for a state, Q(s,a) for a state-action pair. We act greedily w.r.t. Q.
- **The Bellman equation** ties a state's value to its neighbors: one step of reward plus the discounted value of where you land.
- **Value iteration** solves a known MDP by applying the Bellman update until values converge to V*, giving the optimal policy.
- **When the model is unknown**, we learn from sampled experience: Q-learning updates Q from (s,a,r,s′) tuples with no model needed. Policy gradients optimize $\pi\theta(a|s)$ directly — the two main families of modern RL, both scaled by neural function approximation.