ECE7121 Learning-based control – 2025 Fall

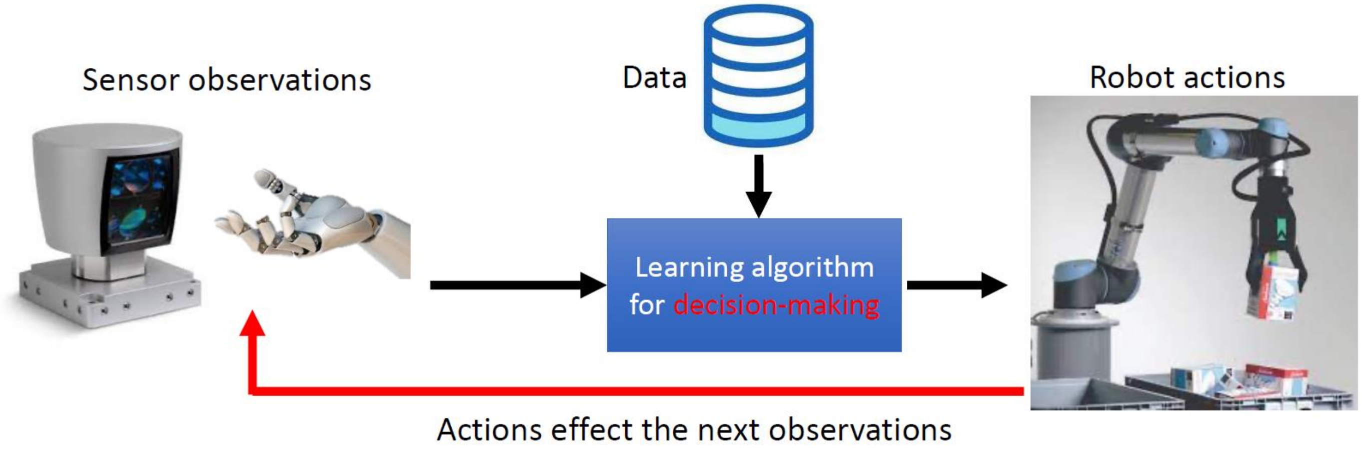
Reinforcement learning basics

INHA UNIVERSITY

Robot learning

> Learning to make sequential decisions in the physical world
  - A system need to make multiple decisions based on stream of information

> The solutions to such problems
  - imitation learning
  - offline & online RL
  - model-free & model-based RL
  - multi-task & meta RL



RL Success

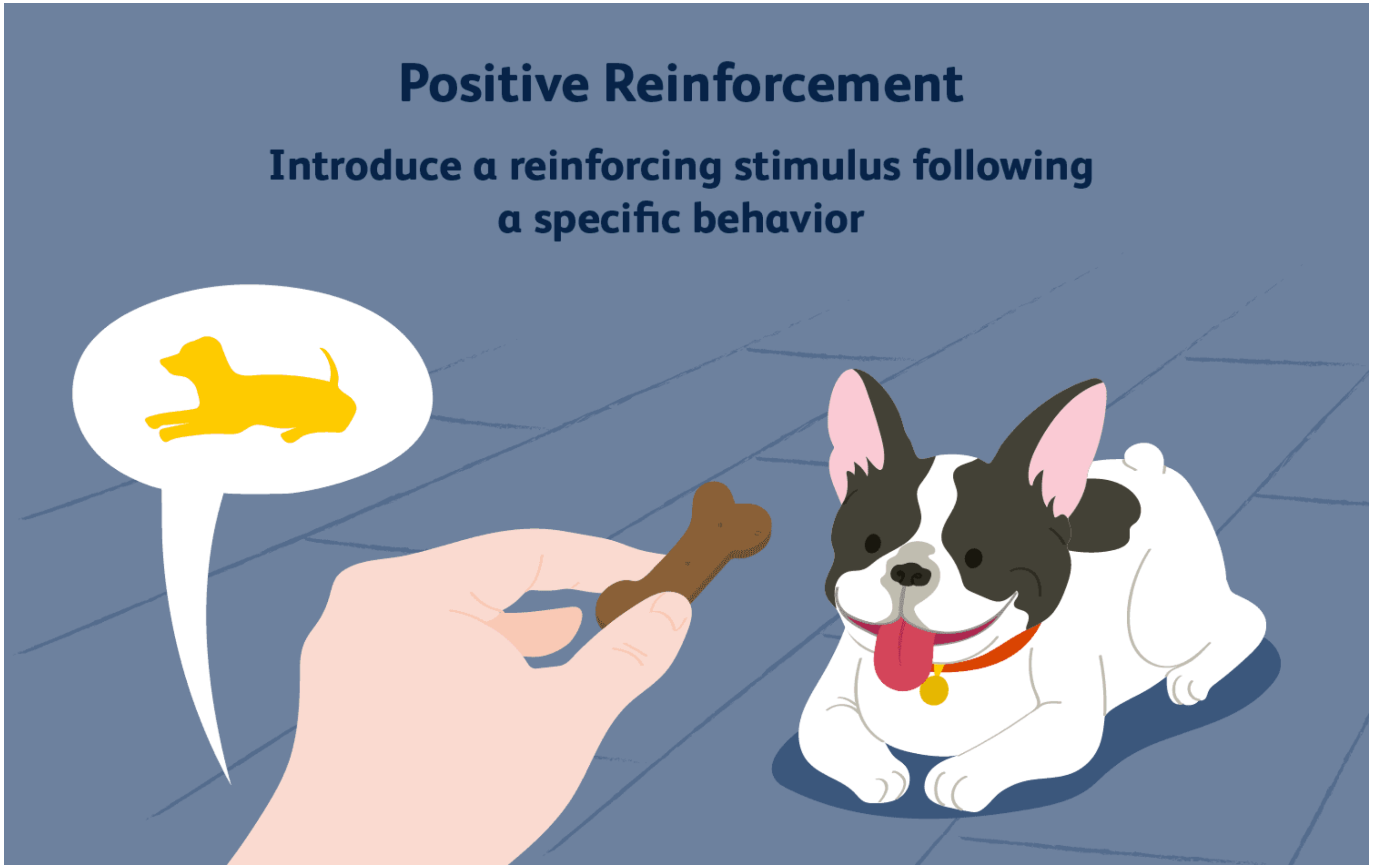
> Game

AlphaGo: Go World champion

AlphaStar: Grandmaster (99.8%)

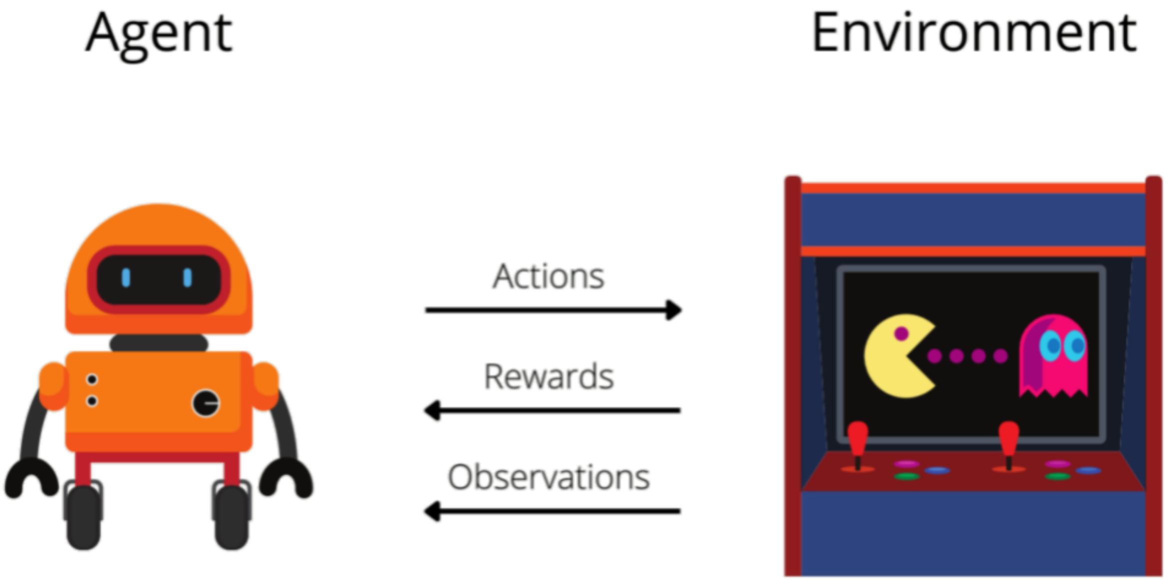
RL = trial-and-error learning

- Reinforcement in educational psychology
  - Big advantage: AI can learn autonomously



RL = trial-and-error learning

> Learning a policy that maximizes rewards by interacting with the environment



- Each action results in an immediate reward.
- We want to choose actions that maximize our immediate reward in expectation.

Multi-armed bandits

> bandit = 

One-armed bandit: slot machine

Multi-armed bandit: multiple slot machines

Multi-armed bandits

- The state does not change!
  - We don’t move. We use the same slot machines.
- We have N slot machines and select one to pull.
  - We have N possible actions.
- We will get an immediate reward from the pulled slot machine.
  - Each slot machine gives a random reward.
  - The reward probability of each machine is fixed but unknown.

- Objective: maximize cumulative rewards

How to earn money?

> Strategy 1: pull each once, exploit the best

Reward: 100₩ 0₩ 200₩ 30₩

> Next, we only pull the third machine

How to earn money?

> Strategy 2: pull each 4 times, exploit the best

Reward:

100₩	0₩	200₩	30₩
100₩	300₩	0₩	500₩
100₩	0₩	0₩	20₩
100₩	400₩	0₩	40₩
400₩	700₩	200₩	590₩

> Next, we only pull the second machine

Achieving a balance

- Pulling the same machine several times
  = learning reward probability distribution and its mean
  = **exploration (collecting information)**

- Pulling the best machine
  = exploiting the best to earn money (given current information)
  = **exploitation (collecting reward)**

- Action-value for action a is its mean reward
  - $Q_t^*(a) = \mathbb{E}[R_{t+1} | A_t = a]$, action-value estimate: $Q_t(a) \approx Q_t^*(a)$
  - $A_t^* = \arg max\, Q_t(a)$
  - If $A_t = A_t^*$: exploiting
  - If $A_t \neq A_t^*$: exploring
- We need to do both

Exploration dilemma

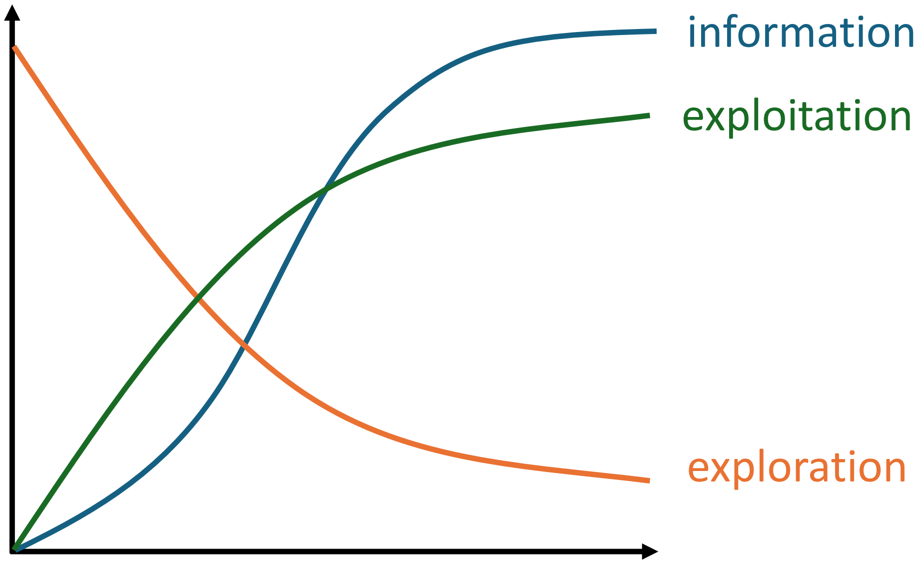
- Exploration vs Exploitation dilemma
  - The best long-term strategy may involve short-term sacrifices
  - This is not a problem unique to RL; it is a fundamental issue in the decision making of any intelligent agent.

- Restaurant selection
  - exploitation: go to your favorite restaurant
  - exploration: try a new restaurant

- Studying
  - exploitation: solve example problems
  - exploration: read additional materials

Exploration dilemma

> $\epsilon - greedy$ algorithm



Markov decision process

> Multi-armed bandit
  - $\tau: (A_t, R_{t+1}, A_{t+1}, R_{t+2}, A_{t+2}, R_{t+3}, \dots)$

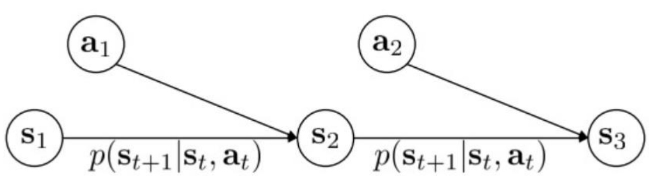
> Markov decision process
  - $\tau: (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, A_{t+2}, R_{t+3}, \dots)$

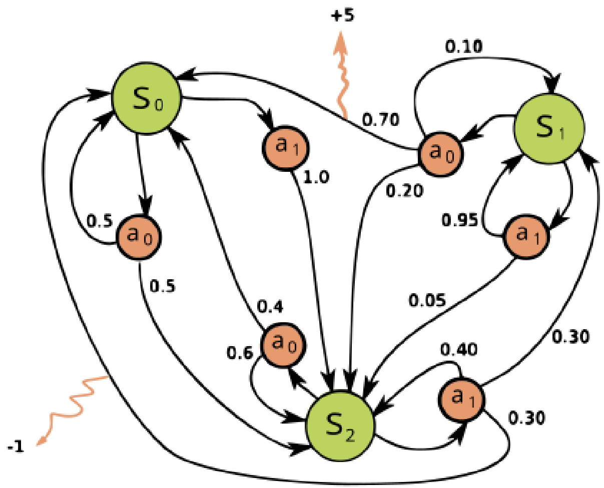
> Markov property
  - $p[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_0, S_1, A_1, R_1, \dots S_t, A_t]$
    $= p[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]$
  - Only the present determines the future; we can ignore the history.

Markov decision process

- Finite Markov decision process is a tuple (S, A, T, r, γ)
  - S is a finite set of states $s \in S$
  - A is a finite set of actions $a \in A$
  - T is one step transition/dynamics function $p(s'|s, a)$
  - $r(s, a, s')$ is a reward function
  - γ is a discount factor $(0 \leq \gamma \leq 1)$

- $\pi(a|s)$: a policy is a distribution over actions given states



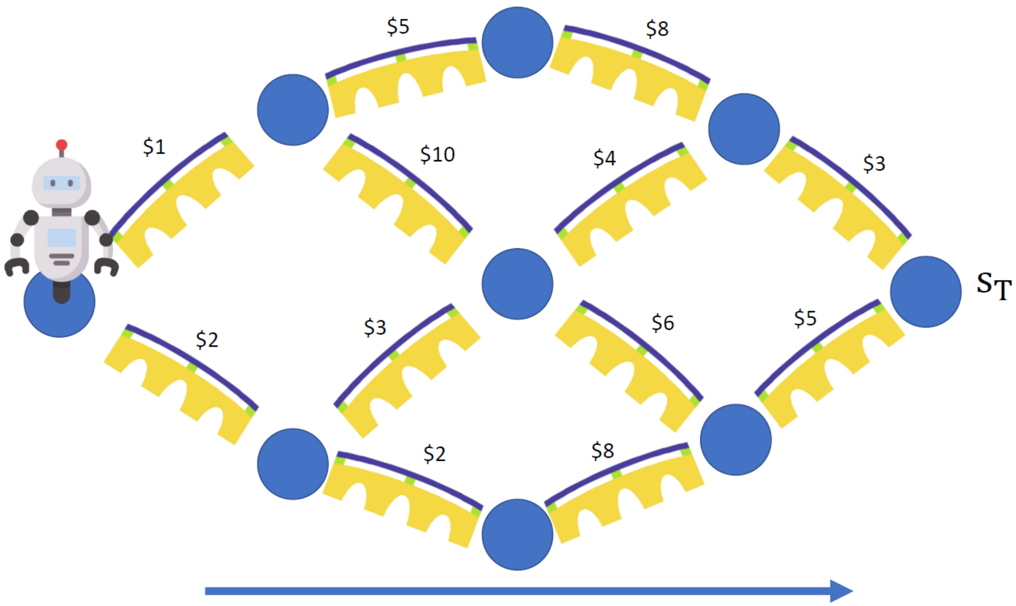


Markov decision process

> Maximize your sum of future rewards (cumulative reward)
  - $G = R(\tau) = R_1 + R_2 + R_3 + R_4 \ldots$

  - Future rewards may less important
  - $G = R(\tau) = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \cdots$

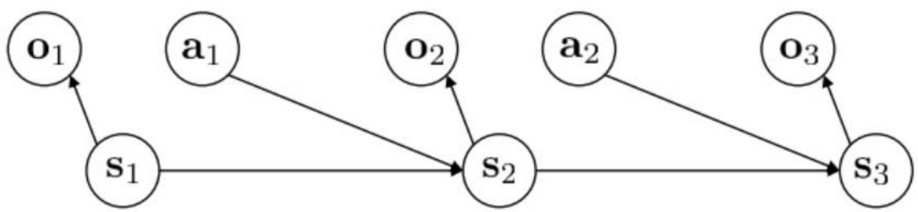
> The robot collects toll on every bridge



POMDP

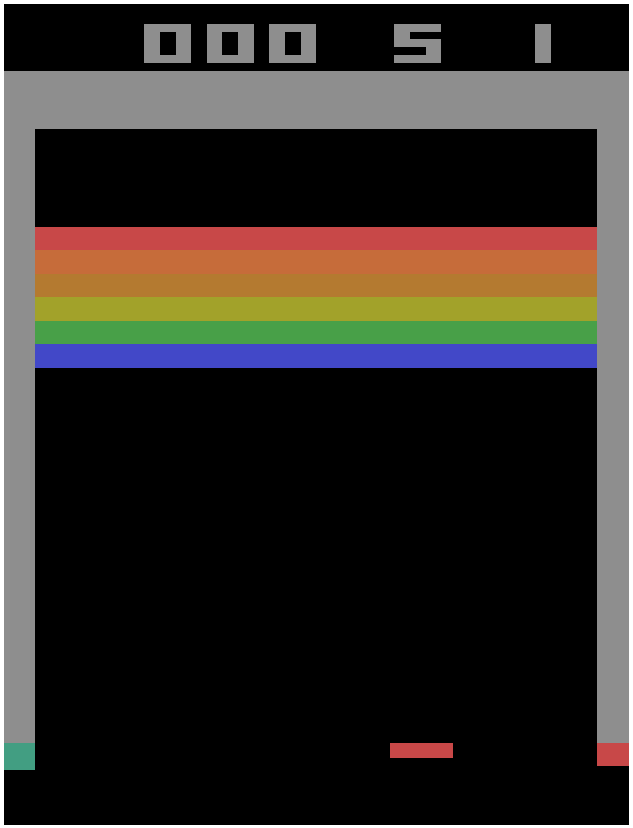
> Partially Observable MDP
  - Finite POMDP is a tuple $(S, A, O, T, h, r, \gamma)$
  - h is the observation model $p(o|s)$

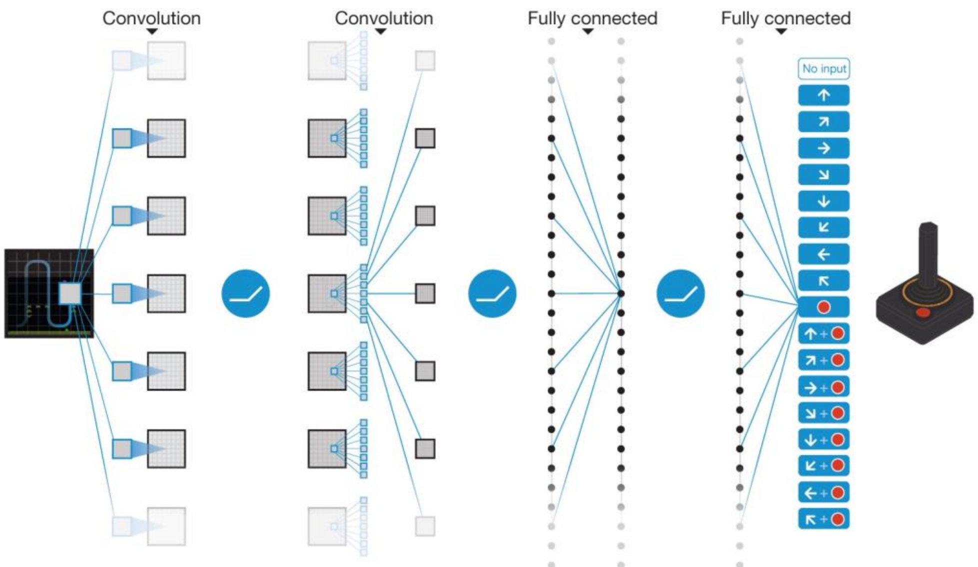
> Learn a policy $\pi(a|o)$



POMDP

> Playing Atari with DQN

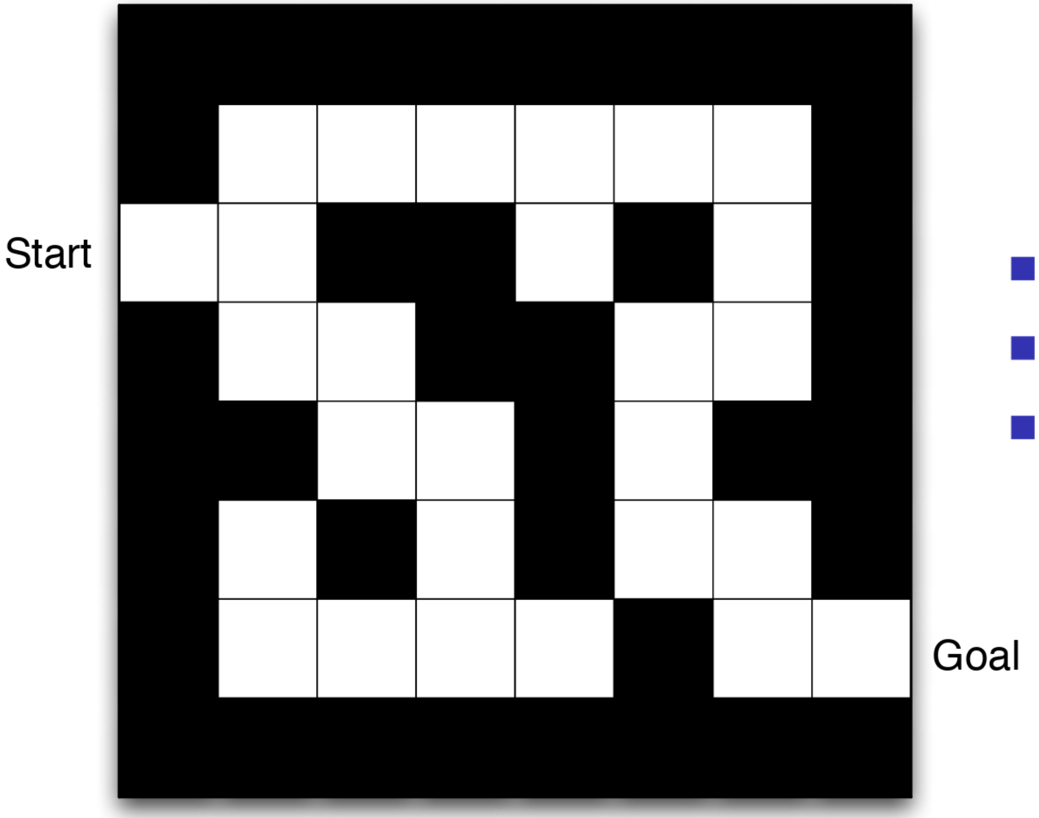




Major components of an RL agent

> An RL agent may include those components:
  - policy: agent's behavior function
    - deterministic policy $A_t = \pi(s)$
    - stochastic policy $A_t = \pi(a|s) = p(A_t = a|S_t = s)$
  - value function: how good is each state and/or action
  - model: agent's representation of the environment
    - predict what the environments will do next
    - next state: $p(S_{t+1} = s'|S_t = s, A_t = a)$
    - next reward: $p(R_{t+1}|S_t = s, A_t = a)$

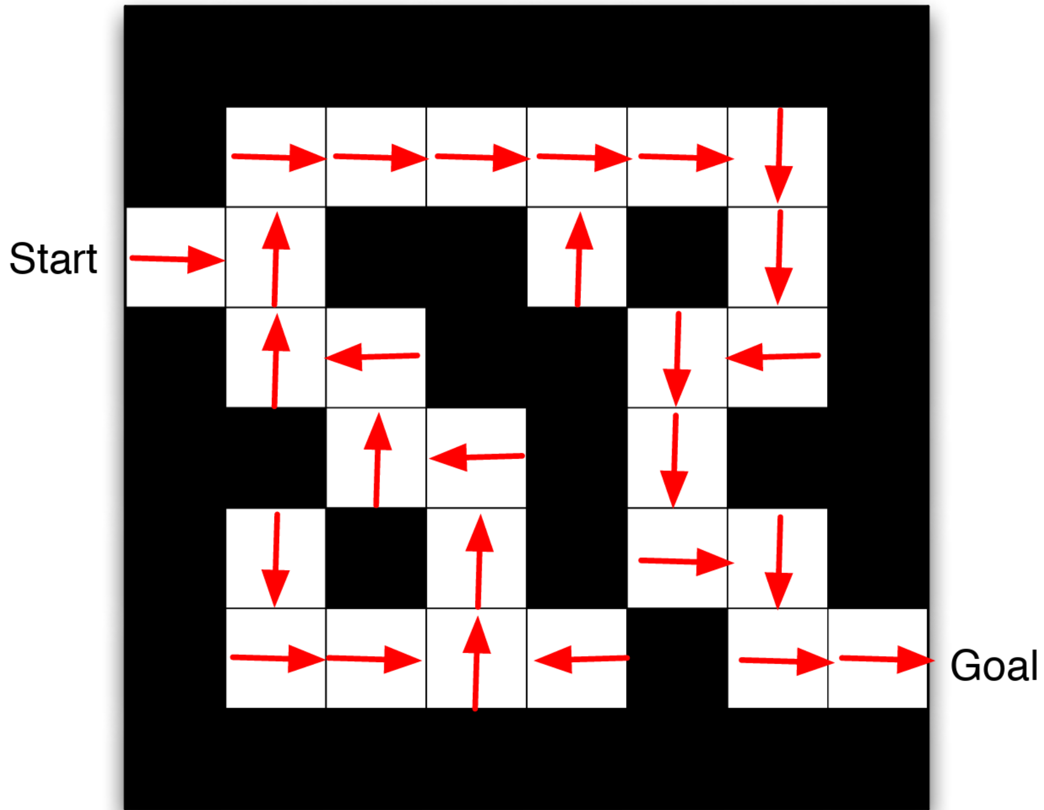
Maze example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

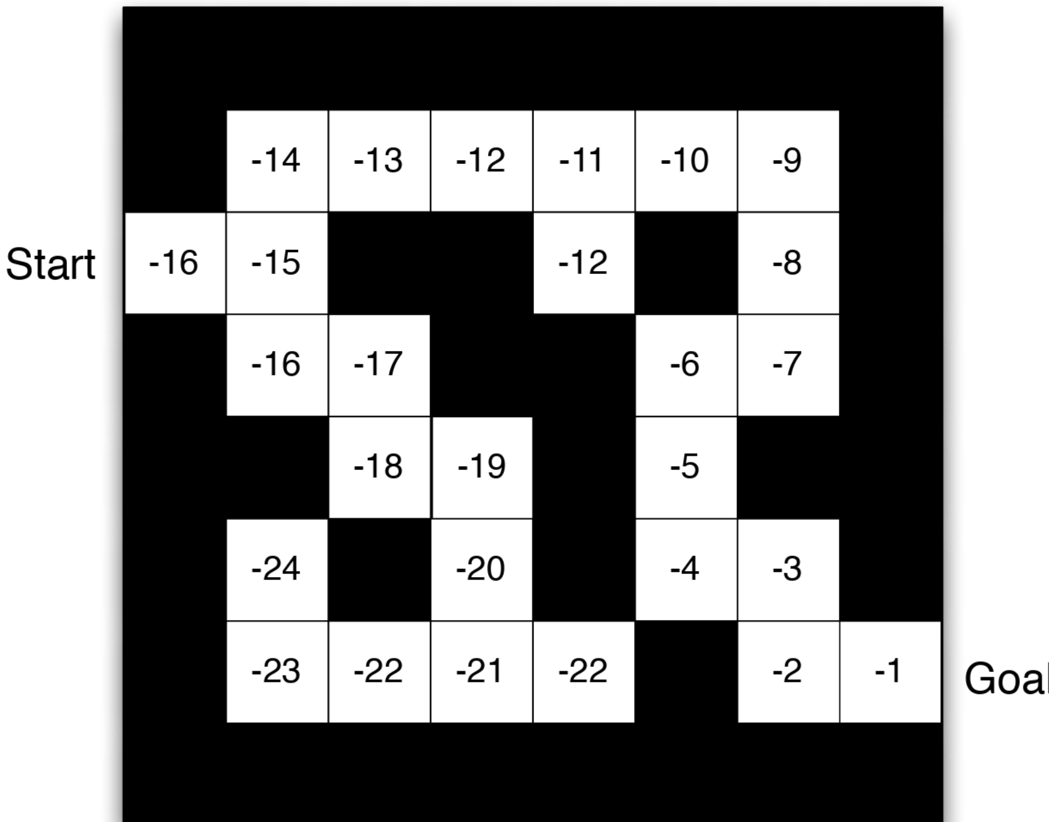
Maze example

- Policy



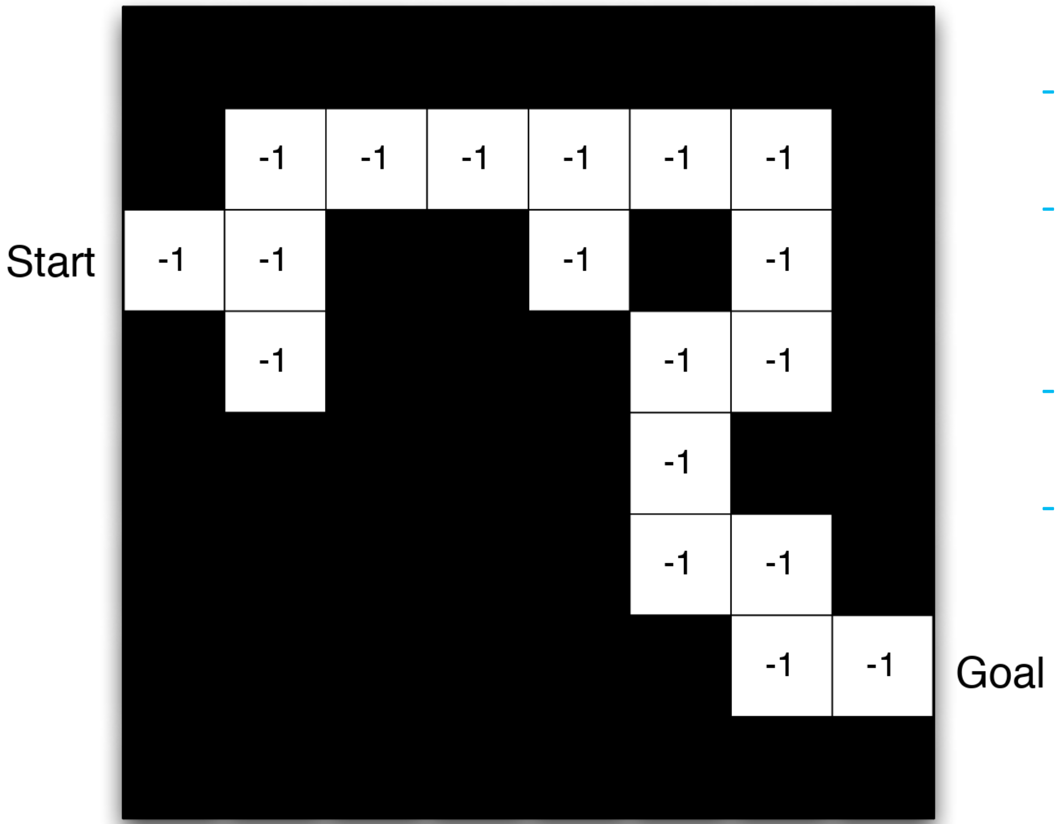
Maze example

> Value function



Maze example

> Model

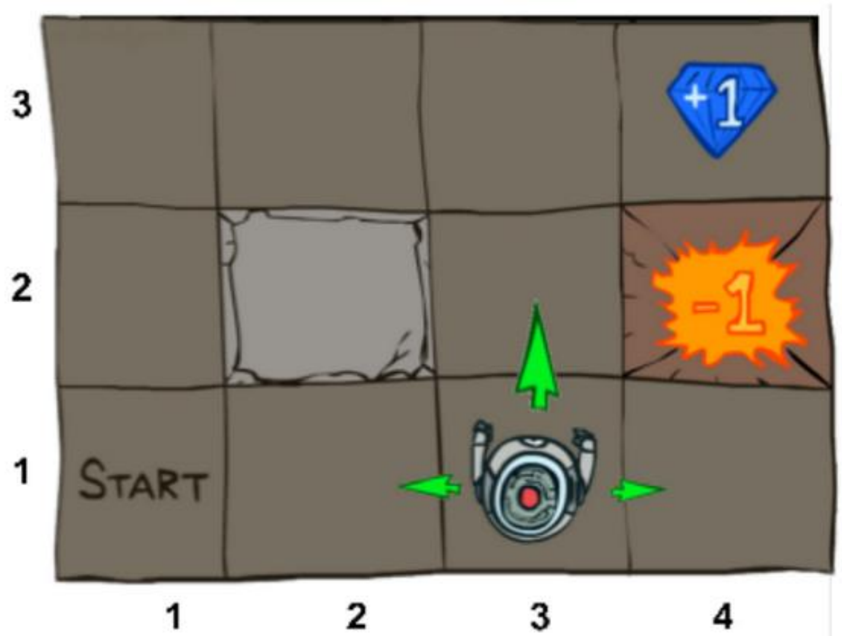


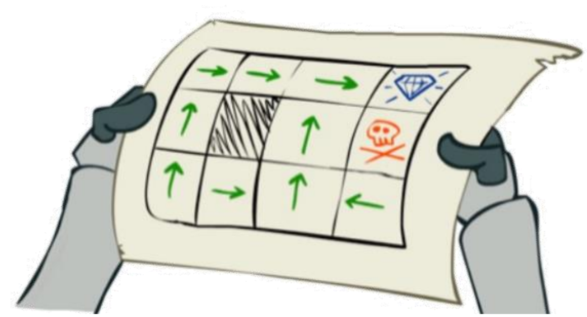
- Dynamics: how actions change the state
- Rewards: how much reward from each state

- Grid layout represents transition model
- Numbers represent immediate reward

The goal of RL

- Finite horizon case: T is finite
- Infinite horizon case: T = ∞
- The cumulative reward is often discounted: $G_t = \sum_t \gamma^t r(s_t, a_t)$
- Goal: find a policy to maximize the cumulative reward





Value functions

> State-value function (V): the expected return starting from state s, and then following policy π

- $V_\pi = \mathbb{E}_\pi[G_t | S_t = s]$
- optimal: $V^*(s) = \max_\pi V_\pi(s)$
  (maximum value function over all policies)
  (the expected return when acting optimally)

> Action-value function (Q): the expected return starting from state s, taking action a, and then following policy π

- $Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$
- optimal: $Q^*(s, a) = max_\pi Q_\pi(s, a)$

> Value functions capture the knowledge of the agent regarding how good is each state for the goal the agent is trying to achieve.

Value functions

> Once we obtain the optimal value function, we can find an optimal policy

> Maximizing over $Q^*(s,a)$

- $\pi^*(a|s) = \begin{cases} 1 & if\ a = argmax_a\, Q^*(s,a) \\ 0 & otherwise \end{cases}$

> Maximizing over $V^*(s)$ with the model dynamics

- $\pi^*(a|s) = \begin{cases} 1 & if\ a = argmax_a \sum p(s',r|s,a)(r+\gamma V^*(s')) \\ 0 & otherwise \end{cases}$
- We need the dynamics to do one step lookahead to choose the optimal a

Roadmap

- Computing state and state-action value functions by solving linear systems of equations

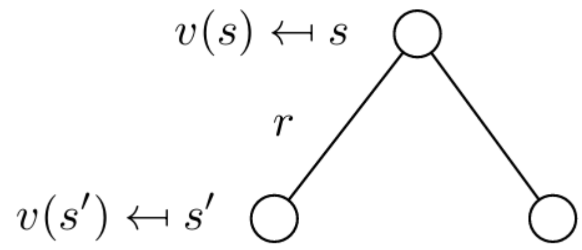
- The matrix inversion is too costly
  -> iterative estimation is required (Bellman backup operation)

- We cannot visit every state
  -> selective backups on state-actions that the agent visits

- We may not know dynamics
  -> Monte Carlo learning or TD learning

Recursive relationships for returns

> $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$
> $\quad = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots)$
> $\quad = R_{t+1} + \gamma G_{t+1}$



> By taking expectations

- $\mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s]$
- $V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V_\pi(s') | S_t = s]$
  $\quad = \sum_{s'} p(s', r | s)[r + \gamma V_\pi(s')]$ (Bellman expectation equation)

- For all states, $V_\pi = R_\pi + \gamma P_\pi V_\pi$
  where v is a column vector with one entry per state

$$\begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} V(1) \\ \vdots \\ V(n) \end{bmatrix}$$

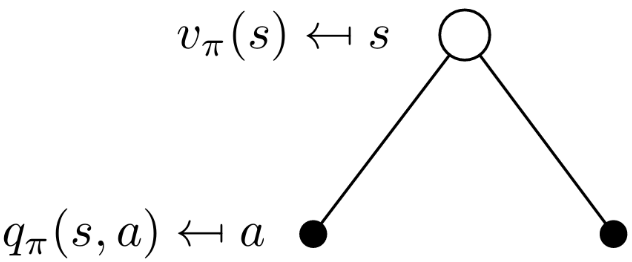
Solving the Bellman expectation equation

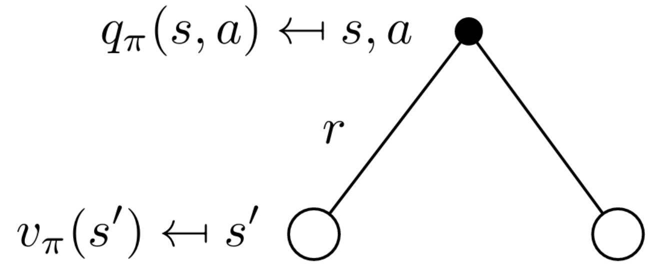
- The Bellman expectation equation is a linear equation
- It can be solved directly:
  - $V_\pi = R_\pi + \gamma P_\pi V_\pi$
    $(I - \gamma P_\pi)V_\pi = R_\pi$
    $V_\pi = (I - \gamma P_\pi)^{-1} R_\pi$
- Computational complexity is $O(n^3)$ for n states
- There are many iterative methods for large state system
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Relating state and state-action value functions

> The action-value function can similarly be decomposed

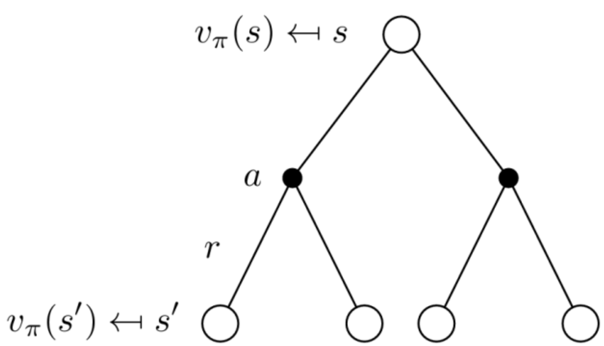
$$Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma Q_\pi(s',a')|S_t = s, A_t = a]$$





$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a)$$

$$Q_\pi(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma V_\pi(s')]$$



$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s',r|s,a)[r + \gamma V_\pi(s')]$$

$$Q_\pi(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma \sum_a \pi(a'|s') Q_\pi(s',a')]$$

Bellman optimality equations

- For the Bellman expectation equations, we sum over all the possibilities.
- Now, we choose only the best action.
  - $V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a) \quad \rightarrow \quad V^* = \max_a Q^*(s,a)$
  - $Q_\pi(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma V_\pi(s')]$

    $\rightarrow Q^*(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma V^*(s')]$
  - $V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s',r|s,a)[r + \gamma V_\pi(s')]$

    $\rightarrow V^*(s) = \max_a \sum_{s'} p(s',r|s,a)[r + \gamma V^*(s')]$
  - $Q_\pi(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma \sum_a \pi(a'|s') Q_\pi(s',a')]$

    $\rightarrow Q^*(s,a) = \sum_{s'} p(s',r|s,a)[r + \gamma \max_a Q^*(s',a')]$

Solving the Bellman optimality equation

- Bellman optimality equation is nonlinear
- No closed form solution (in general)
- Many iterative solution methods
  - Using models / dynamic programming
    - Value iteration
    - Policy iteration
  - Using samples
    - Monte Carlo
    - Q-learning
    - SARSA

Extensions to MDPs

- Infinite and continuous MDP
- Continuous state and/or action spaces
  - closed form for linear quadratic model (LQR)
- Continuous time
  - requires partial differential equations
  - Hamilton-Jacobi-Bellman (HJB) equation

Planning by dynamic programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for planning in an MDP

- For prediction:
  - input: MDP and policy
  - output: value function V_π

- For control:
  - input: MDP
  - output: optimal value function V^* and optimal policy π^*

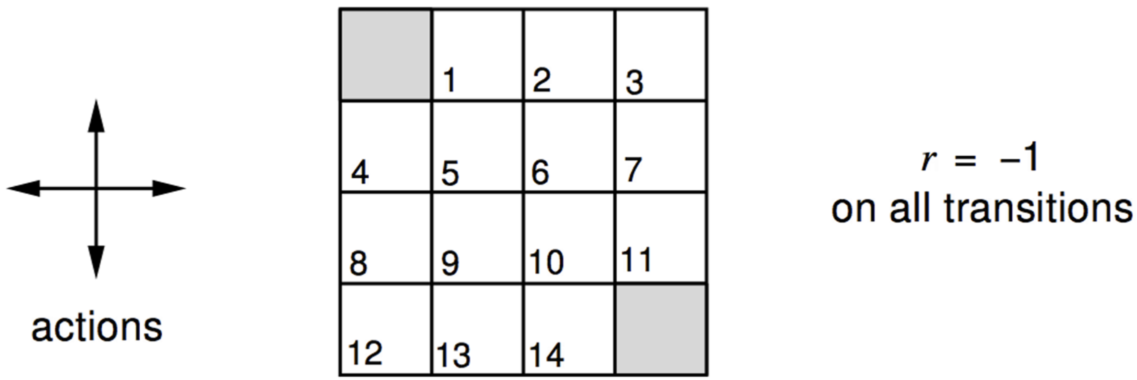
Policy Evaluation

- Problem: evaluate a given policy π (prediction)
- Solution: iterative application of Bellman expectation backup
  - $V_1 \rightarrow V_2 \rightarrow \ \dots \rightarrow V_\pi$
  - Synchronous backups – for all states

- Iterative policy evaluation
- $V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} p(s', r|s, a)[r + \gamma V_k(s')]$

Policy Evaluation

> Evaluating a random policy in the small GridWorld



- undiscounted episodic MDP ($\gamma = 1$)
- one terminal state (shown twice as shaded squares)
- actions leading out of the grid leave state unchanged
- reward is -1 until the terminal state is reached
- agent follows uniform random policy
  (each action has a probability of 0.25)

Policy Evaluation

> $V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} p(s', r|s, a)[r + \gamma V_k(s')]$

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Evaluation

> Overall algorithm

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

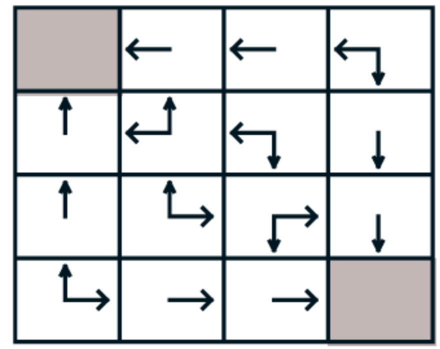
> It will converge to the fixed value function

Policy Evaluation

> Once we found the converged (optimal) value function, we can get a better (optimal) policy

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy Evaluation

- Convergence proof
  - Define the Bellman expectation backup operator
  - $T_\pi(V) = R_\pi + \gamma P_\pi V$
  - This operator is a γ-contraction; it makes value functions closer by at least γ
  - $$\begin{aligned} \|T_\pi(U) - T_\pi(V)\|_\infty &= \|R_\pi + \gamma P_\pi \mathrm{U} - (R_\pi + \gamma P_\pi V)\|_\infty \\ &= \|\gamma P_\pi (U - V)\|_\infty \\ &\leq \gamma \|P_\pi\|_\infty \|U - V\|_\infty \\ &= \gamma \|U - V\|_\infty \end{aligned}$$
  - T_π converges to a unique fixed point at a linear convergence rate γ

Policy iteration

- Now, we want to move to the control problem
- Policy iteration
  - Evaluate the policy
  - Improve the policy by acting greedily with respect to V_π

$$\pi' = greedy(V_\pi)$$

  - It always converges to optimal policy π^*

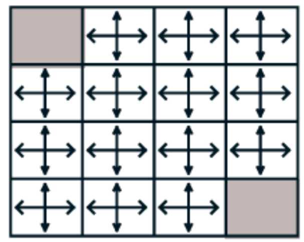
Policy iteration

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

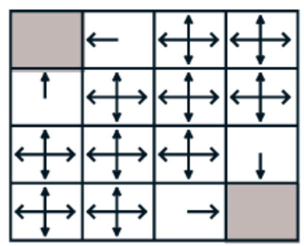
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

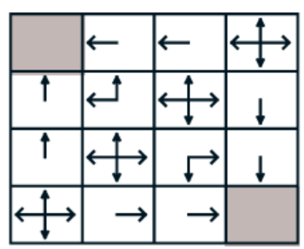
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

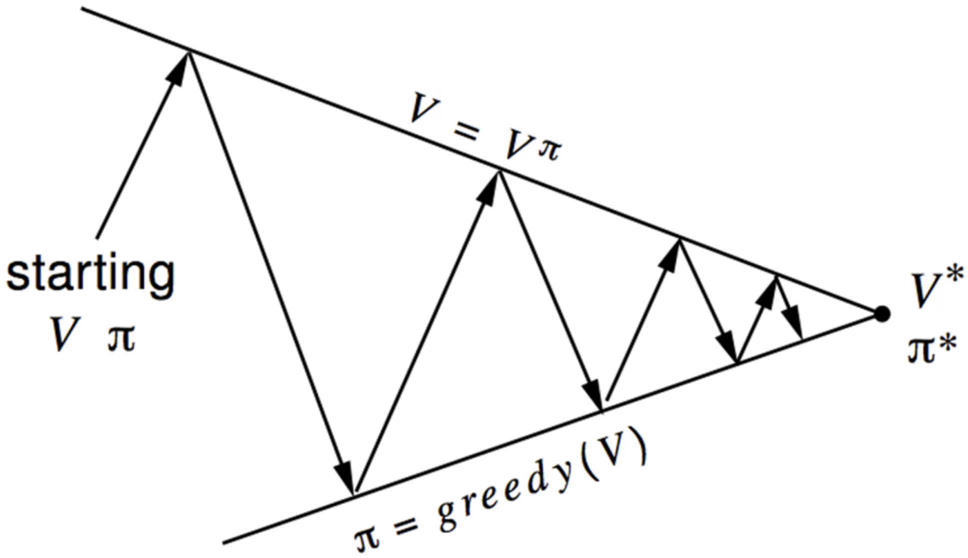
0.0	-1.0	-2.0	-2.0
-1.0	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.0
-2.0	-2.0	-1.0	0.0

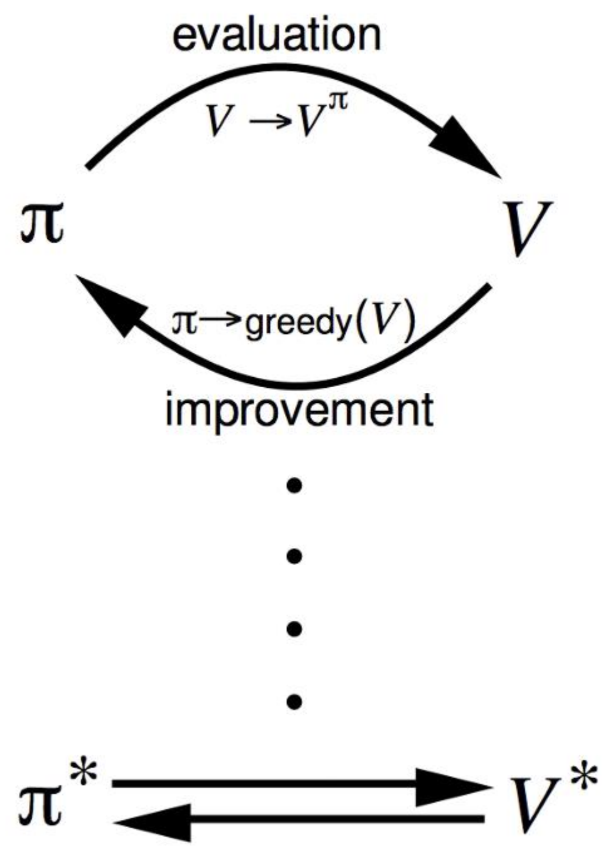


$k = 3$

0.0	-1.0	-2.0	-3.0
-1.0	-2.0	-3.0	-2.0
-2.0	-3.0	-2.0	-1.0
-3.0	-2.0	-1.0	0.0

Policy iteration





Policy iteration

- Consider a deterministic policy, $a = \pi(s)$
- Improve the policy by acting greedily, $\pi'(s) = argmax_a Q_\pi(s, a)$
- It improves the value function
  - $Q_\pi\big(s, \pi'(s)\big) = \max_a Q_\pi(s, a) \geq Q_\pi\big(s, \pi(s)\big) = V_\pi(s)$
  - $V_\pi(s) \leq Q_\pi\big(s, \pi'(s)\big) = \mathbb{E}_{\pi'}[R_{t+1} + \gamma V_\pi(s') | S_t = s]$

$$
\begin{aligned}
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma Q_\pi(s', \pi'(s')) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma V_\pi(s'')] | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 Q_\pi(s', \pi'(s'') | S_t = s] \\
&\quad \ldots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma R_{t+3} + \cdots | S_t = s] = V_{\pi'}(s)
\end{aligned}
$$

Policy iteration

- If improvements stop, $Q_\pi\big(s, \pi'(s)\big) = V_\pi(s)$
- Then, the Bellman optimality equation has been satisfied

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

- Therefore, V and π are optimal

Generalized policy iteration

- Any interleaving of policy evaluation and policy improvement
- Evaluation and improvement do not need to be exact or complete at each step
  - partial updates, approximations, or asynchronous updates are allowed
- All RL methods are a form of GPI

- Policy iteration: full evaluation + full improvement
  - using Bellman expectation eqn.
- Value iteration: one-step evaluation
  - using Bellman optimality eqn.

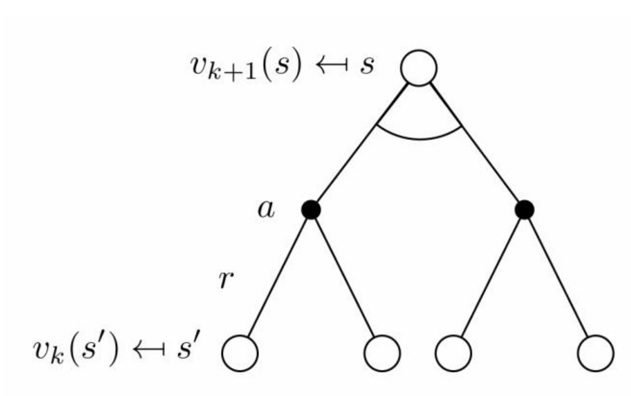
- How GPI leads optimality?
  - Over time, even with approximate steps the policy becomes progressively better

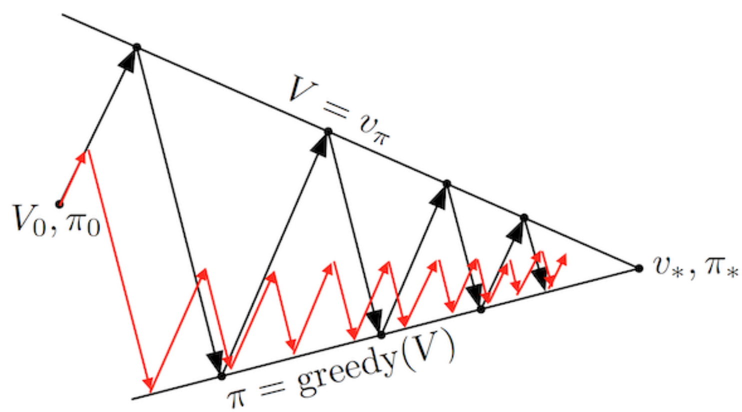
Value iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup

- Using synchronous backups
- Unlike policy iteration, there is no explicit policy

- $V_{k+1}(s) = \max_a [R(s,a) + \gamma \sum_{s'} p(s', r|s, a) V_k(s')]$





Value iteration

> $V_{k+1}(s) = \max_a [R(s,a) + \gamma \sum_{s'} p(s',r|s,a) V_k(s')]$

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

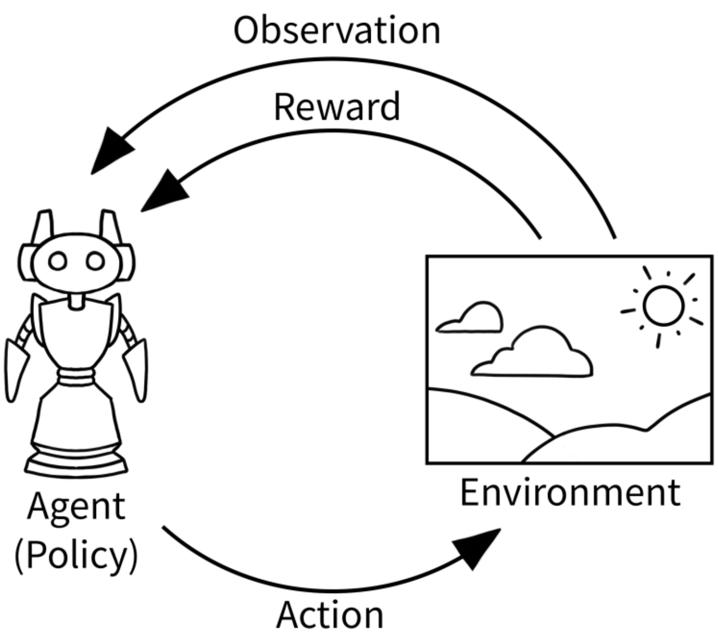
Synchronous dynamic programming algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman expectation equation	Iterative policy evaluation
Control	Bellman expectation equation +Greedy policy improvement	Policy iteration
Control	Bellman optimality equation	Value iteration

- Algorithms are based on state-value function $V_\pi(s)$ or $V^*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $Q_\pi(s,a)$ or $Q^*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

Environment for RL

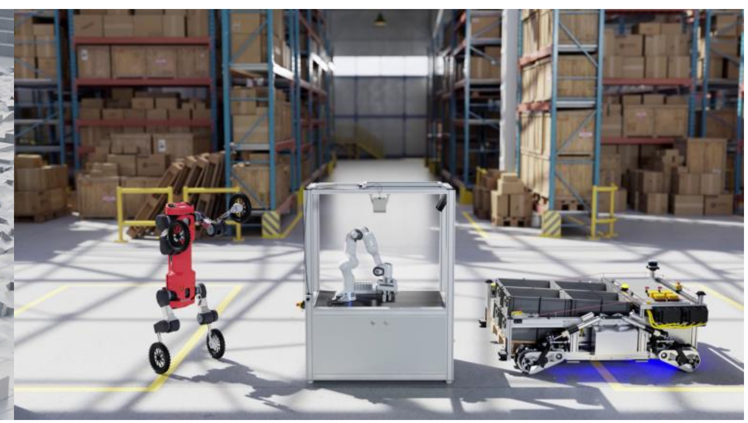
- Gymnasium
  - Provide an API for all single agent RL envs.
  - Maintained by OpenAI



env = gym.make
observation, info = env.reset
action = env.action_space.sample()
observation, reward, terminated, truncated,
info = env.step(action)

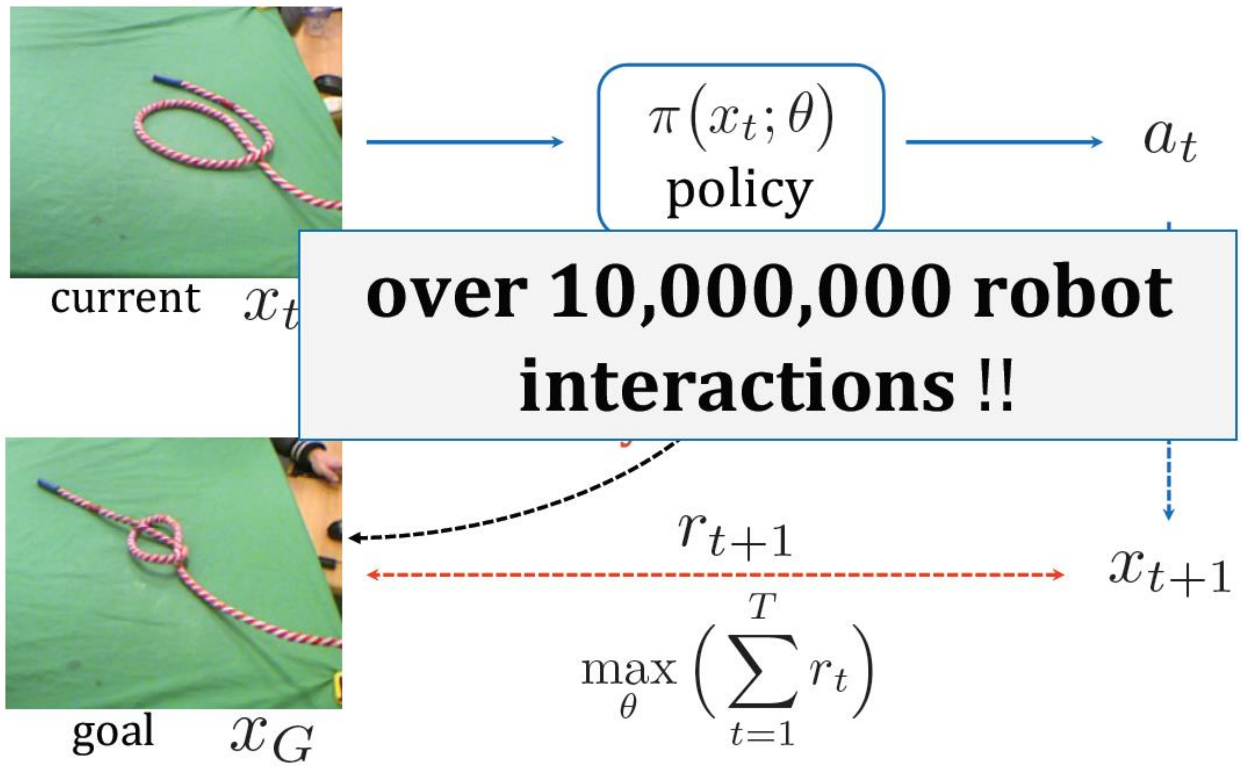
IssacSim

- Developed by NVIDIA
  - End-to-end GPU accelerated
  - Massive parallelization of thousands of environments
  - PhysX 5 physics engine, including fluid dynamics

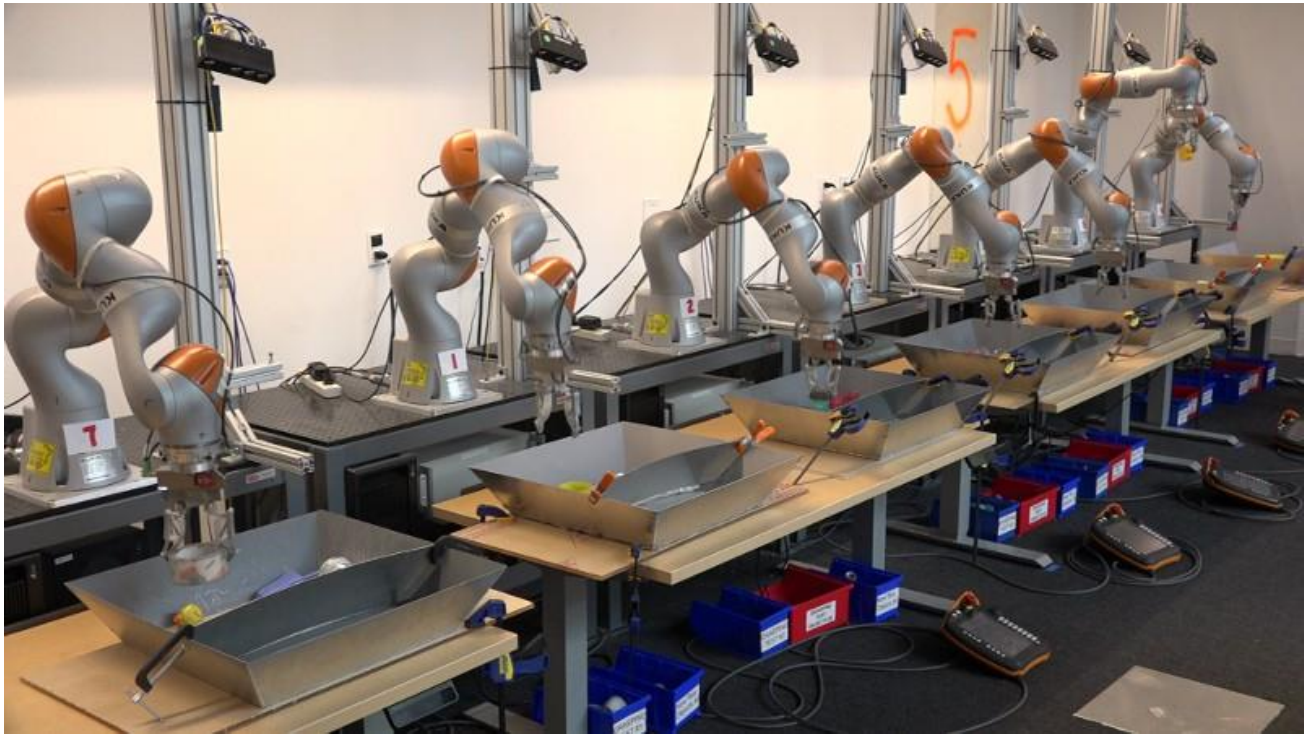
Why simulators for robot learning

> RL is very sample inefficient



Why simulators for robot learning

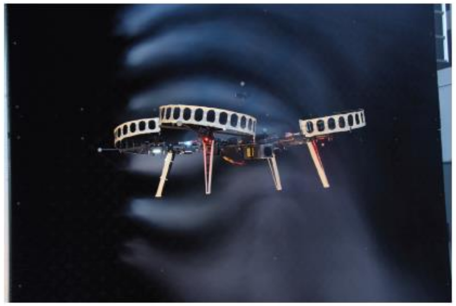
> Google QT-Opt
  - 4 months, 800 robot hours, 7 robots, 580,000 attempts



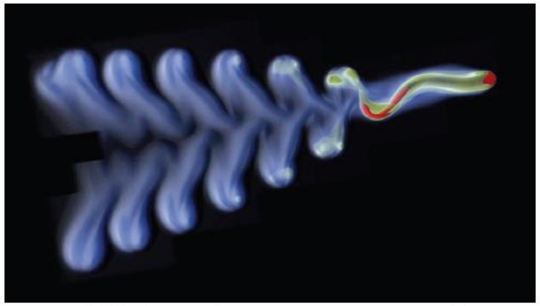
Why simulators for robot learning

> Advantages of using simulated data
  - Cheap, fast, and scalable
  - Safe
  - Labeled (we have access to ground truth
  - No physical harm or deformation (wear and tear)

> Disadvantages
  - Not exactly same with real env. (sim2real)



Aerodynamics in wind, *Neural-Fly*

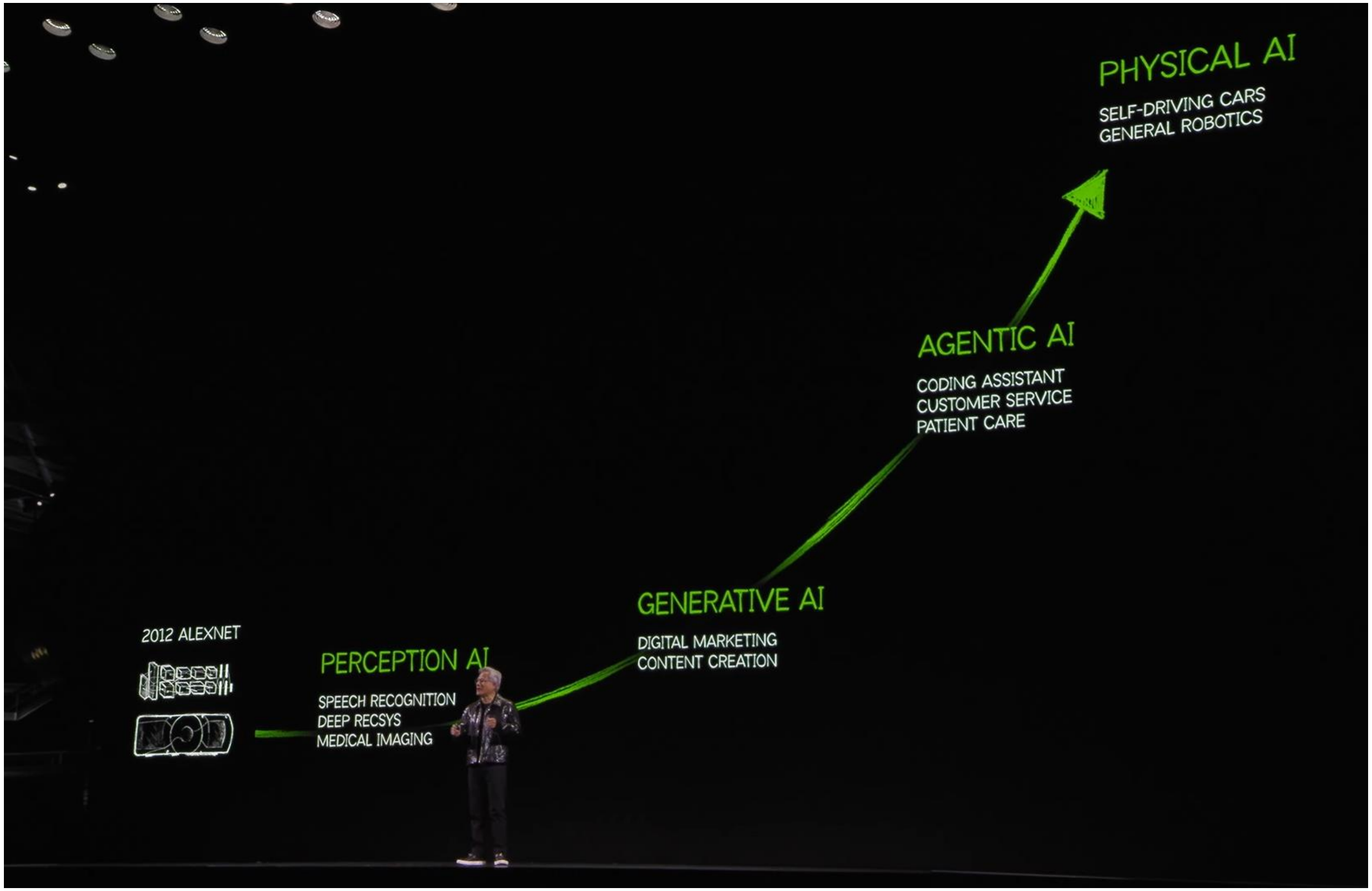


Fluid dynamics, MIT van Rees Lab

Offroad vehicle dynamics, UW Racer team

Ultimate goal

> Build general-purpose embodied intelligence by learning to make sequential decisions in the physical world.



Where are we today: non-learning method

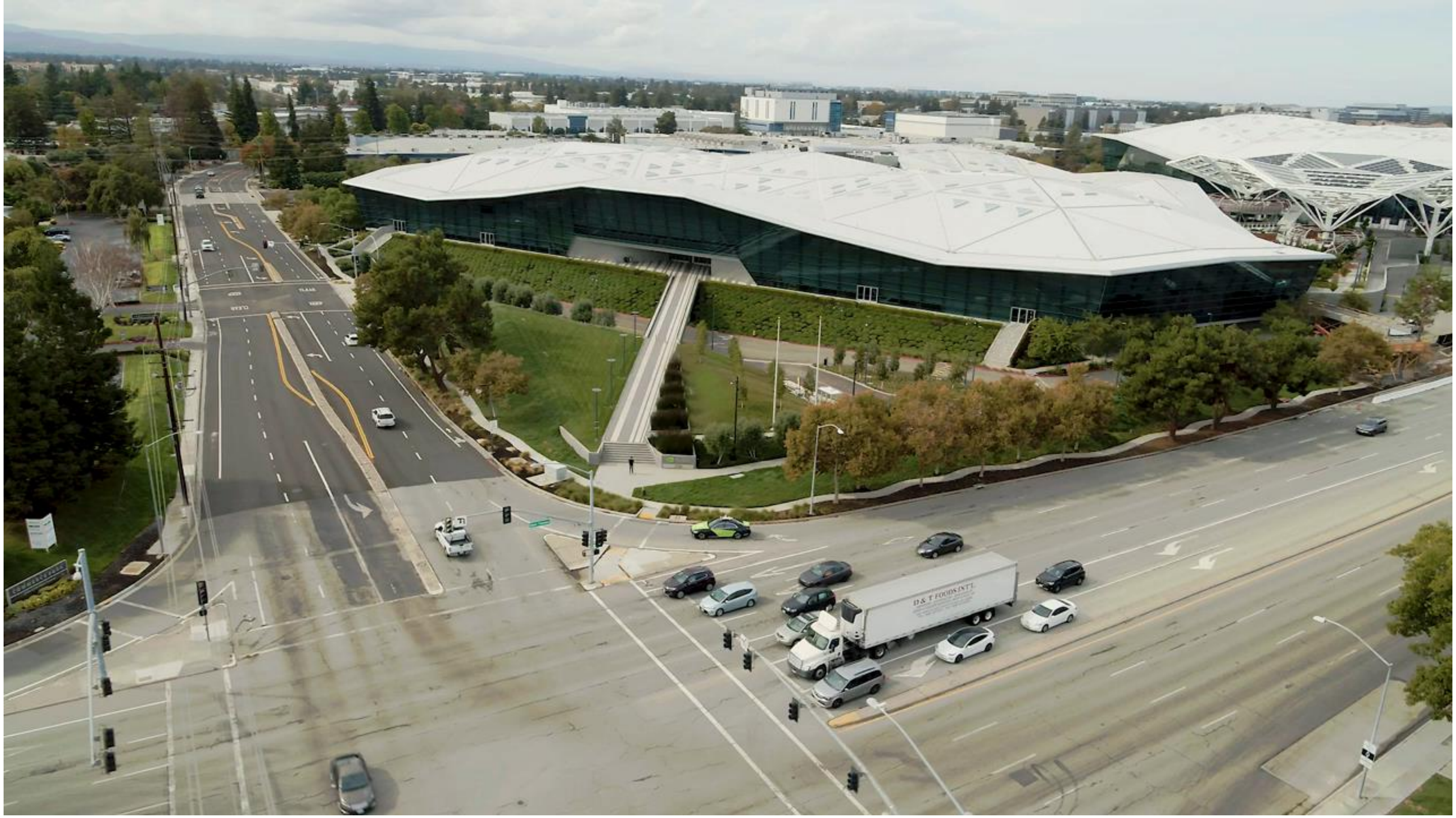
- trajectory optimization and control: optimal control + robust control

Where are we today: non-learning method

- trajectory optimization + MPC

Where are we today: learning method

> Sim2Real - NVIDIA



Where are we today: learning method

> Collect real-world data efficiently – Mobile ALOHA

Where are we today: learning method

> Control foundation model: A general navigation model (GNM)