COM SCI 188

Intro to Robotics

Lecture 11

Yuchen Cui
Winter 2026

Agenda

- Announcements
- Recap: Motion Planning

- TODAY: **Sequential Decision Making**
  - Markov Decision Processes
  - Dynamic Programming
  - Reinforcement Learning

Announcements

- Problem Set 3 due Friday
- **Coding Assignment 3**
  - SLAM + motion planning
  - due 2/23 (Monday after midterm)

Announcements

Midterm is next Thursday 2/19 (in-class)

- Practice problems this Friday
- Review Session next Tuesday
- You are allowed to bring 1 double-sided hand-written notes

Announcements

Final project

Join teams on Bruinlearn
Project proposal (due 2/27)

- 1-page description of the goal and success criteria
- Submit only 1 per team
- Open project: make sure you get your project checked by TA or me

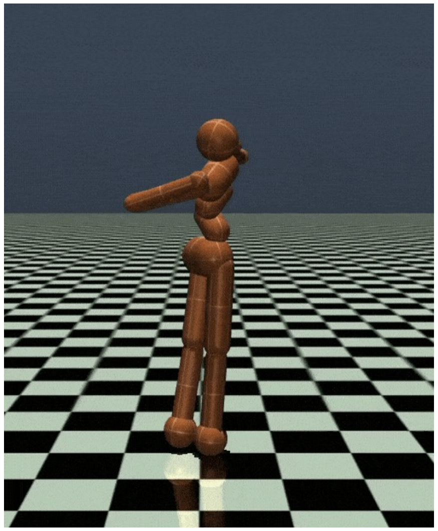
Sequential Decision Making

Markov Decision Processes & Reinforcement Learning

Slides adapted from USC CSCI 445: Introduction to robotics
Credit: Erdem Biyik, Heather Culbertson

Sequential Decision Making

A sequential decision-making problem involves making a series of interdependent decisions over time to optimize a long-term objective,



Markov decision processes (MDP)

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot | s_t, a_t)$

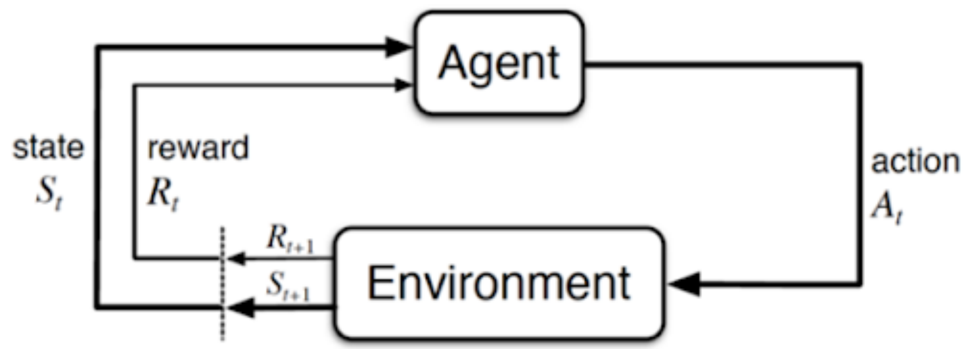
Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$

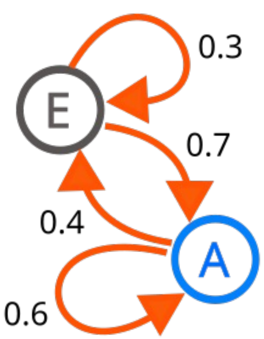
Goal:

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



Markov Property

Markov property refers to the memoryless property of a stochastic process, which means that its future evolution is independent of its history.



In probability theory and statistics, a **Markov chain** or **Markov process** is a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

Andrey Markov

Decision making in *deterministic* systems

State: $s_t \in \boldsymbol{\mathcal{S}}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

Transition: $s_{t+1} = f_t(s_t, a_t)$

Total reward: $J(s_0; a_0, \dots, a_{T-1}) = r_T(s_T) + \sum_{t=0}^{T-1} r_t(s_t, a_t)$

Decision making problem:

$$J^*(s_0) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t), t=0,1,\dots,T-1} J(s_0; a_0, \dots, a_{T-1})$$

Principle of optimality

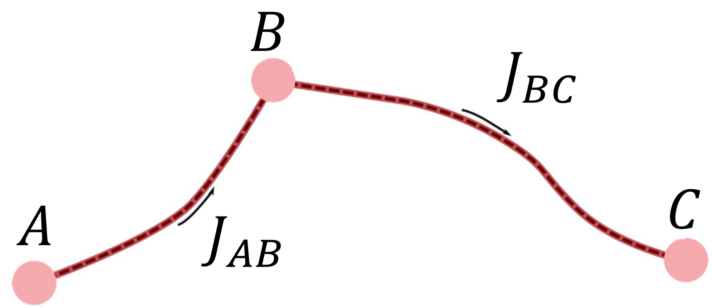
It's the key concept behind the dynamic programming approach.

Suppose $A - B - C$ is the optimal path from A to C.

First segment reward: J_{AB}

Second segment reward: J_{BC}

Optimal reward $J_{AC}^{*} = J_{AB} + J_{BC}$



Principle of optimality (Deterministic)

Suppose $(a_0^*, a_1^*, \dots, a_{T-1}^*)$ is an optimal solution to the decision making problem for an initial state s_0^*, and the systems evolves as $(s_0^*, s_1^*, \dots, s_T^*)$ for this initial state and action sequence.

Then, an optimal solution to the subproblem for moving from state s_t^* at time t until time T is $(a_t^*, a_{t+1}^*, \dots, a_{T-1}^*)$.

Tail of an optimal solution = Optimal for the tail subproblem

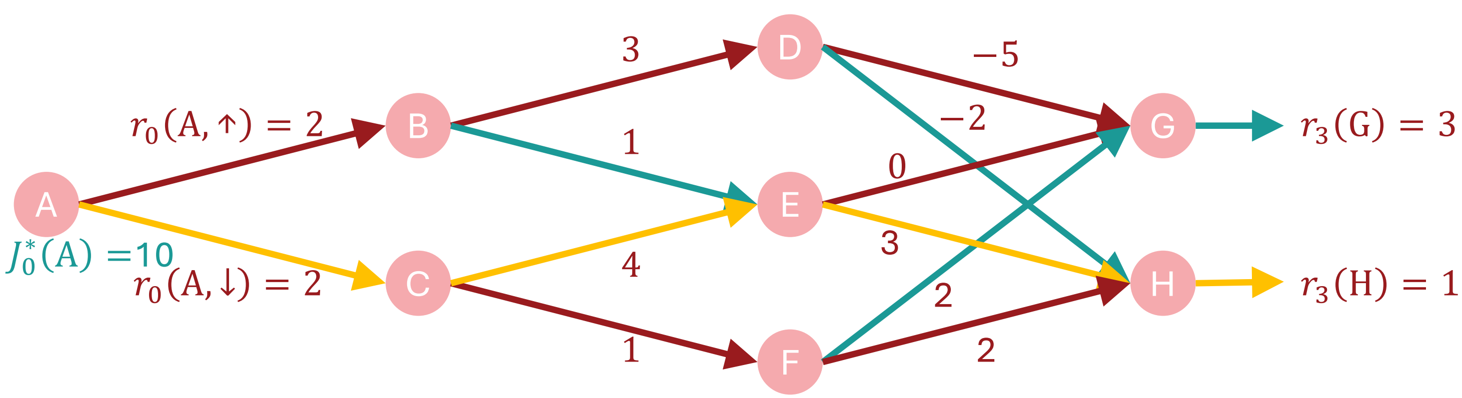
Dynamic programming (deterministic)

$J_T^*(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{\mathcal{S}}$

for $t = T - 1$ to 0 do

$\quad J_t^*(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} r_t(s_t, a_t) + J_{t+1}^*(f_t(s_t, a_t))$, for all $s_t \in \boldsymbol{\mathcal{S}}$

return $J_0^*(\cdot), J_1^*(\cdot), \dots, J_T^*(\cdot)$



Decision making in stochastic systems

This is a random variable.

State: $s_t \in \boldsymbol{\mathcal{S}}$

Action: $a_t \in \boldsymbol{\mathcal{A}}(s_t)$

Transition: $s_{t+1} = f_t(s_t, a_t, w_t)$, or $s_{t+1} \sim P(\cdot \mid s_t, a_t)$

Policies: $\pi = (\pi_0, \pi_1, \ldots, \pi_{T-1})$ where $a_t = \pi_t(s_t)$

We introduce policies since we will find an optimal *closed-loop* policy.

Expected total reward:

$$J_\pi(s_0) = \mathbb{E}_{w_0, w_1, \ldots w_{T-1}}\left[r_T(s_T) + \sum_{t=0}^{T-1} r_t(s_t, \pi_t(s_t), w_t)\right]$$

Decision making problem: $J^*(s_0) = \max_\pi J_\pi(s_0)$

Principle of optimality (stochastic)

Suppose $(\pi_0^*, \pi_1^*, \dots, \pi_{T-1}^*)$ is an optimal solution to the decision making problem and assume state s_t is reachable.

Then, an optimal solution to the subproblem for moving from state s_t at time t until time T is $(\pi_t^*, \pi_{t+1}^*, \dots, \pi_{T-1}^*)$.

Tail of optimal policies = Optimal for the tail subproblem

Dynamic Programming (stochastic)

$J_T(s_T) = r_T(s_T)$, for all $s_T \in \boldsymbol{\mathcal{S}}$

for $t = T-1$ **to** 0 **do**

$$J_t(s_t) = \max_{a_t \in \boldsymbol{\mathcal{A}}(s_t)} \mathbb{E}_{w_t}\left[r_t(s_t, a_t, w_t) + J_{t+1}(f_t(s_t, a_t, w_t))\right], \text{ for all } s_t \in \boldsymbol{\mathcal{S}}$$

return $J_0(\cdot), J_1(\cdot), \ldots, J_T(\cdot)$

Comments

DP in stochastic systems suffers from the same problems as DP in deterministic systems.

Also, modeling transitions perfectly is not always possible.

Infinite horizon MDP's

We are looking at the infinite horizon case, since it makes stationary policies optimal.

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot \mid s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi: \boldsymbol{\mathcal{S}} \rightarrow \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{S}} \rightarrow \Delta\boldsymbol{\mathcal{A}}$

We removed the dependence on the state, although that's also creates interesting research questions.

Infinite horizon MDP's

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot | s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

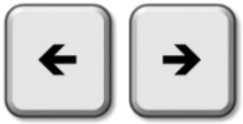
Policy: $\pi: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{S}} \to \Delta\boldsymbol{\mathcal{A}}$

Goal: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\big(s_t, \pi(s_t)\big)\right]$

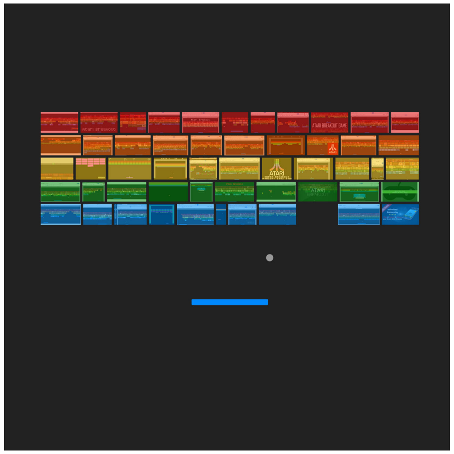
An example MDP

Goal: Achieve a high score in the Atari game “Breakout”

States: Image of the current screen (?)

Actions: Left and right actions 

Reward: Change in the score of the game



Another example MDP

Goal: Make an RC helicopter fly and perform some maneuvers

States: Sensory input of the helicopter

Actions: Control inputs

Reward: Positive for the maneuvers, negative for crashing

(This is usually what we need to hand-design)

Not that easy!

Goal: Make an RC helicopter fly and perform some maneuvers

States: Sensory input of the helicopter

Actions: Control inputs

Reward: Positive for the maneuvers, negative for crashing

This is a very naïve reward function. They instead learned the reward from expert demonstrations. We will cover this topic in a few weeks.

Infinite horizon MDP's

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot \mid s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi : \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$ or $\pi : \boldsymbol{\mathcal{S}} \to \Delta\boldsymbol{\mathcal{A}}$

Goal:

Reinforcement learning tries to solve this problem.

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

Partially observable MDP's

State: $s \in \boldsymbol{\mathcal{S}}$

Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot | s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi: \boldsymbol{\mathcal{O}} \to \boldsymbol{\mathcal{A}}$ or $\pi: \boldsymbol{\mathcal{O}} \to \Delta\boldsymbol{\mathcal{A}}$

Goal:

Observation: $o \in \boldsymbol{\mathcal{O}}$

Observation Model: $o_t \sim \Omega(\cdot | s_t)$

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

Value functions

State value function: $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$

State-action value function: $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$

$$\begin{cases} V^{\pi}(s) = R\big(s, \pi(s)\big) + \gamma \mathbb{E}_{s' \sim P(\cdot \mid s, \pi(s))}[V^{\pi}(s')] \\ Q^{\pi}(s,a) = R(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot \mid s,a), a' \sim \pi(s')}[Q^{\pi}(s',a')] \end{cases}$$

For any stationary policy, these have unique solutions.
Hint: Think of it as a system of linear equations.

Bellman equations

State value function: $V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s]$

State-action value function: $Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a]$

Current Value = Expected [Immediate Reward + (Discount Factor × Value of Next State)]

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

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

Value iteration

Idea: Take Bellman equation and iterate until it converges.
It does converge because it is a contractive mapping.

$V_0(s) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$
for $k = 0,1, \ldots$ **until** convergence:
for all $s \in \boldsymbol{\mathcal{S}}$:

$$V_{k+1}(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V_k(s') \right)$$

Each iteration is $O(|\boldsymbol{\mathcal{S}}|^2 |\boldsymbol{\mathcal{A}}|)$.

Value iteration

Idea: Take Bellman equation and iterate until it converges.
It does converge because it is a contractive mapping.

$Q_0(s,a) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$
for $k = 0,1,\ldots$ **until** convergence:
  for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$:
    $Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) \max_{a' \in \boldsymbol{\mathcal{A}}} Q_k(s',a')$

Each iteration is $O(|\boldsymbol{\mathcal{S}}|^2 |\boldsymbol{\mathcal{A}}|^2)$.

Policy iteration

Initialize a random policy π_0.

for $k = 0,1, \ldots$ until convergence:

Solve the following system for V^{π_k}:

This is called **policy evaluation.**
It is $O(|\mathcal{S}|^3)$.

$$V^{\pi_k}(s) = \mathbb{E}_{a \sim \pi_k(s)}\left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^{\pi_k}(s')\right]$$

This is called **policy improvement.**
It is $O(|\mathcal{S}|^2|\mathcal{A}|)$.

for all $s \in \mathcal{S}$:

$$\pi_k(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s,a) V^{\pi_k}(s')\right)$$

Value iteration vs policy iteration

- Both converge.
- Policy iteration requires more complex implementation.
- In practice, policy iteration usually converges faster.

<break>

Markov decision processes (MDP)

State: $s \in \boldsymbol{\mathcal{S}}$

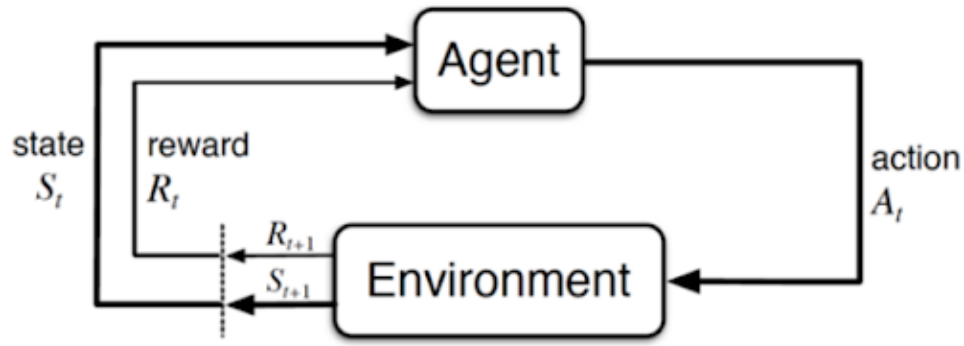
Action: $a \in \boldsymbol{\mathcal{A}}$

Transition: $s_{t+1} \sim P(\cdot | s_t, a_t)$

Reward: $r_t = R(s_t, a_t)$

Discount: $\gamma \in [0,1)$

Policy: $\pi: \boldsymbol{\mathcal{S}} \to \boldsymbol{\mathcal{A}}$



Goal:

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

Value iteration

Idea: Take Bellman equation and iterate until it converges.

$V_0(s) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$
for $k = 0,1, \ldots$ **until** convergence:
  for all $s \in \boldsymbol{\mathcal{S}}$:
    $V_{k+1}(s) = \max_a \left(R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V_k(s') \right)$

$Q_0(s,a) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$
for $k = 0,1, \ldots$ **until** convergence:
  for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$:
    $Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) \max_{a' \in \boldsymbol{\mathcal{A}}} Q_k(s',a')$

Policy iteration

Initialize a random policy π_0.

for $k = 0{,}1, \ldots$ **until** convergence:

Solve the following system for V^{π_k}: # policy evaluation.

$$V^{\pi_k}(s) = \mathbb{E}_{a \sim \pi_k(s)}\left[R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V^{\pi_k}(s')\right]$$

for all $s \in \boldsymbol{\mathcal{S}}$: # policy improvement.

$$\pi_k(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V^{\pi_k}(s')\right)$$

Value Iteration

$V_0(s) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$
for $k = 0,1, \ldots$ until convergence:
for all $s \in \boldsymbol{\mathcal{S}}$:
$V_{k+1}(s) = \max_a \big(R(s,a) +$
$\gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V_k(s')\big)$

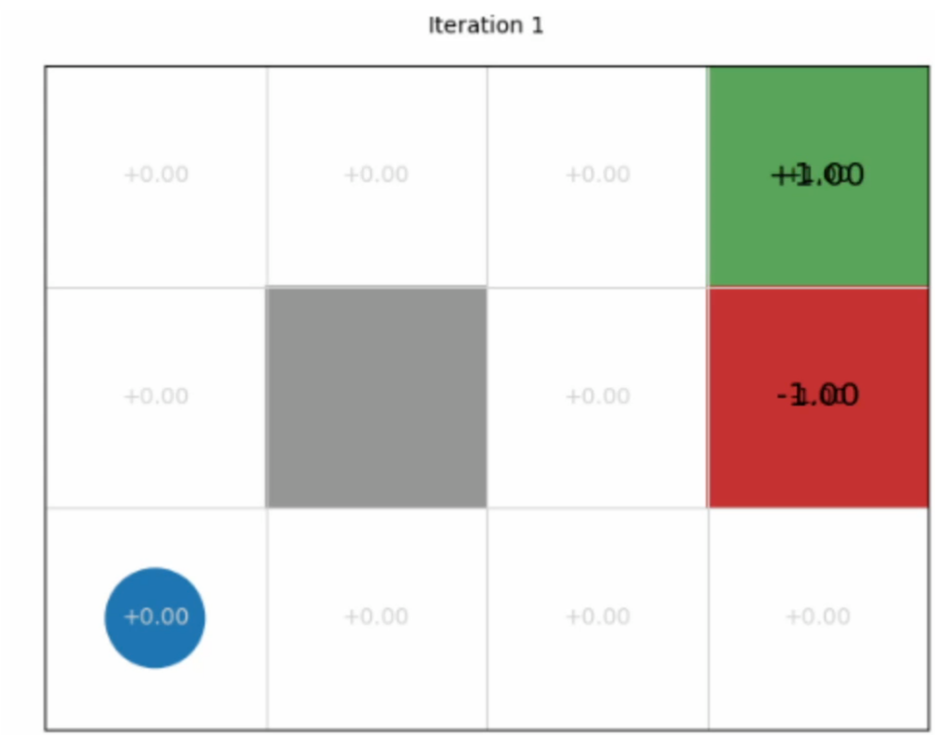
$Q_0(s,a) = 0$ for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$
for $k = 0,1, \ldots$ until convergence:
for all $s \in \boldsymbol{\mathcal{S}}$, $a \in \boldsymbol{\mathcal{A}}$:
$Q_{k+1}(s,a) = R(s,a) + \gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid$
$s,a) \max_{a' \in \boldsymbol{\mathcal{A}}} Q_k(s',a')$

Policy iteration

Initialize a random policy π_0.
for $k = 0,1, \ldots$ until convergence:
Solve the following system for V^{π_k}:
$V^{\pi_k}(s) = \mathbb{E}_{a \sim \pi_k(s)}\big[R(s,a) +$
$\gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V^{\pi_k}(s')\big]$

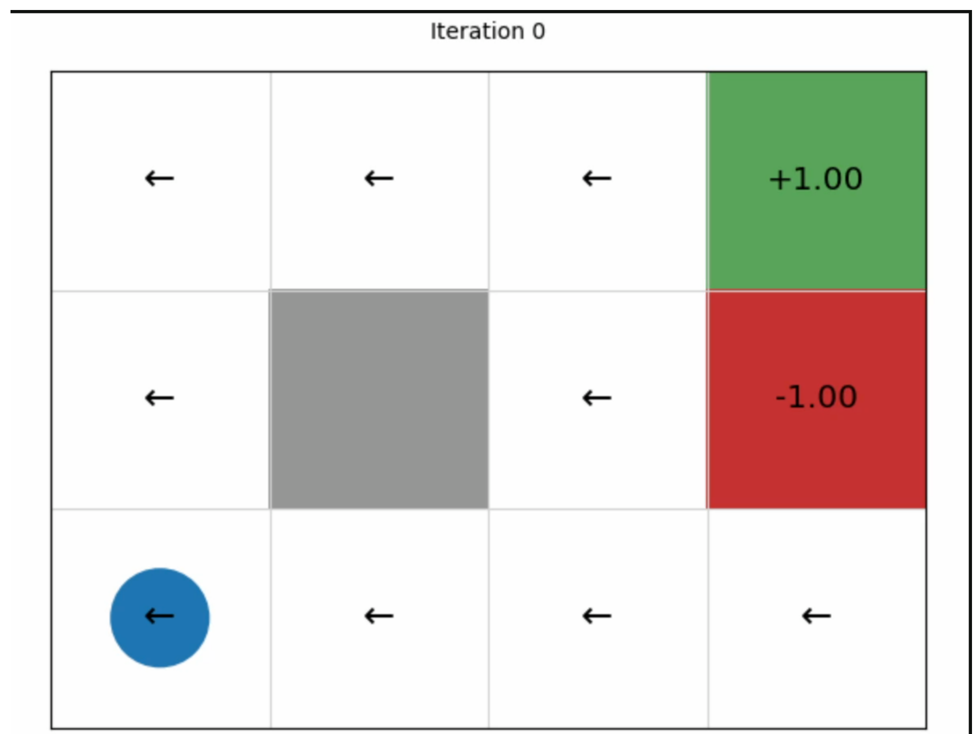
for all $s \in \boldsymbol{\mathcal{S}}$:
$\pi_k(s) = \arg\max_a \big(R(s,a) +$
$\gamma \sum_{s' \in \boldsymbol{\mathcal{S}}} P(s' \mid s,a) V^{\pi_k}(s')\big)$

Value Iteration



converges after 10 iterations

Policy iteration

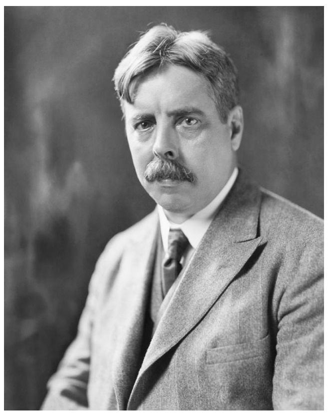


converges after 4 iterations

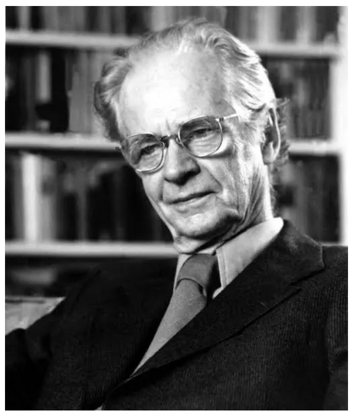
Source: https://gibberblot.github.io/rl-notes/single-agent/MDPs.html

A Brief History of Reinforcement Learning

1. Psychological Foundations (1900s–1950s)

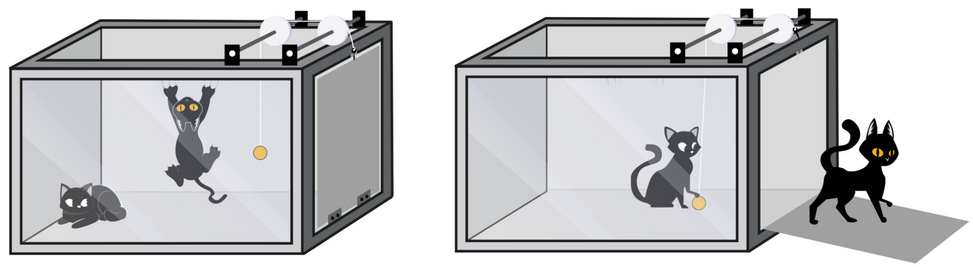


Edward Thorndike



B.F. Skinner

"Law of Effect" suggests actions followed by satisfaction are strengthened



Operant conditioning is a learning process where behaviors are strengthened or weakened by their consequences

These early concepts inspired the *trial-and-error learning* at the heart of RL.

2. Control Theory & Early AI (1950s–1970s)

Richard Bellman

Introduced *Dynamic Programming* and the *Bellman* equation, foundational for RL. Formalized optimal decision-making in MDPs.

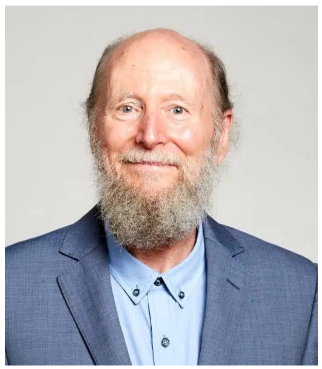
Arthur Samuel

Built one of the first learning programs (checkers), which used ideas akin to RL.

DP provided a mathematical framework; AI began applying learning to games.

3. Temporal-Difference Learning & Modern Formulation (1980s)

Andrew Barto



Richard Sutton

Charles Anderson

2024 Turing Award winners

Temporal-Difference (TD) learning (1983):
DP + supervised learning
"Learning a Guess from a Guess" (Bootstrapping)

This era established RL as a distinct field with foundational algorithms.

4. RL Meets Machine Learning (1990s–2000s)

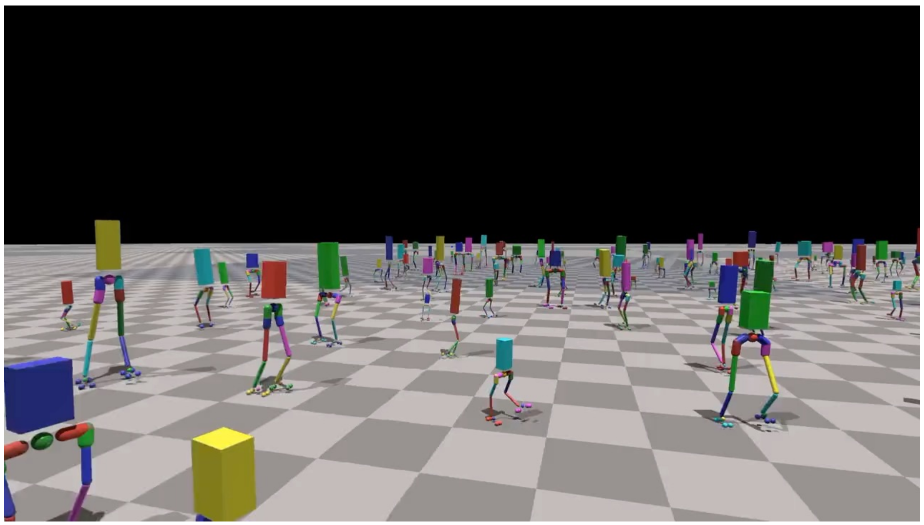
- **Policy Gradient methods** introduced for continuous action spaces.
- **Applications** expanded to robotics, games, and scheduling.
- **TD-Gammon (1992)**: Gerald Tesauro's backgammon player used TD learning and beat top human players — a milestone for RL in practice.

5. Deep Reinforcement Learning (2010s–Present)

- **Deep Q-Networks (DQN)** by DeepMind (2013–2015): Combined Q-learning with deep neural networks, achieving human-level play in Atari games.
- **AlphaGo (2016)**: Combined deep RL with Monte Carlo Tree Search to beat world champions in Go.
- **Proximal Policy Optimization (PPO), A3C, SAC**: New policy optimization methods made training more stable and efficient.
- **OpenAI Five (2019)**: Used RL to master Dota 2 in a complex multi-agent environment.
- …

Deep Reinforcement Learning in Robotics

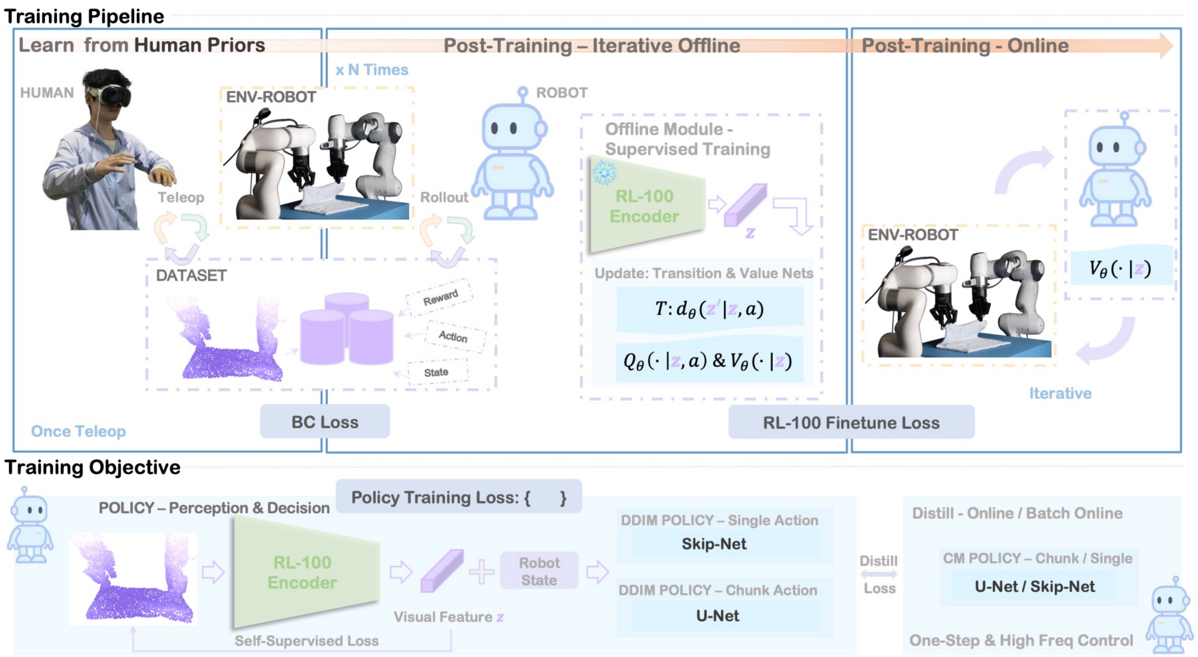
SoTA Locomotion Policies: domain randomization + RL --> sim2real



https://generalist-locomotion.github.io/

Deep Reinforcement Learning in Robotics

RL in manipulation?



https://lei-kun.github.io/RL-100/

That’s it for today!

Questions?