Lecture 14:
Robot Learning

So far: Supervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Classification

Cat

So far: Self-Supervised Learning

Self-Supervised Learning

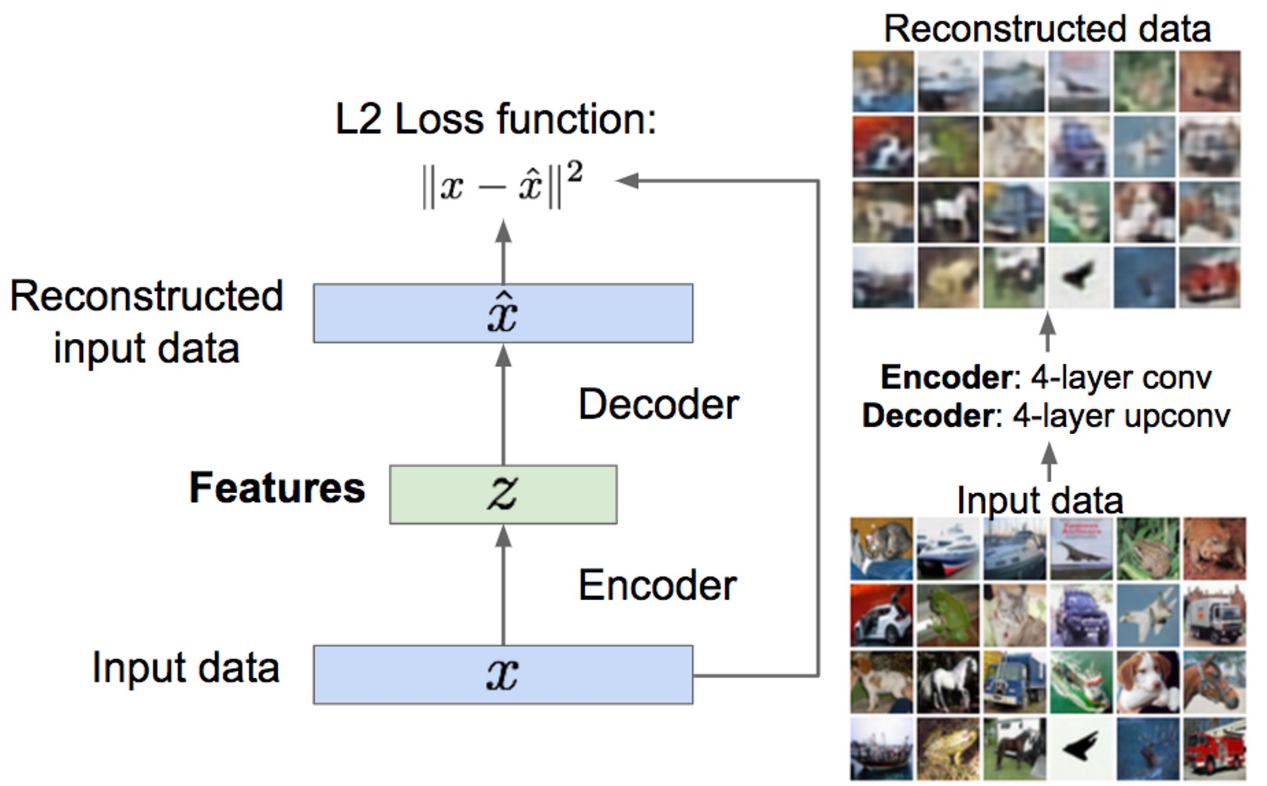
Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

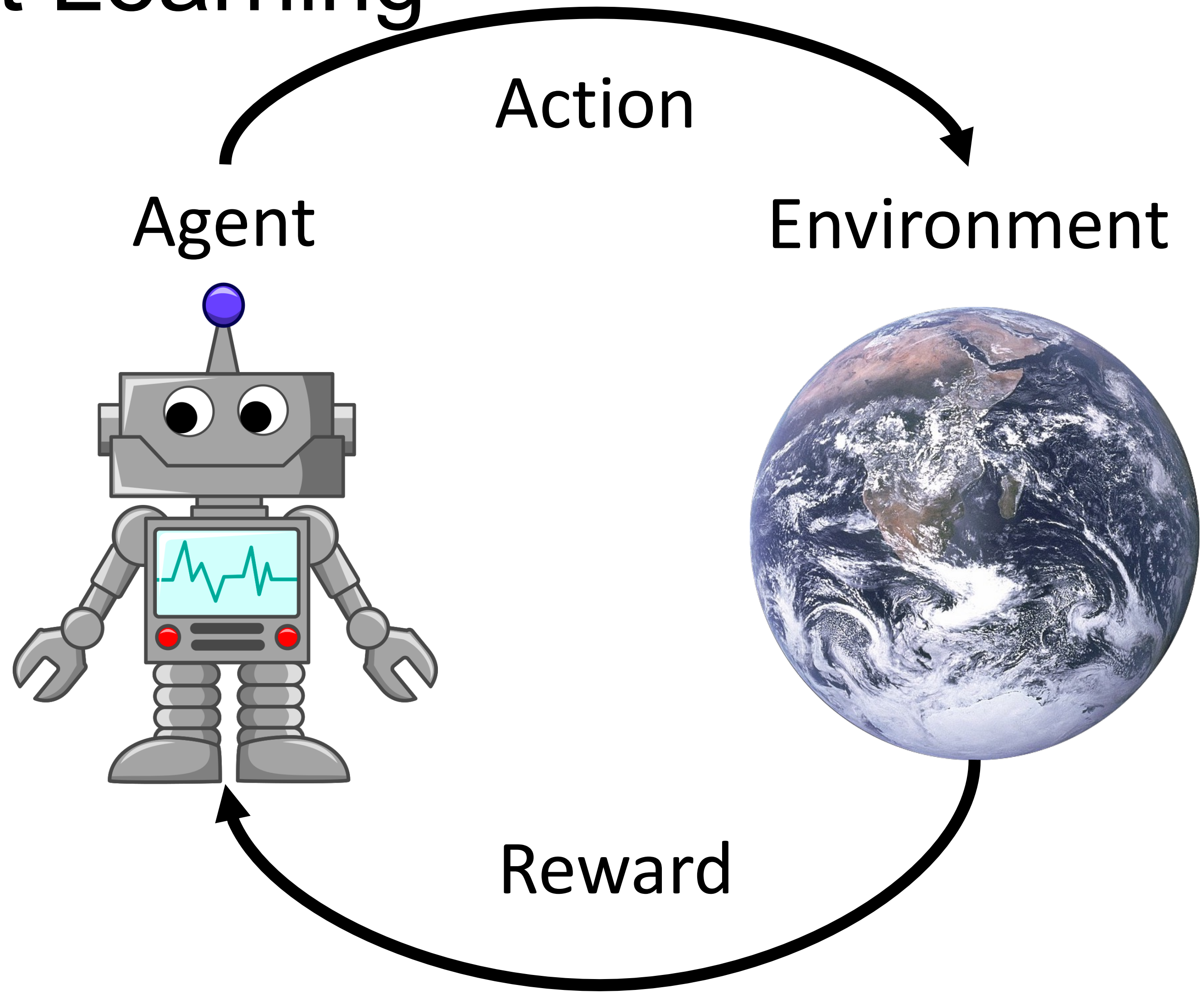
Feature Learning
(e.g. autoencoders)



Today: Reinforcement Learning

Problems where an **agent** performs **actions** in **environment**, and receives **rewards**

Goal: Learn how to take actions that maximize reward



Earth photo is in the public domain
Robot image is in the public domain

Overview

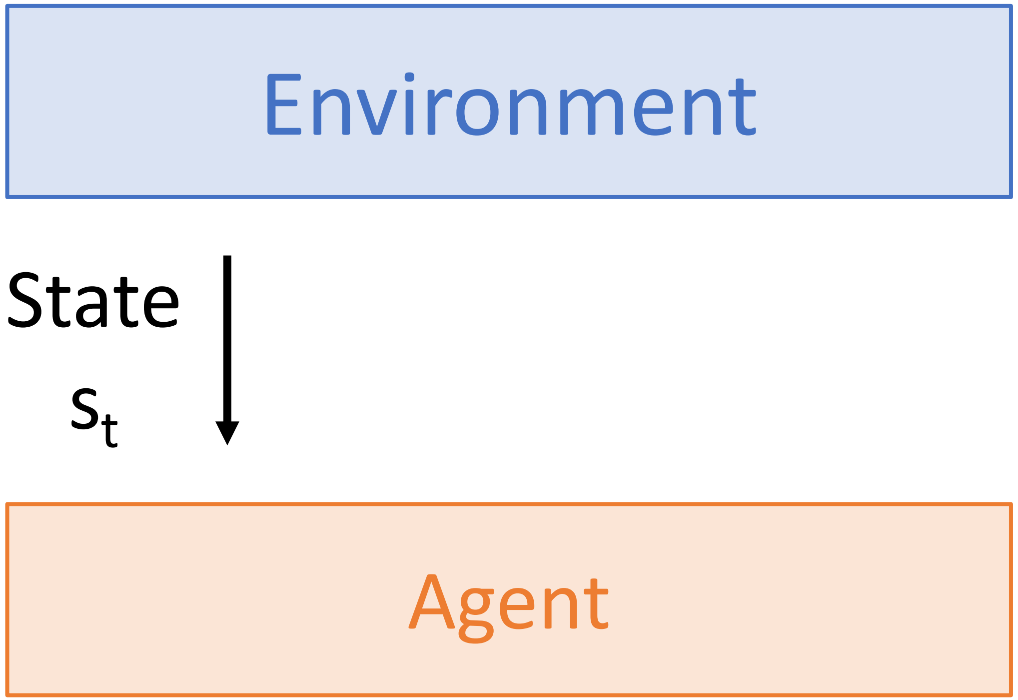
- What is reinforcement learning?
- Algorithms for reinforcement learning
  - Q-Learning
  - Policy Gradients
  - Model-based RL and planning

Reinforcement Learning

Environment

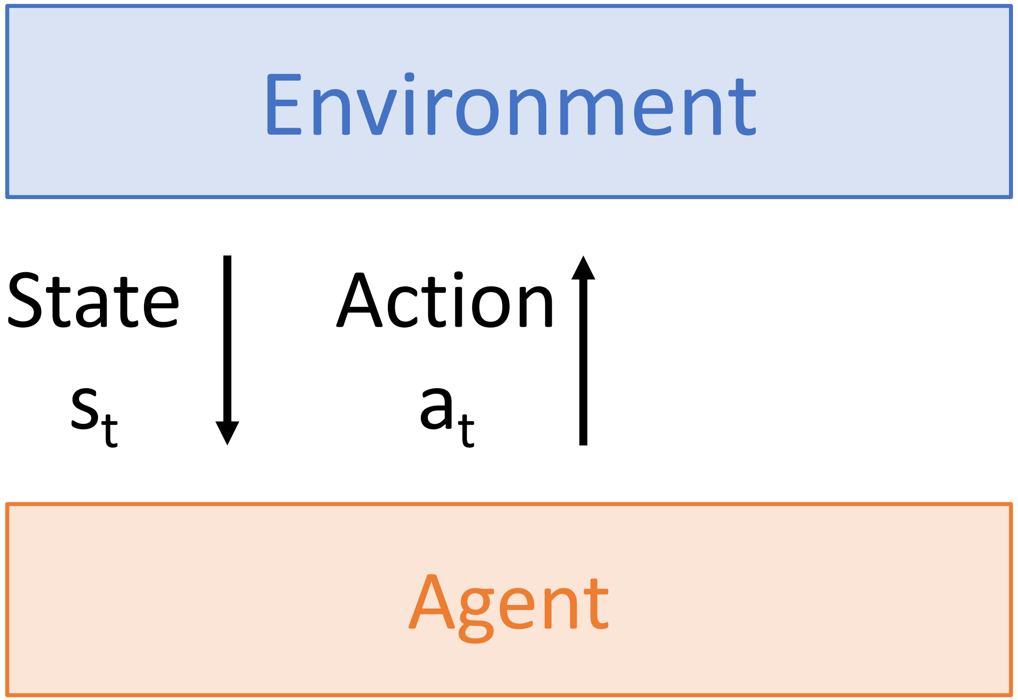
Agent

Reinforcement Learning



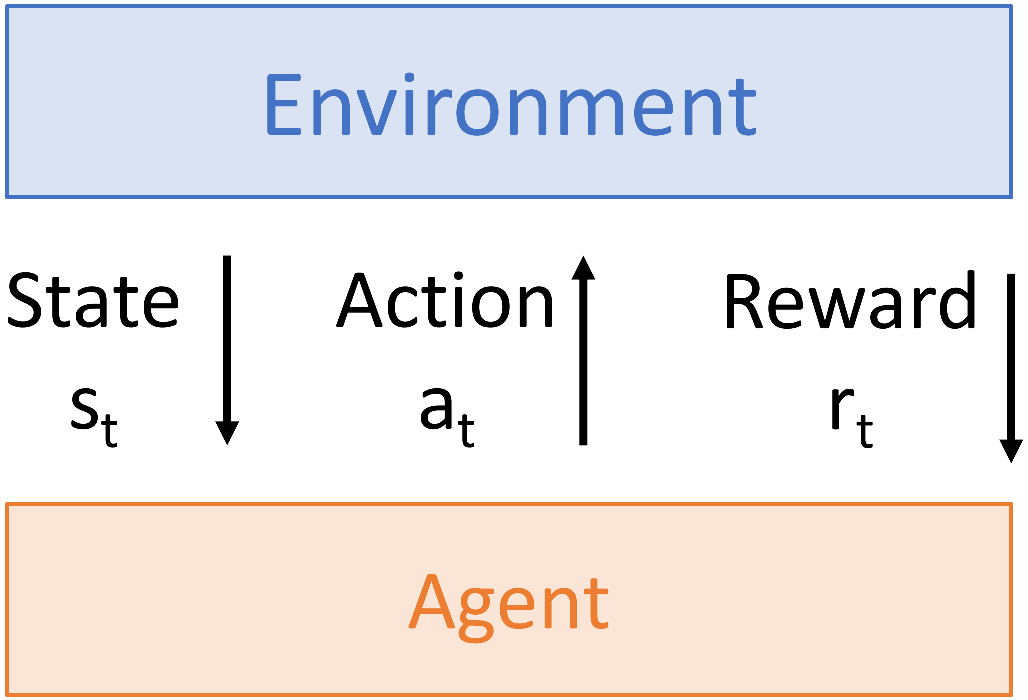
The agent sees a **state**; may be noisy or incomplete

Reinforcement Learning



The makes an **action** based on what it sees

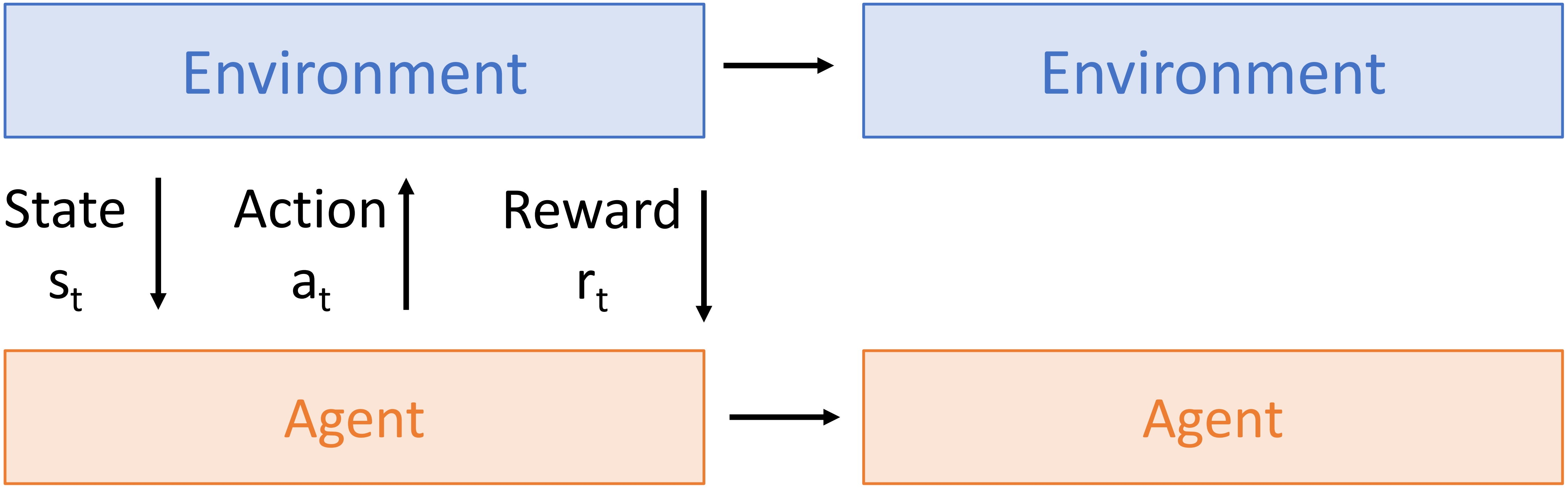
Reinforcement Learning



Reward tells the agent how well it is doing

Reinforcement Learning

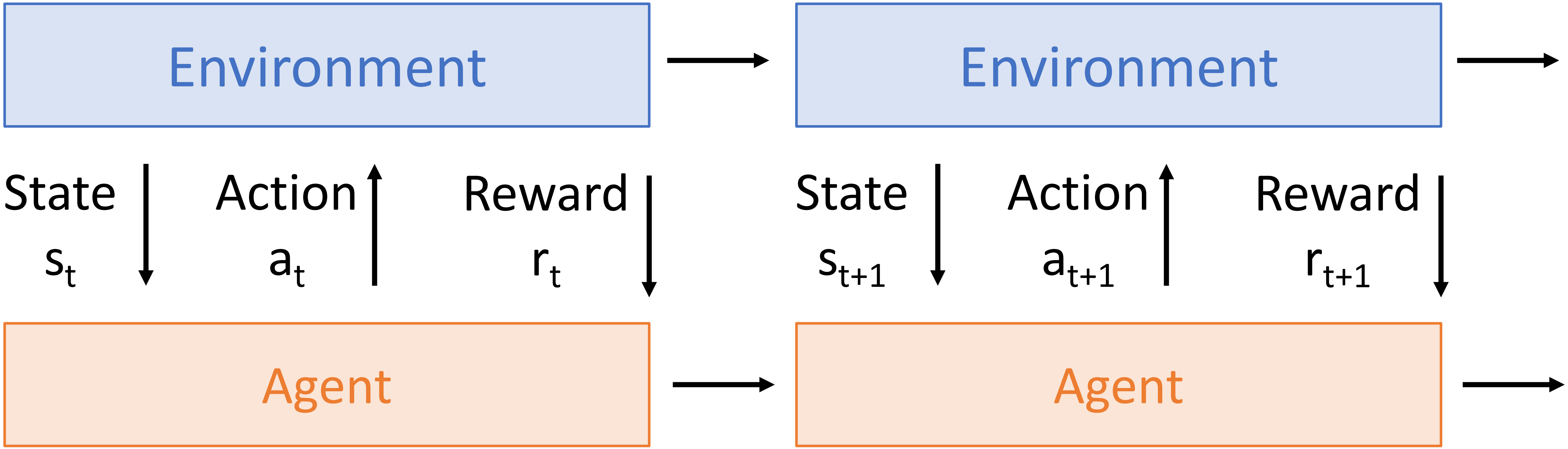
Action causes change to environment



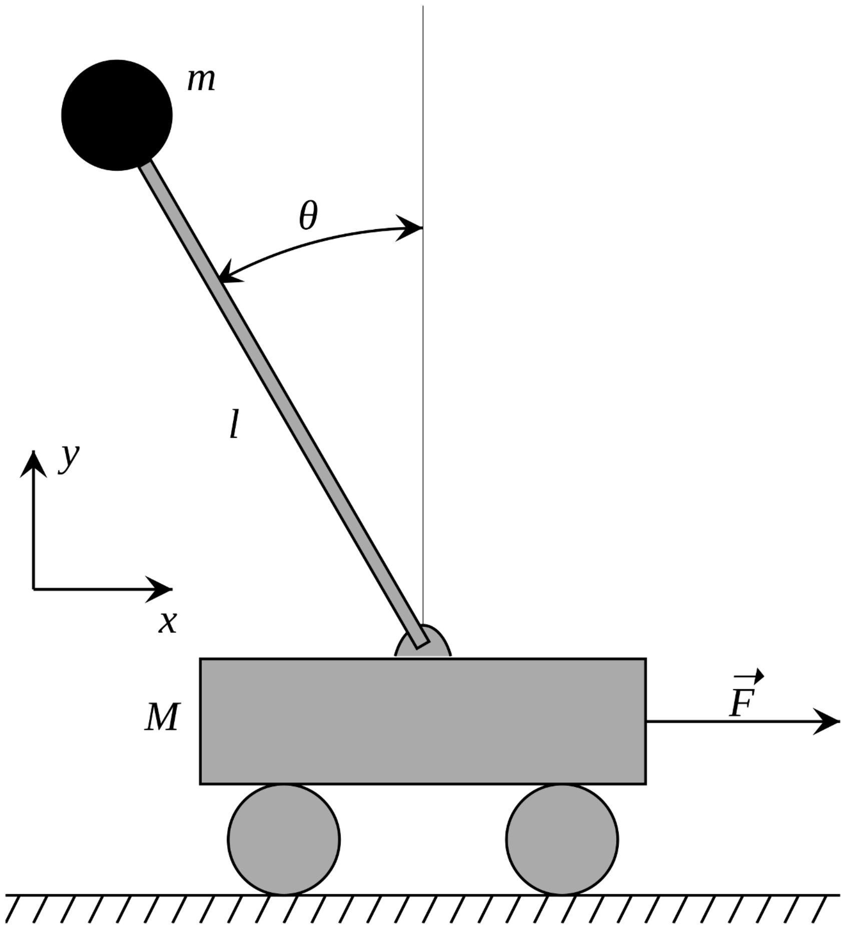
Agent learns

Reinforcement Learning

Process repeats



Example: Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity

Action: horizontal force applied on the cart

Reward: 1 at each time step if the pole is upright

Example: Robot Locomotion

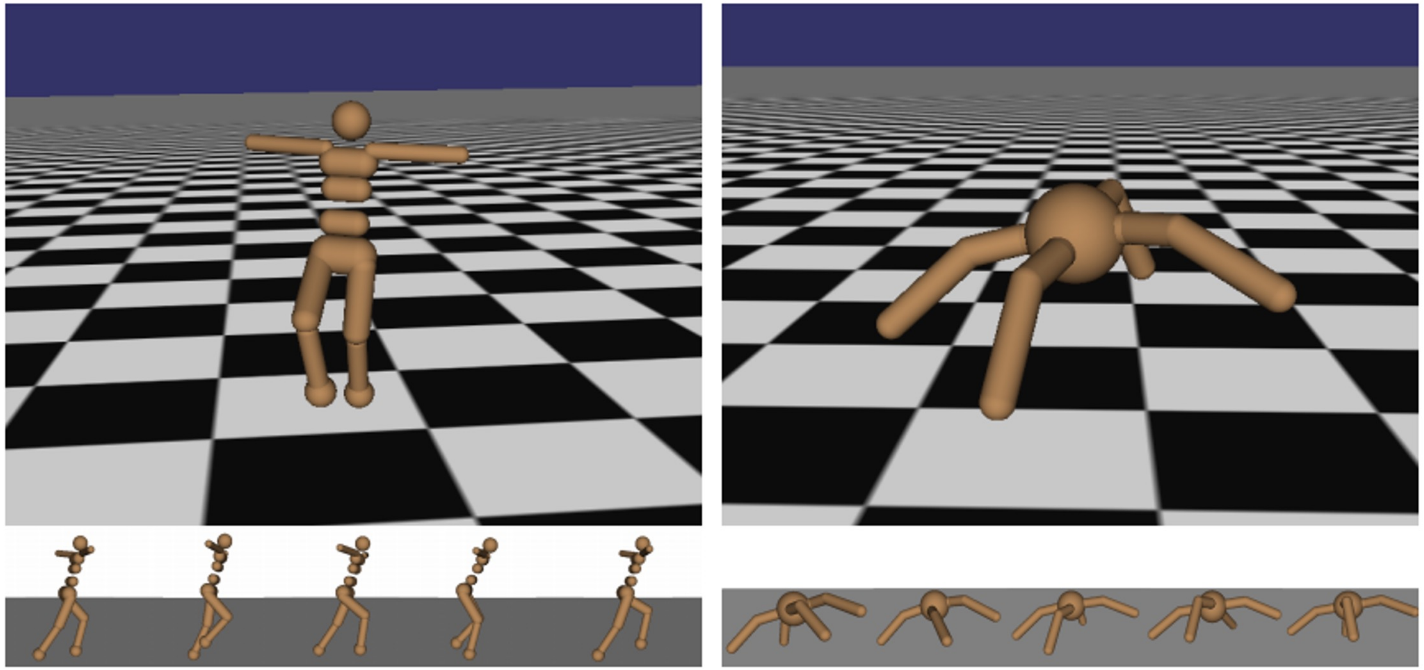


Figure from: Schulman et al, “High-Dimensional Continuous Control Using Generalized Advantage Estimation”, ICLR 2016

Objective: Make the robot move forward

State: Angle, position, velocity of all joints

Action: Torques applied on joints

Reward: 1 at each time step upright + forward movement

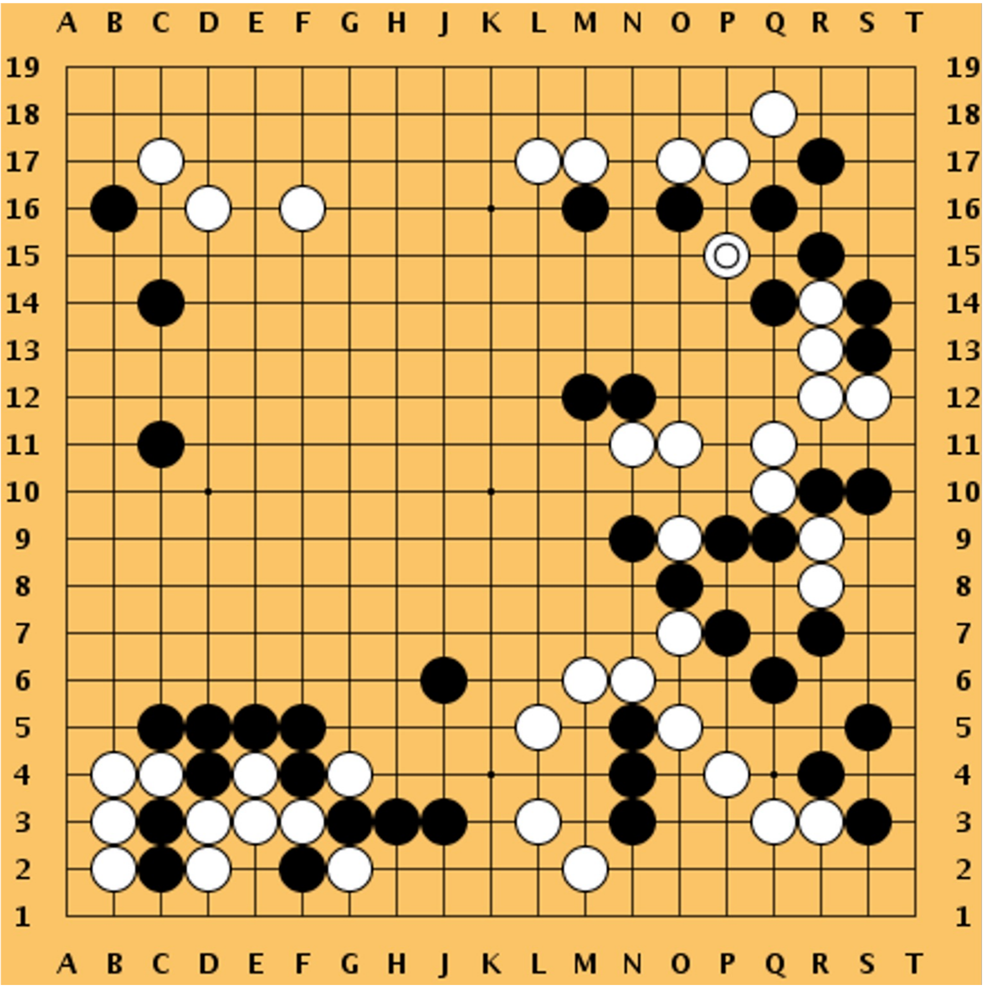
Example: Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game screen
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

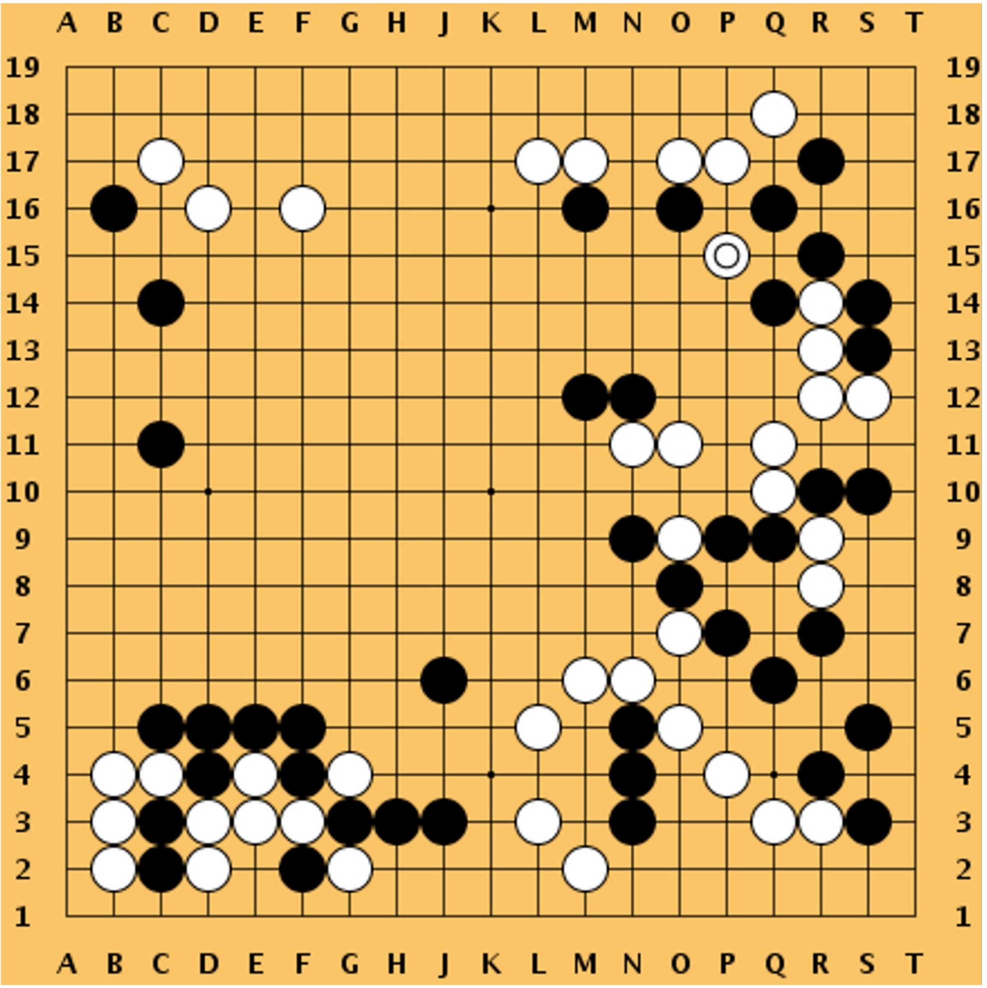
Mnih et al, “Playing Atari with Deep Reinforcement Learning”, NeurIPS Deep Learning Workshop, 2013

Example: Go



Objective: Win the game!

Example: Go



Objective: Win the game!

State: Position of all pieces

Action: Where to put the next piece down

Reward: On last turn: 1 if you won, 0 if you lost

Example: Image Classification

Classification

Cat

Objective: Classify the image!

Example: Image Classification

Classification

Cat

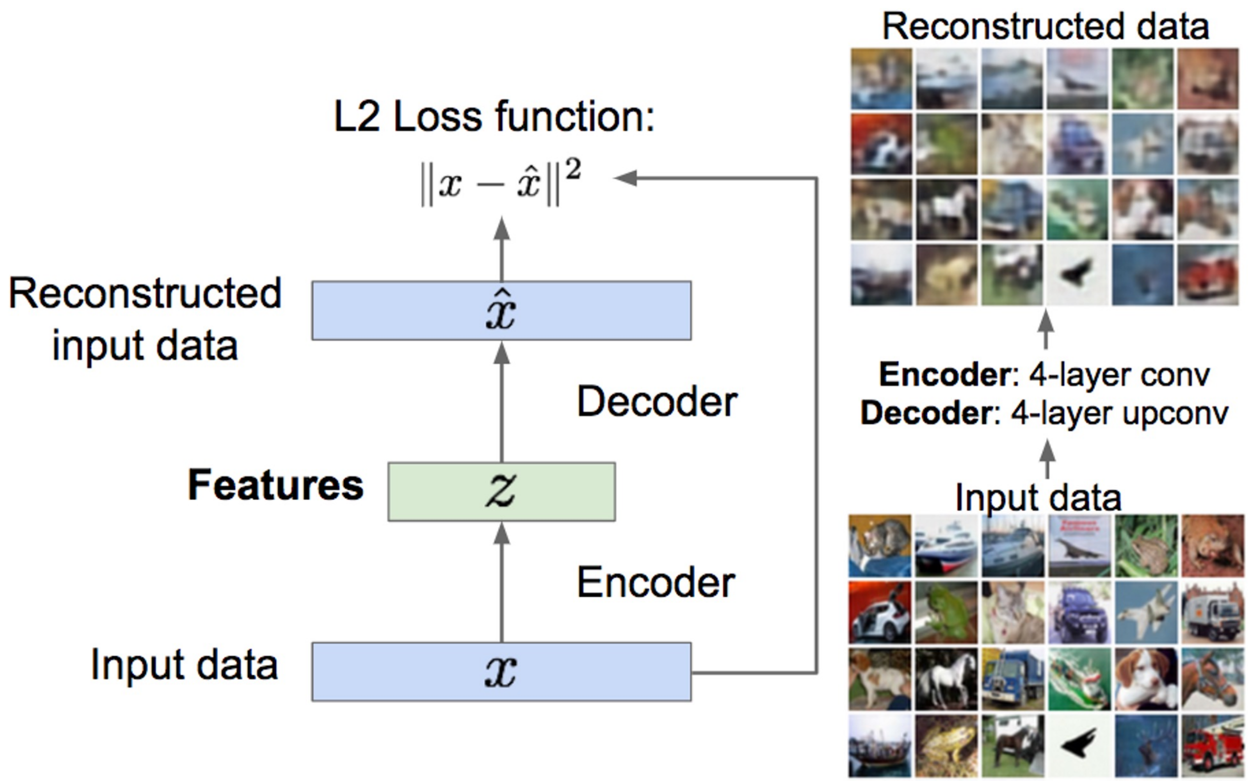
Objective: Classify the image!

State: Raw pixels

Action: Class labels

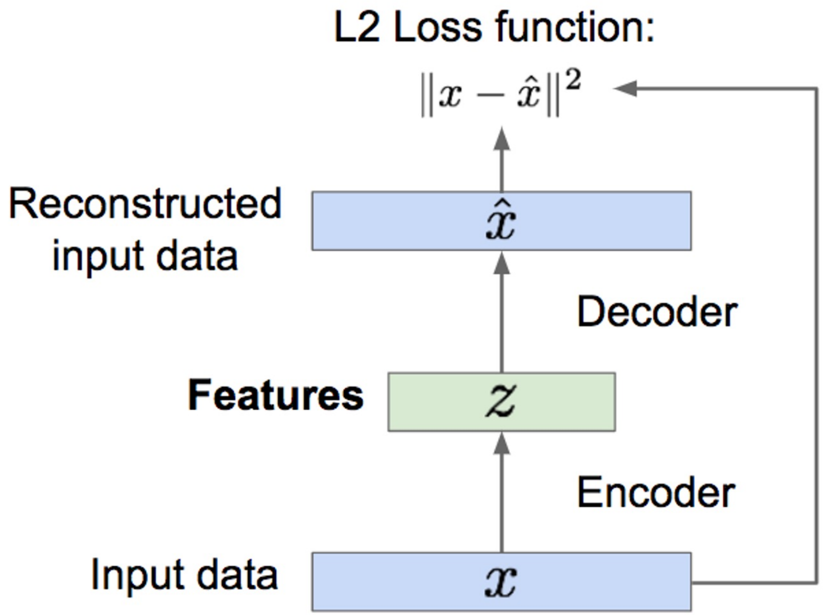
Reward: 1 if you classify correctly, 0 ow

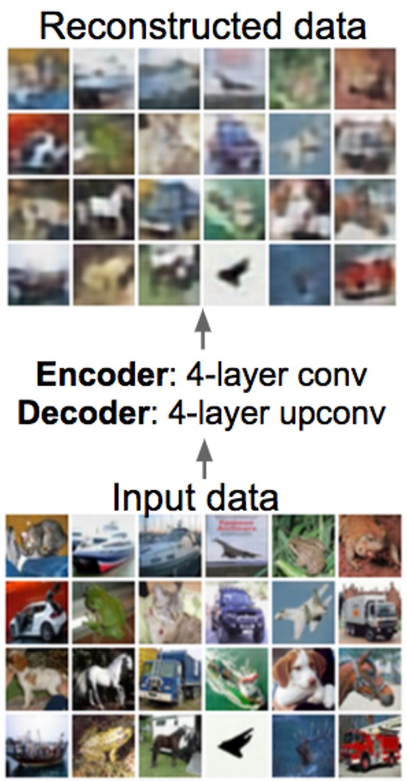
Example: Image Reconstruction



Objective: Reconstruct the whole image!

Example: Image Reconstruction





Objective: Reconstruct the whole image!

State: Raw pixels

Action: Raw pixels

Reward: Reconstruction loss (negated)

Example: Training Your Dog to Sit

Objective: Teach a dog to sit!

Example: Training Your Dog to Sit

Objective: Teach a dog to sit!

State: Posture of the dog

Action: Where to put the next piece down

Reward: “Good girl!” + treat if sit down

Example: Painting Robot

Objective: Replicate this painting

Example: Painting Robot

Objective: Replicate this painting

State: Raw pixels of canvas

Action: Strokes

Reward: Replication loss (negated)

Example: Text Generation

Objective: Predict the next word!

<s> CS231n midterm was ___

Example: Text Generation

<s> CS231n midterm was ___

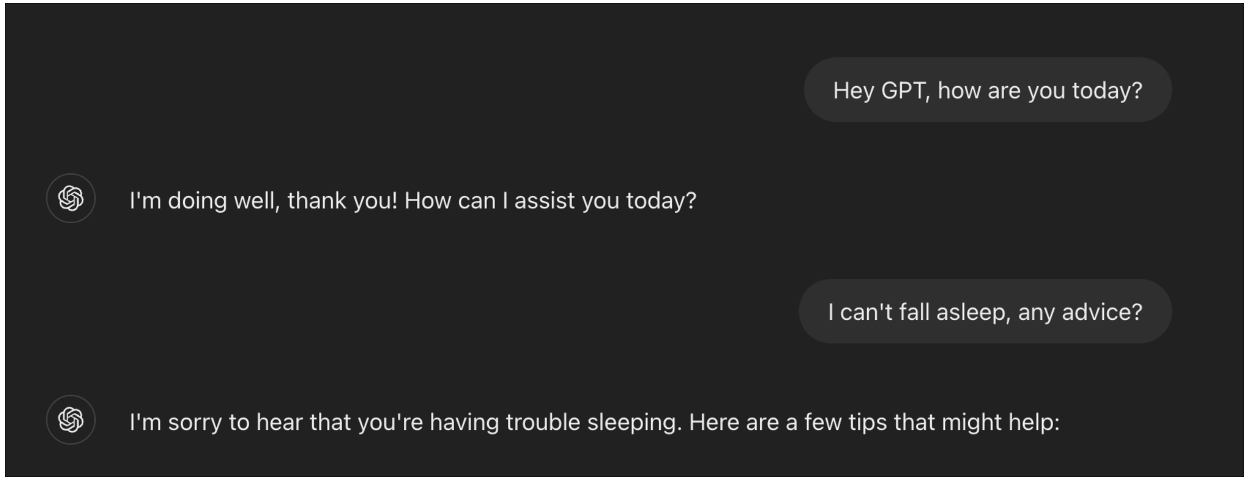
Objective: Predict the next word!

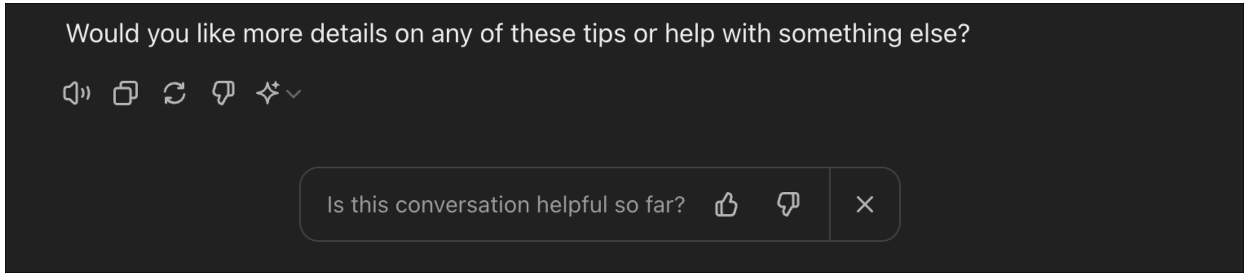
State: Current words in the sentence

Action: Next word

Reward: 1 if correct, 0 ow

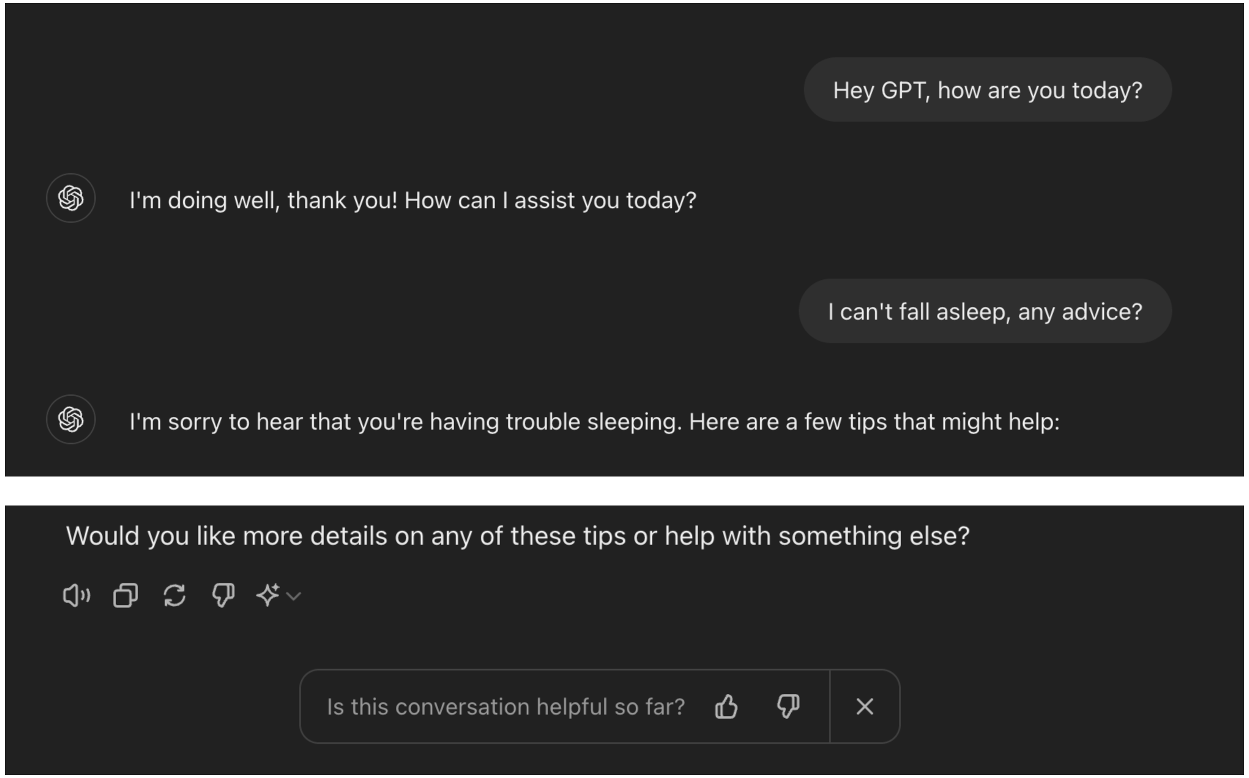
Example: Chatbot





Objective: Be a good companion!

Example: Chatbot



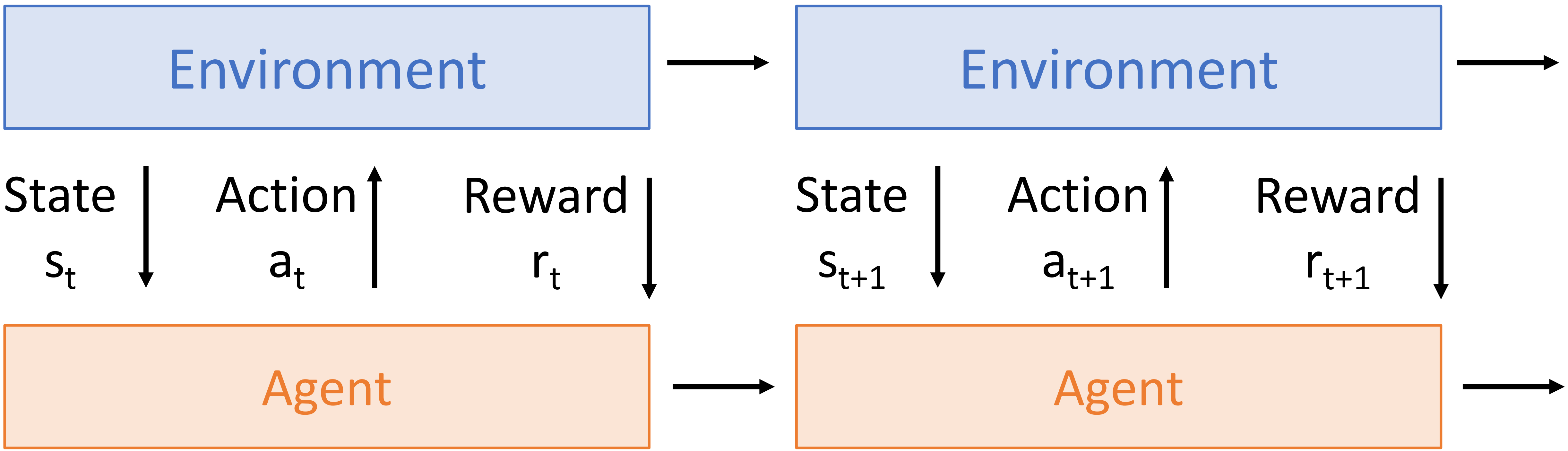
Objective: Be a good companion!

State: Current conversation

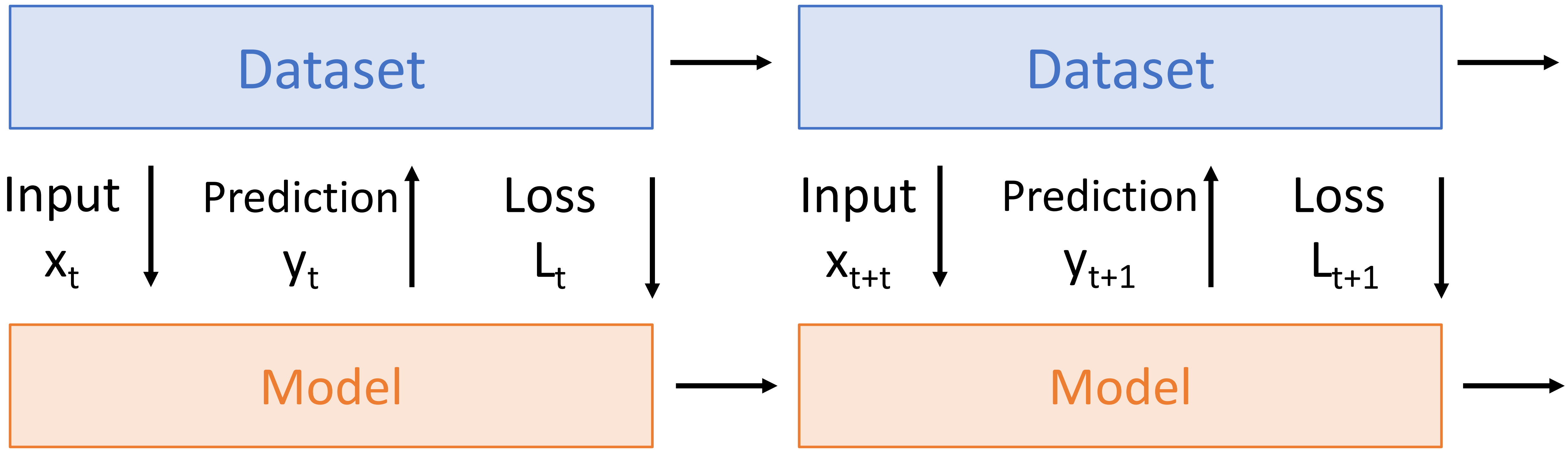
Action: Next sentence

Reward: Human evaluation, 1 if satisfied, -1 if unsatisfied, 0 neutral

Reinforcement Learning vs Supervised Learning

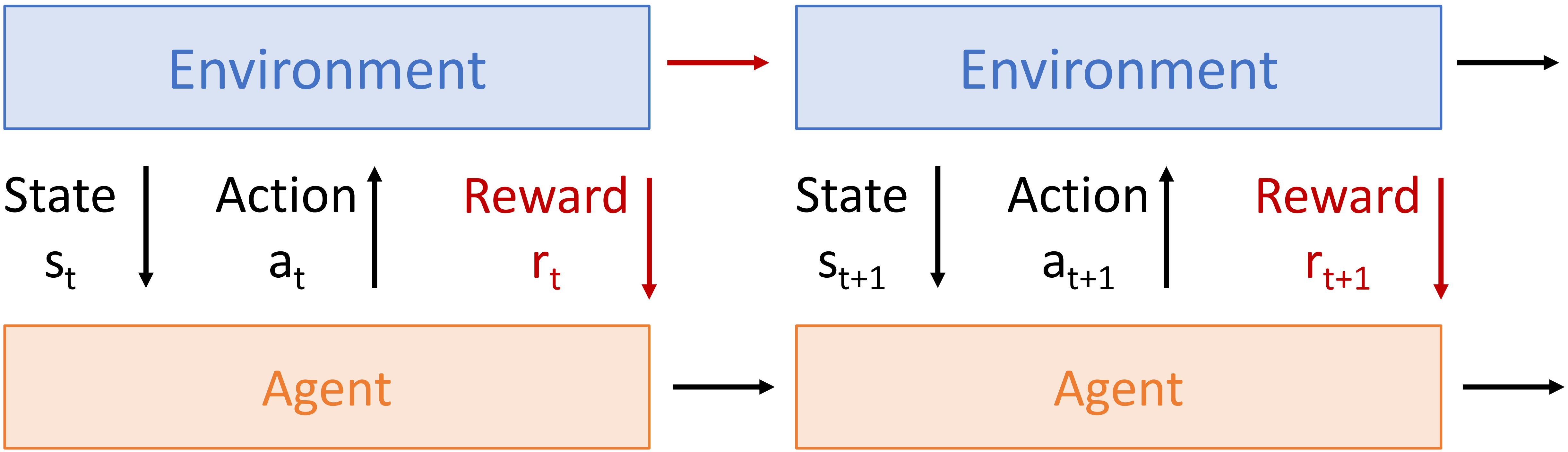


Reinforcement Learning vs Supervised Learning



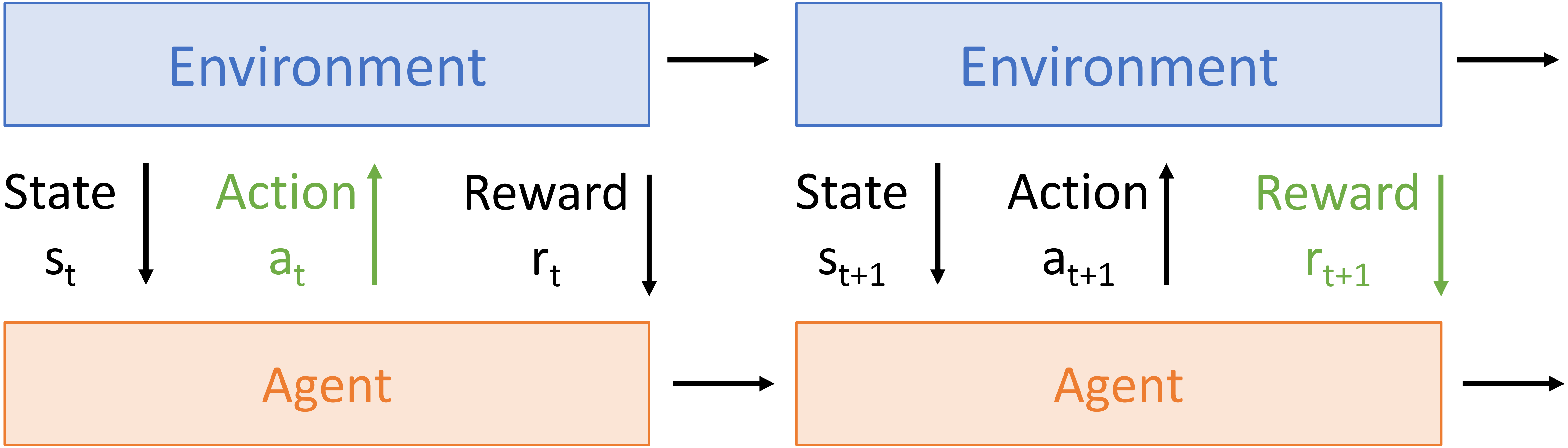
Why is RL different from normal supervised learning?

Reinforcement Learning vs Supervised Learning



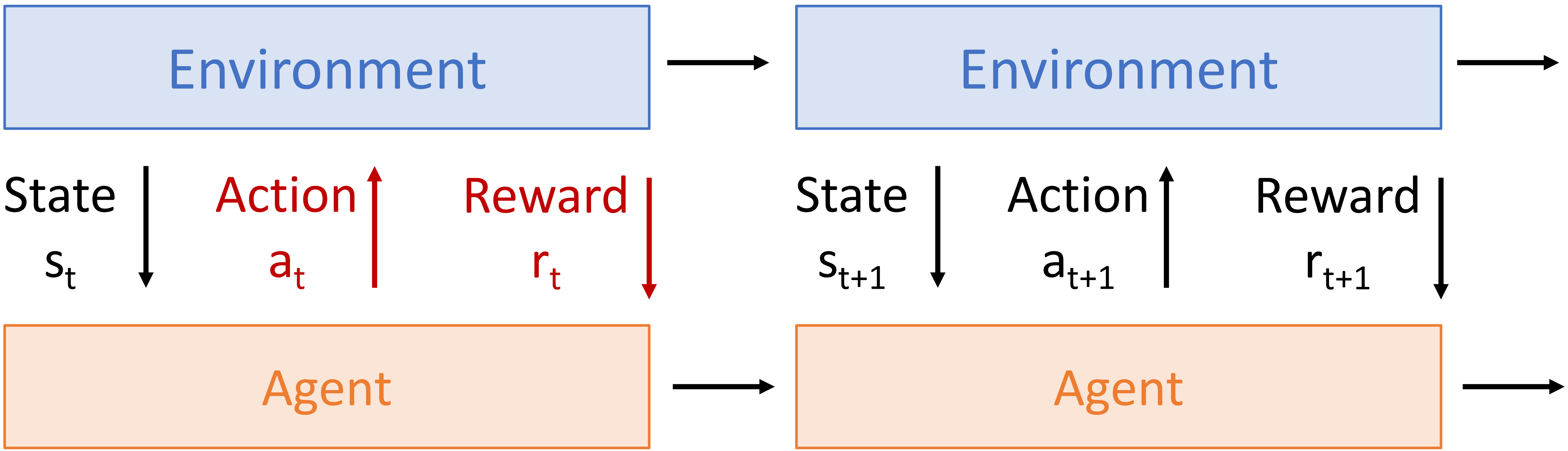
Stochasticity: Rewards and state transitions may be random

Reinforcement Learning vs Supervised Learning



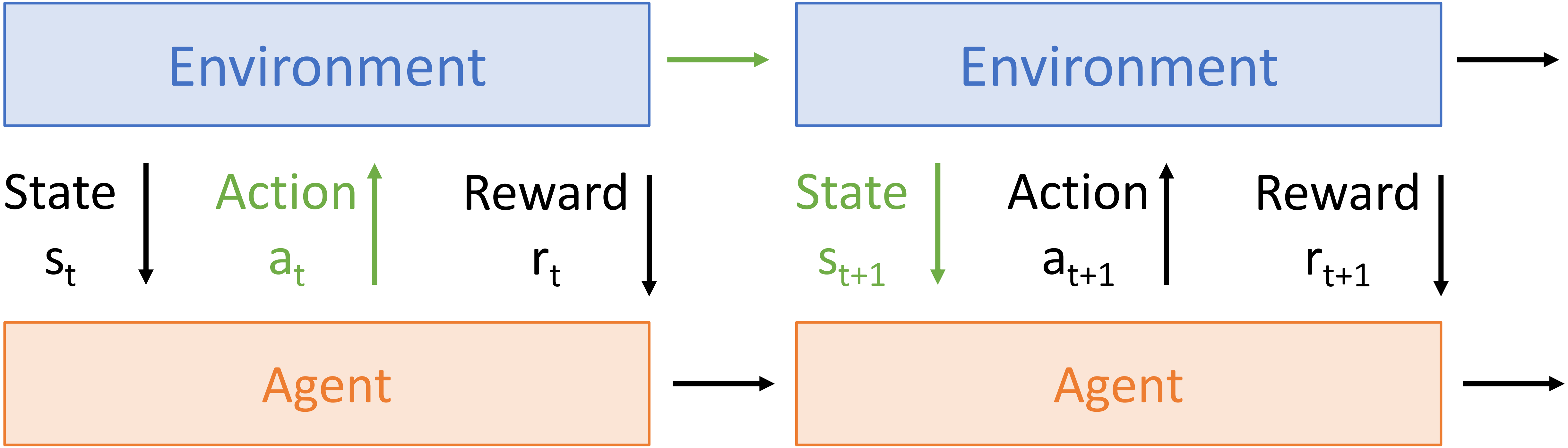
Credit assignment: Reward r_t may not directly depend on action a_t

Reinforcement Learning vs Supervised Learning



Nondifferentiable: Can't backprop through world; can't compute dr_t/da_t

Reinforcement Learning vs Supervised Learning



Nonstationary: What the agent experiences depends on how it acts

Markov Decision Process (MDP)

Mathematical formalization of the RL problem: A tuple (S, A, R, P, γ)

S: Set of possible states
A: Set of possible actions
R: Distribution of reward given (state, action) pair
P: Transition probability: distribution over next state given (state, action)
γ: Discount factor (tradeoff between future and present rewards)

Markov Property: The current state completely characterizes the state of the world. Rewards and next states depend only on current state, not history.

Markov Decision Process (MDP)

Mathematical formalization of the RL problem: A tuple (S, A, R, P, γ)

S: Set of possible states
A: Set of possible actions
R: Distribution of reward given (state, action) pair
P: Transition probability: distribution over next state given (state, action)
γ: Discount factor (tradeoff between future and present rewards)

Agent executes a **policy** π giving distribution of actions conditioned on states

Markov Decision Process (MDP)

Mathematical formalization of the RL problem: A tuple (S, A, R, P, γ)

S: Set of possible states
A: Set of possible actions
R: Distribution of reward given (state, action) pair
P: Transition probability: distribution over next state given (state, action)
γ: Discount factor (tradeoff between future and present rewards)

Agent executes a **policy** π giving distribution of actions conditioned on states
Goal: Find policy π^* that maximizes cumulative discounted reward: $\sum_t \gamma^t r_t$

Markov Decision Process (MDP)

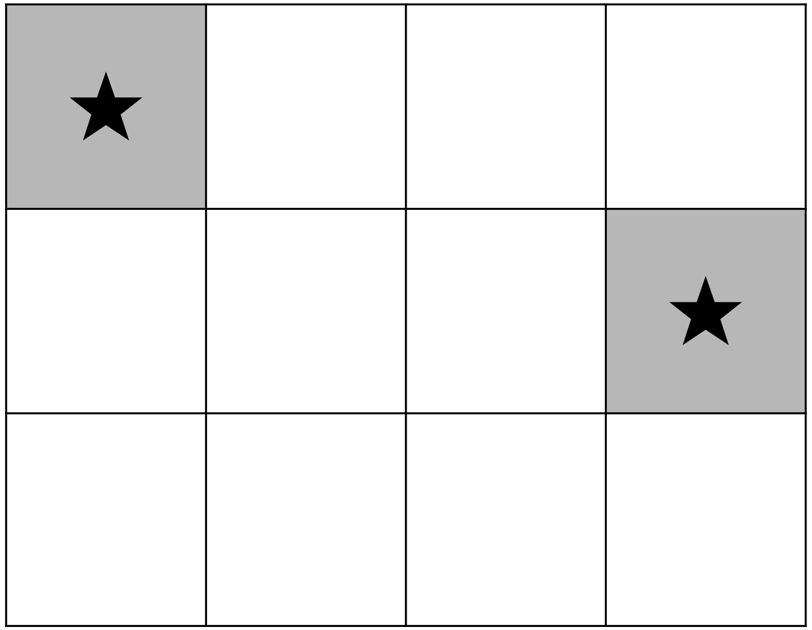
- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action $a_t \sim \pi(a \mid s_t)$
  - Environment samples reward $r_t \sim R(r \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(s \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

A simple MDP: Grid World

Actions:

1. Right
2. Left
3. Up
4. Down

States



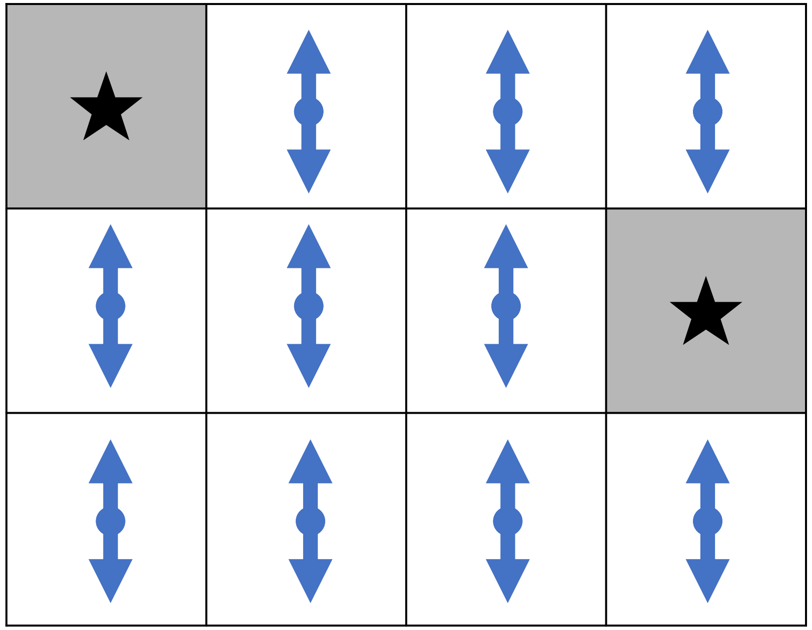
Reward

Set a negative "reward" for each transition (e.g. r = -1)

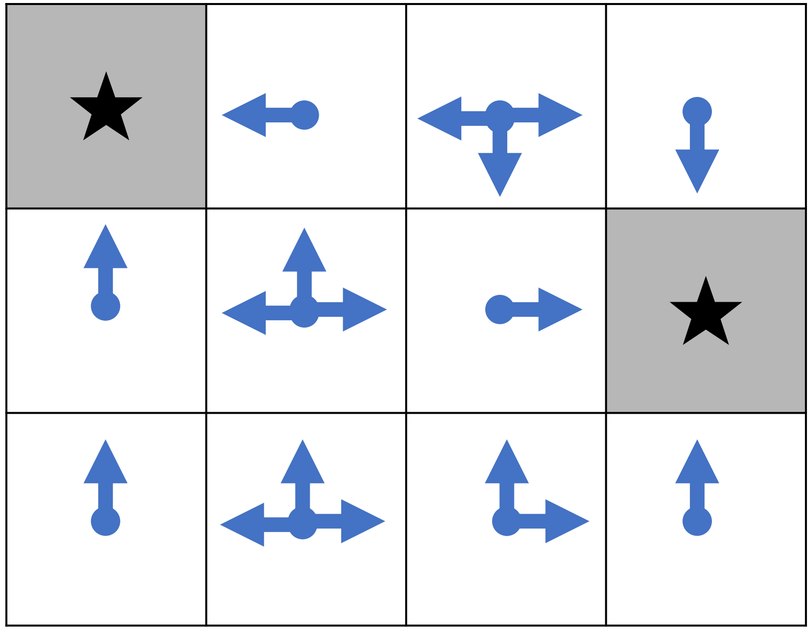
Objective: Reach one of the terminal states in as few moves as possible

A simple MDP: Grid World

Bad policy



Optimal Policy



Finding Optimal Policies

Goal: Find the optimal policy π^* that maximizes (discounted) sum of rewards.

Finding Optimal Policies

Goal: Find the optimal policy π^* that maximizes (discounted) sum of rewards.

Problem: Lots of randomness! Initial state, transition probabilities, rewards

Finding Optimal Policies

Goal: Find the optimal policy π^* that maximizes (discounted) sum of rewards.

Problem: Lots of randomness! Initial state, transition probabilities, rewards

Solution: Maximize the expected sum of rewards

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi\right]$$

$$s_0 \sim p(s_0)$$
$$a_t \sim \pi(a \mid s_t)$$
$$s_{t+1} \sim P(s \mid s_t, a_t)$$

Value Function and Q Function

Following a policy π produces **sample trajectories** (or paths) s_0, a_0, r_0, s_1, a_1, r_1, …

Value Function and Q Function

Following a policy π produces **sample trajectories** (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

How good is a state? The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

Value Function and Q Function

Following a policy π produces **sample trajectories** (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

How good is a state? The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right]$$

How good is a state-action pair? The **Q function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Bellman Equation

Optimal Q-function: $Q^*(s, a)$ is the Q-function for the optimal policy π^*

It gives the max possible future reward when taking action a in state s:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Bellman Equation

Optimal Q-function: $Q^*(s, a)$ is the Q-function for the optimal policy π^*

It gives the max possible future reward when taking action a in state s:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Q* encodes the optimal policy: $\pi^*(s) = \arg\max_{a'} Q(s,a')$

Bellman Equation

Optimal Q-function: $Q^*(s, a)$ is the Q-function for the optimal policy π^*
It gives the max possible future reward when taking action a in state s:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Q* encodes the optimal policy: $\pi^*(s) = \arg\max_{a'} Q(s,a')$

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Bellman Equation

Optimal Q-function: $Q^*(s, a)$ is the Q-function for the optimal policy π^*
It gives the max possible future reward when taking action a in state s:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

Q* encodes the optimal policy: $\pi^*(s) = \arg\max_{a\prime} Q(s, a')$

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s\prime}\left[r + \gamma \max_{a\prime} Q^*(s', a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Intuition: After taking action a in state s, we get reward r and move to a new state s'. After that, the max possible reward we can get is $\max_{a\prime} Q^*(s', a')$

Solving for the optimal policy: Value Iteration

Bellman Equation: Q* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Idea: If we find a function Q(s, a) that satisfies the Bellman Equation, then it must be Q*.

Solving for the optimal policy: Value Iteration

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Idea: If we find a function Q(s, a) that satisfies the Bellman Equation, then it must be Q^*.
Start with a random Q, and use the Bellman Equation as an update rule:

$$Q_{i+1}(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q_i(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Solving for the optimal policy: Value Iteration

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Idea: If we find a function Q(s, a) that satisfies the Bellman Equation, then it must be Q^*.
Start with a random Q, and use the Bellman Equation as an update rule:

$$Q_{i+1}(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q_i(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Amazing fact: Q_i converges to Q^* as $i \to \infty$

Solving for the optimal policy: Value Iteration

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Idea: If we find a function Q(s, a) that satisfies the Bellman Equation, then it must be Q^*.
Start with a random Q, and use the Bellman Equation as an update rule:

$$Q_{i+1}(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q_i(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Amazing fact: Q_i converges to Q^* as $i \to \infty$

Problem: Need to keep track of Q(s, a) for all (state, action) pairs – impossible if infinite

Solving for the optimal policy: Value Iteration

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Idea: If we find a function Q(s, a) that satisfies the Bellman Equation, then it must be Q^*.
Start with a random Q, and use the Bellman Equation as an update rule:

$$Q_{i+1}(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q_i(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Amazing fact: Q_i converges to Q^* as $i \to \infty$
Problem: Need to keep track of Q(s, a) for all (state, action) pairs – impossible if infinite
Solution: Approximate Q(s, a) with a neural network, use Bellman Equation as loss!

Solving for the optimal policy: Deep Q-Learning

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Train a neural network (with weights θ) to approximate Q^*: $Q^*(s,a) \approx Q(s,a;\theta)$

Solving for the optimal policy: Deep Q-Learning

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Train a neural network (with weights θ) to approximate Q^*: $Q^*(s,a) \approx Q(s,a;\theta)$

Use the Bellman Equation to tell what Q should output for a given state and action:

$$y_{s,a,\theta} = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q(s',a';\theta)\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Solving for the optimal policy: Deep Q-Learning

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Train a neural network (with weights θ) to approximate Q^*: $Q^*(s,a) \approx Q(s,a;\theta)$

Use the Bellman Equation to tell what Q should output for a given state and action:

$$y_{s,a,\theta} = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q(s',a';\theta)\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Use this to define the loss for training Q: $L(s,a) = \left(Q(s,a;\theta) - y_{s,a,\theta}\right)^2$

Solving for the optimal policy: Deep Q-Learning

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Train a neural network (with weights θ) to approximate Q^*: $Q^*(s,a) \approx Q(s,a;\theta)$

Use the Bellman Equation to tell what Q should output for a given state and action:

$$y_{s,a,\theta} = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q(s',a';\theta)\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Use this to define the loss for training Q: $L(s,a) = \left(Q(s,a;\theta) - y_{s,a,\theta}\right)^2$

Problem: Nonstationary! The “target” for Q(s, a) depends on the current weights θ!

Solving for the optimal policy: Deep Q-Learning

Bellman Equation: Q^* satisfies the following recurrence relation:

$$Q^*(s,a) = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Train a neural network (with weights θ) to approximate Q^*: $Q^*(s,a) \approx Q(s,a;\theta)$

Use the Bellman Equation to tell what Q should output for a given state and action:

$$y_{s,a,\theta} = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q(s',a';\theta)\right]$$

Where $r \sim R(s,a), s' \sim P(s,a)$

Use this to define the loss for training Q: $L(s,a) = \left(Q(s,a;\theta) - y_{s,a,\theta}\right)^2$

Problem: Nonstationary! The "target" for Q(s, a) depends on the current weights θ!

Problem: How to sample batches of data for training?

Case Study: Playing Atari Games

Objective: Complete the game with the highest score

State: Raw pixel inputs of the game screen
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Mnih et al, "Playing Atari with Deep Reinforcement Learning", NeurIPS Deep Learning Workshop, 2013

Mnih et al, "Playing Atari with Deep Reinforcement Learning", NeurIPS Deep Learning Workshop, 2013

Case Study: Playing Atari Games

$Q(s, a; \theta)$
Neural network with weights θ

Network output:
Q-values for all actions

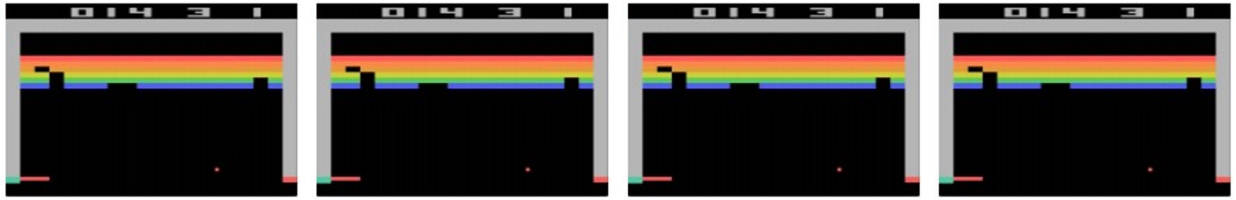
FC-A (Q-values)

FC-256

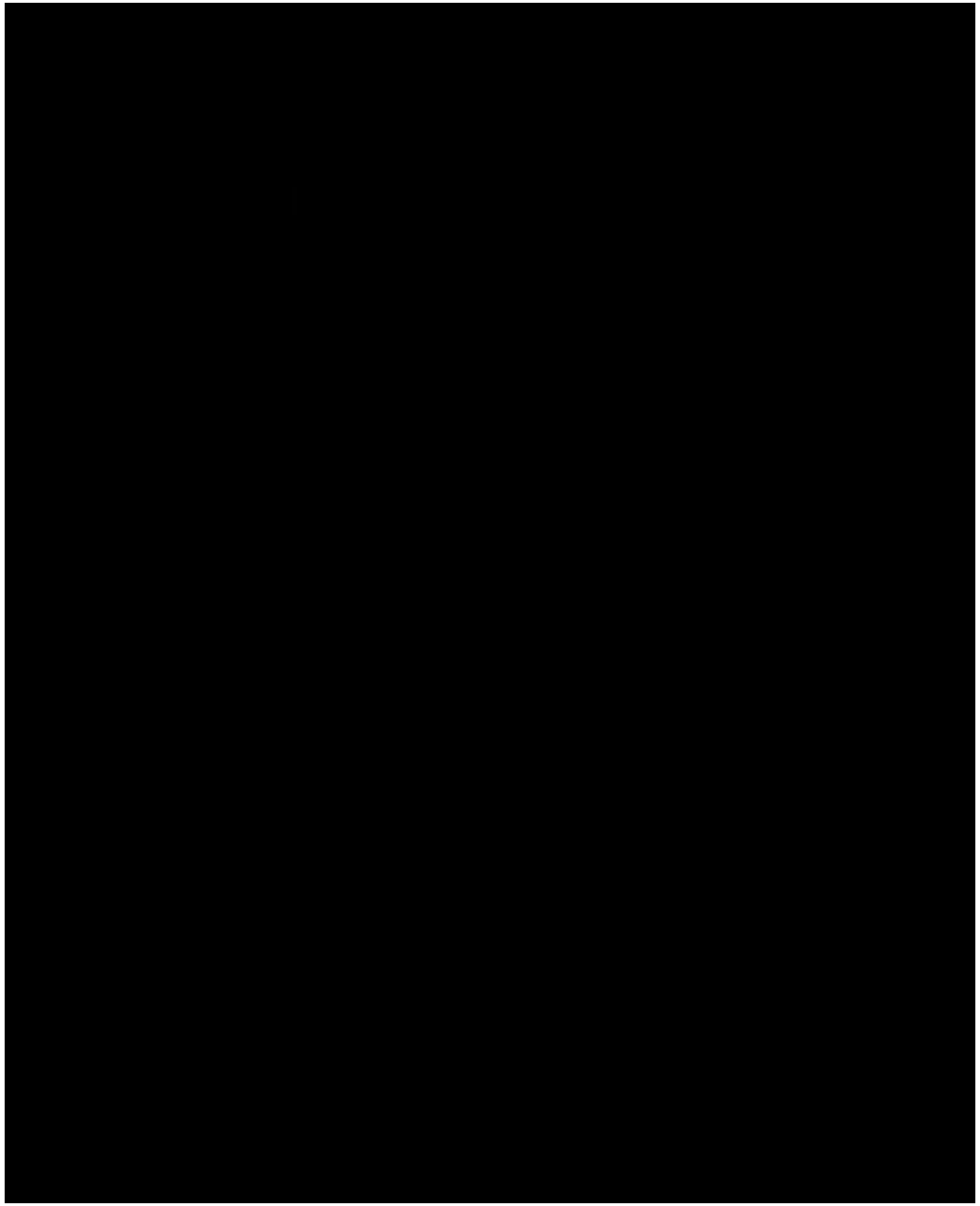
Conv(16->32, 4x4, stride 2)

Conv(4->16, 8x8, stride 4)

With 4 actions: last layer gives values $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$



Network input: state s_t: 4x84x84 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)



https://www.youtube.com/watch?v=V1eYniJ0Rnk

Q-Learning

Q-Learning: Train network $Q_\theta(s, a)$ to estimate future rewards for every (state, action) pair

Problem: For some problems this can be a hard function to learn.
For some problems it is easier to learn a mapping from states to actions

Q-Learning vs Policy Gradients

Q-Learning: Train network $Q_\theta(s, a)$ to estimate future rewards for every (state, action) pair

Problem: For some problems this can be a hard function to learn.
For some problems it is easier to learn a mapping from states to actions

Policy Gradients: Train a network $\pi_\theta(a \mid s)$ that takes state as input, gives distribution over which action to take in that state

Q-Learning vs Policy Gradients

Q-Learning: Train network $Q_\theta(s, a)$ to estimate future rewards for every (state, action) pair

Problem: For some problems this can be a hard function to learn.
For some problems it is easier to learn a mapping from states to actions

Policy Gradients: Train a network $\pi_\theta(a \mid s)$ that takes state as input, gives distribution over which action to take in that state

Objective function: Expected future rewards when following policy π_θ:

$$J(\theta) = \mathbb{E}_{r \sim p_\theta}\left[\sum_{t \geq 0} \gamma^t r_t\right]$$

Find the optimal policy by maximizing: $\theta^* = \arg\max_\theta J(\theta)$ **(Use gradient ascent!)**

Policy Gradients

Objective function: Expected future rewards when following policy π_θ:

$$J(\theta) = \mathbb{E}_{r\sim p_\theta}\left[\sum_{t\geq 0} \gamma^t r_t\right]$$

Find the optimal policy by maximizing: $\theta^* = \arg\max_\theta J(\theta)$ **(Use gradient ascent!)**

Problem: Nondifferentiability! Don't know how to compute $\frac{\partial J}{\partial \theta}$

Policy Gradients

Objective function: Expected future rewards when following policy π_θ:

$$J(\theta) = \mathbb{E}_{r \sim p_\theta}\left[\sum_{t \geq 0} \gamma^t r_t\right]$$

Find the optimal policy by maximizing: $\theta^* = \arg\max_\theta J(\theta)$ **(Use gradient ascent!)**

Problem: Nondifferentiability! Don't know how to compute $\frac{\partial J}{\partial \theta}$

General formulation: $J(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)]$ Want to compute $\frac{\partial J}{\partial \theta}$

So far: Q-Learning and Policy Gradients

Q-Learning: Train network $Q_\theta(s, a)$ to estimate future rewards for every (state, action) pair
Use Bellman Equation to define loss function for training Q:

$$y_{s,a,\theta} = \mathbb{E}_{r,s'}\left[r + \gamma \max_{a'} Q(s', a'; \theta)\right] \qquad \text{Where } r \sim R(s, a), s' \sim P(s, a)$$

$$L(s, a) = \left(Q(s, a; \theta) - y_{s,a,\theta}\right)^2$$

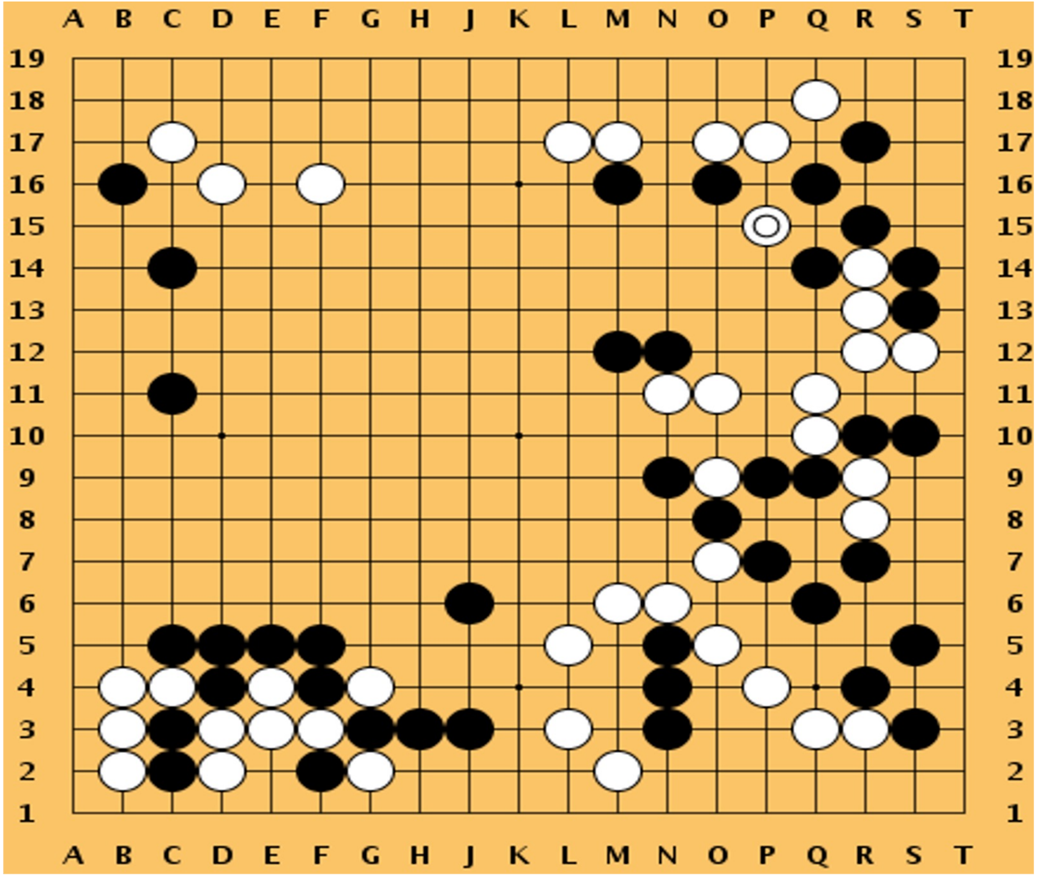
Policy Gradients: Train a network $\pi_\theta(a \mid s)$ that takes state as input, gives distribution over which action to take in that state. Use REINFORCE Rule for computing gradients:

$$J(\theta) = \mathbb{E}_{x\sim p_\theta}[f(x)] \qquad \frac{\partial J}{\partial \theta} = \mathbb{E}_{x\sim p_\theta}\left[f(x) \textstyle\sum_{t\geq 0} \frac{\partial}{\partial \theta} log\, \pi_\theta(a_t | s_t)\right]$$

Improving policy gradients: Add **baseline** to reduce variance of gradient estimator

Case Study: Playing Games

AlphaGo: (January 2016)
- Used imitation learning + tree search + RL
- Beat 18-time world champion Lee Sedol



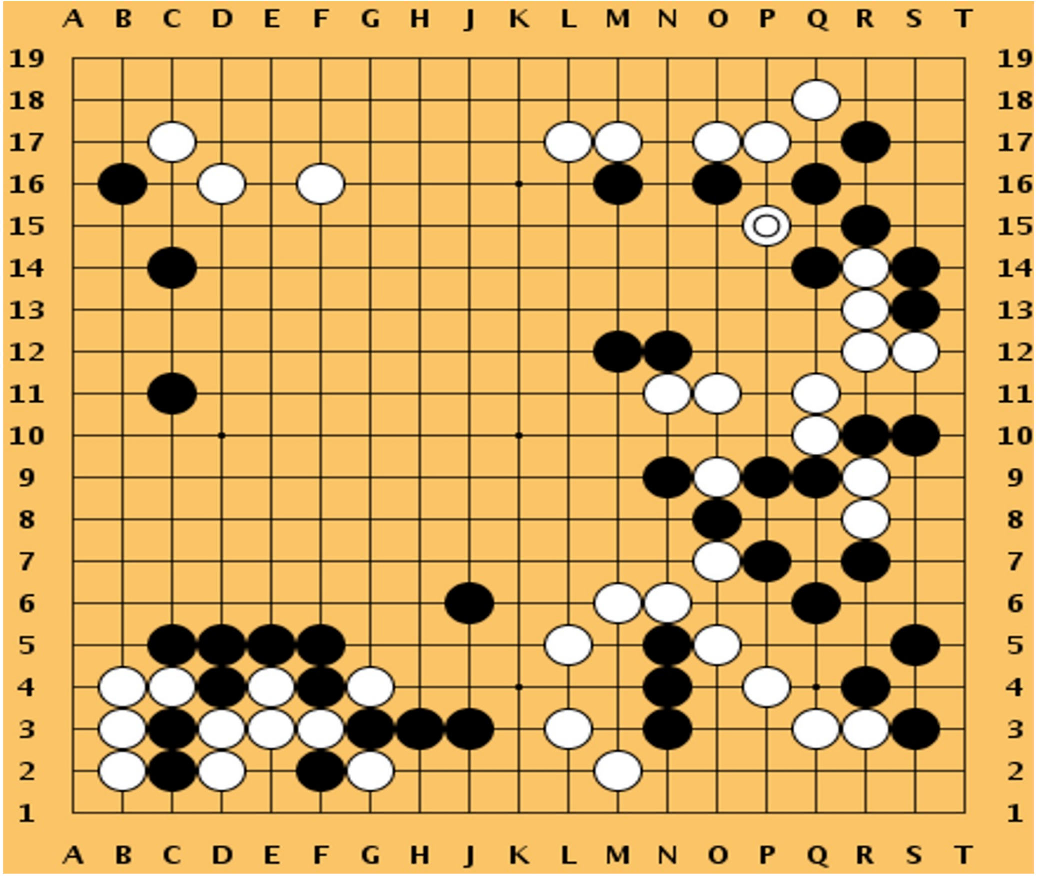
This image is CC0 public domain

Silver et al, "Mastering the game of Go with deep neural networks and tree search", Nature 2016
Silver et al, "Mastering the game of Go without human knowledge", Nature 2017
Silver et al, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play", Science 2018
Schrittwieser et al, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model", arXiv 2019

Case Study: Playing Games

AlphaGo: (January 2016)
- Used imitation learning + tree search + RL
- Beat 18-time world champion Lee Sedol

AlphaGo Zero (October 2017)
- Simplified version of AlphaGo
- No longer using imitation learning
- Beat (at the time) #1 ranked Ke Jie



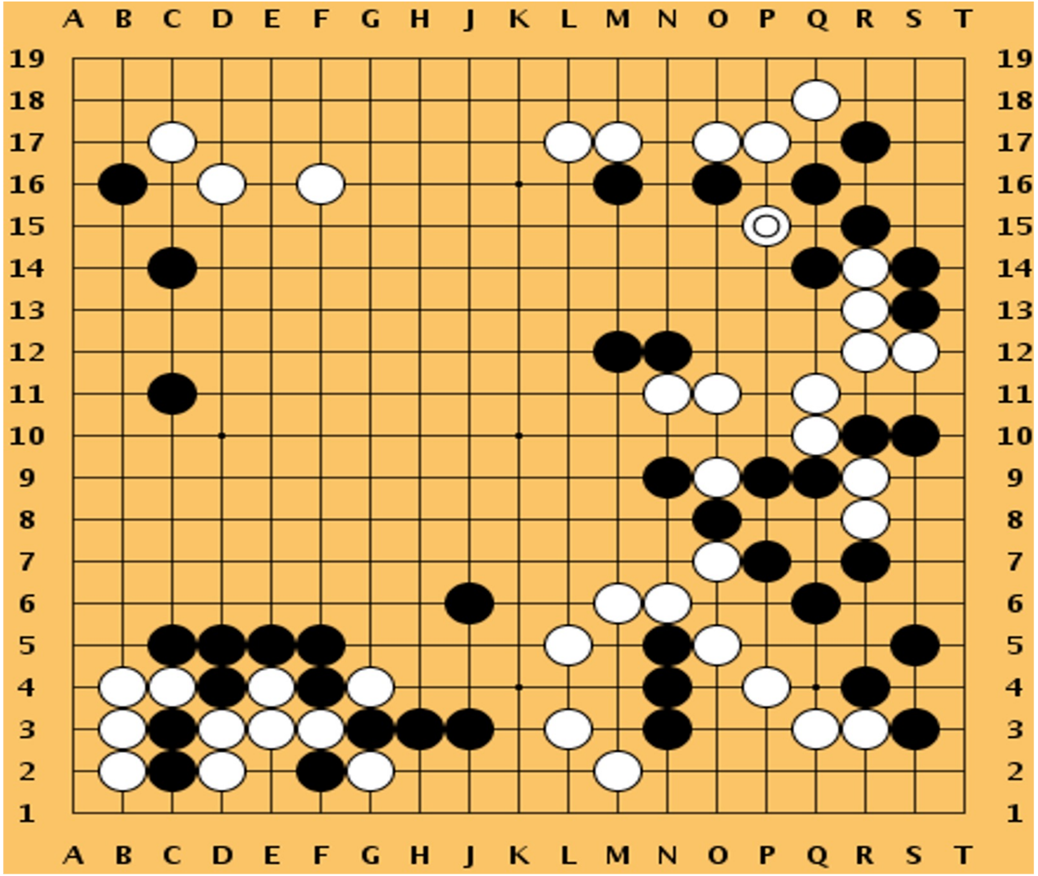
Silver et al, “Mastering the game of Go with deep neural networks and tree search”, Nature 2016
Silver et al, “Mastering the game of Go without human knowledge”, Nature 2017
Silver et al, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play”, Science 2018
Schrittwieser et al, “Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model”, arXiv 2019

Case Study: Playing Games

AlphaGo: (January 2016)
- Used imitation learning + tree search + RL
- Beat 18-time world champion Lee Sedol

AlphaGo Zero (October 2017)
- Simplified version of AlphaGo
- No longer using imitation learning
- Beat (at the time) #1 ranked Ke Jie

Alpha Zero (December 2018)
- Generalized to other games: Chess and Shogi



Silver et al, "Mastering the game of Go with deep neural networks and tree search", Nature 2016
Silver et al, "Mastering the game of Go without human knowledge", Nature 2017
Silver et al, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play", Science 2018
Schrittwieser et al, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model", arXiv 2019

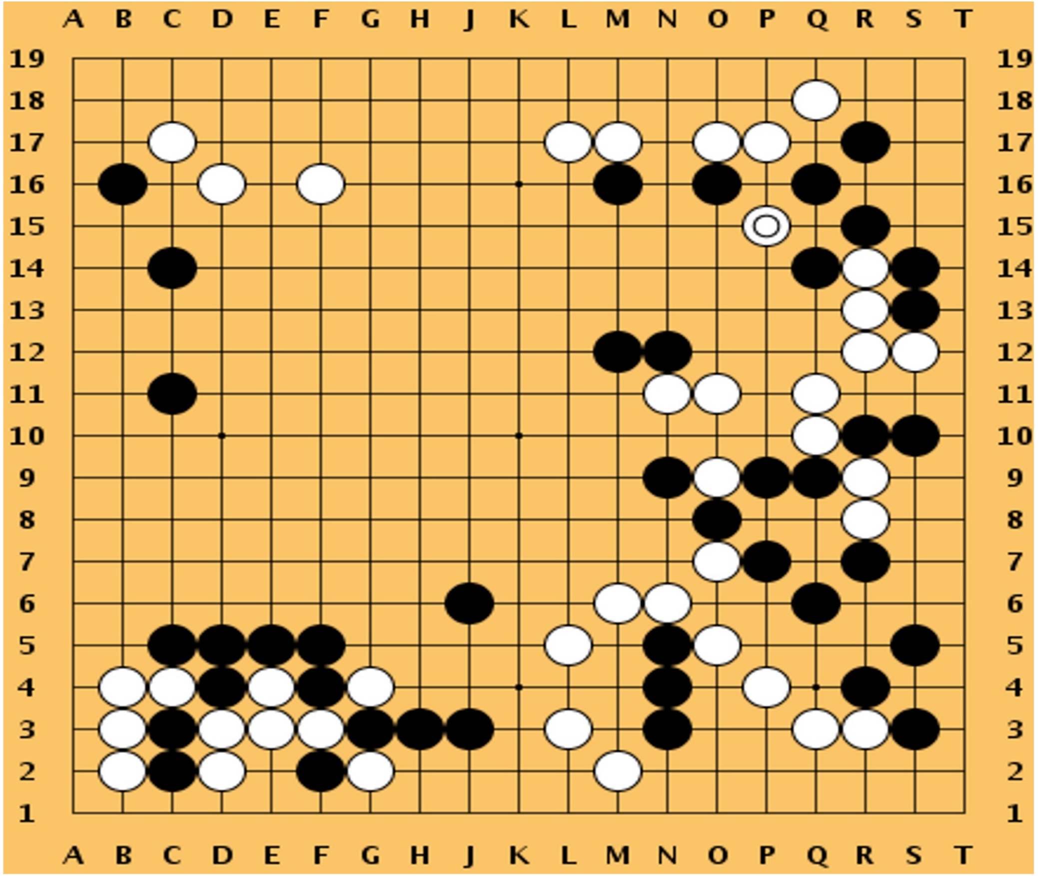
Case Study: Playing Games

AlphaGo: (January 2016)
- Used imitation learning + tree search + RL
- Beat 18-time world champion Lee Sedol

AlphaGo Zero (October 2017)
- Simplified version of AlphaGo
- No longer using imitation learning
- Beat (at the time) #1 ranked Ke Jie

Alpha Zero (December 2018)
- Generalized to other games: Chess and Shogi

MuZero (November 2019)
- Plans through a learned model of the game



Silver et al, "Mastering the game of Go with deep neural networks and tree search", Nature 2016
Silver et al, "Mastering the game of Go without human knowledge", Nature 2017
Silver et al, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play", Science 2018
Schrittwieser et al, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model", arXiv 2019

Case Study: Playing Games

AlphaGo: (January 2016)
- Used imitation learning + tree search + RL
- Beat 18-time world champion Lee Sedol

AlphaGo Zero (October 2017)
- Simplified version of AlphaGo
- No longer using imitation learning
- Beat (at the time) #1 ranked Ke Jie

Alpha Zero (December 2018)
- Generalized to other games: Chess and Shogi

MuZero (November 2019)
- Plans through a learned model of the game

November 2019: Lee Sedol announces retirement

"With the debut of AI in Go games, I've realized that I'm not at the top even if I become the number one through frantic efforts"
"Even if I become the number one, there is an entity that cannot be defeated"

Silver et al, "Mastering the game of Go with deep neural networks and tree search", Nature 2016
Silver et al, "Mastering the game of Go without human knowledge", Nature 2017
Silver et al, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play", Science 2018
Schrittwieser et al, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model", arXiv 2019

Quotes from: https://en.yna.co.kr/view/AEN20191127004800315

More Complex Games

StarCraft II: AlphaStar (October 2019)
Vinyals et al, "Grandmaster level in StarCraft II using multi-agent reinforcement learning", Science 2018

Dota 2: OpenAI Five (April 2019)
No paper, only a blog post:
https://openai.com/five/#how-openai-five-works

Problems of Model-Free RL

- Learns from trials and error
- Require extensive interactions

AlphaGo Zero: Google DeepMind supercomputer learns 3,000 years of human knowledge in 40 days

Problems of Model-Free RL

- Learns from trials and error
- Require extensive interactions

- Safety concerns
- Limited interpretability
    - What if things go wrong?

Problems of Model-Free RL

- Learns from trials and error
- Require extensive interactions

- Safety concerns
- Limited interpretability
  - What if things go wrong?

- Humans maintain an intuitive model of the world
  - Widely applicable
  - Sample efficient

Markov Decision Process (MDP)

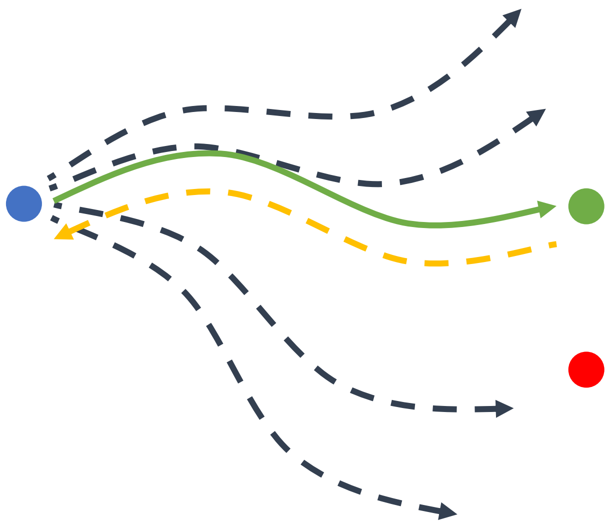
Mathematical formalization of the RL problem: A tuple (S, A, R, P, γ)

S: Set of possible states
A: Set of possible actions
R: Distribution of reward given (state, action) pair
P: Transition probability: distribution over next state given (state, action)
γ: Discount factor (tradeoff between future and present rewards)

Agent executes a **policy** π giving distribution of actions conditioned on states
Goal: Find policy π^* that maximizes cumulative discounted reward: $\sum_t \gamma^t r_t$

Model-Based RL

Model-Based: Learn a model of the world's state transition function $P(s_{t+1}|s_t, a_t)$ and then use planning through the model to make decisions



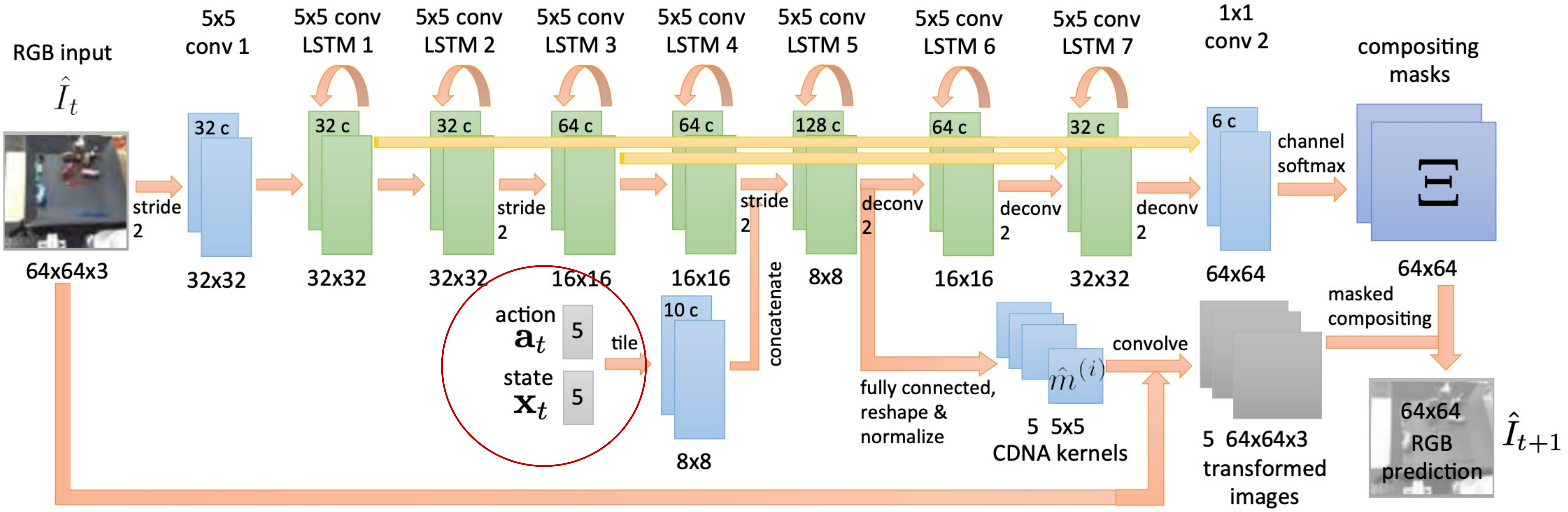
Model might not be accurate enough.

1. Execute the first action
2. Obtain new state
3. Re-optimize the action sequence using gradient descent

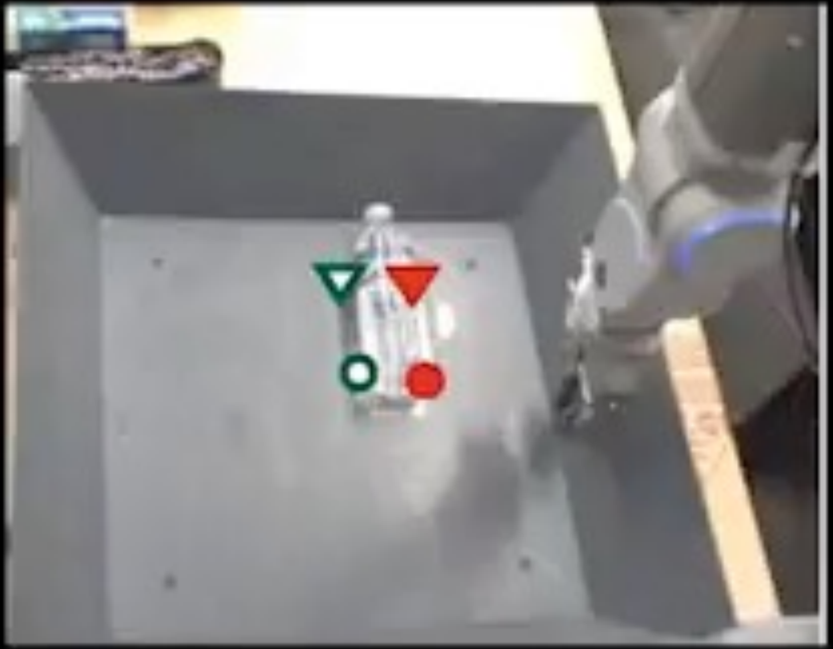
Key: GPU for parallel sampling / gradient descent

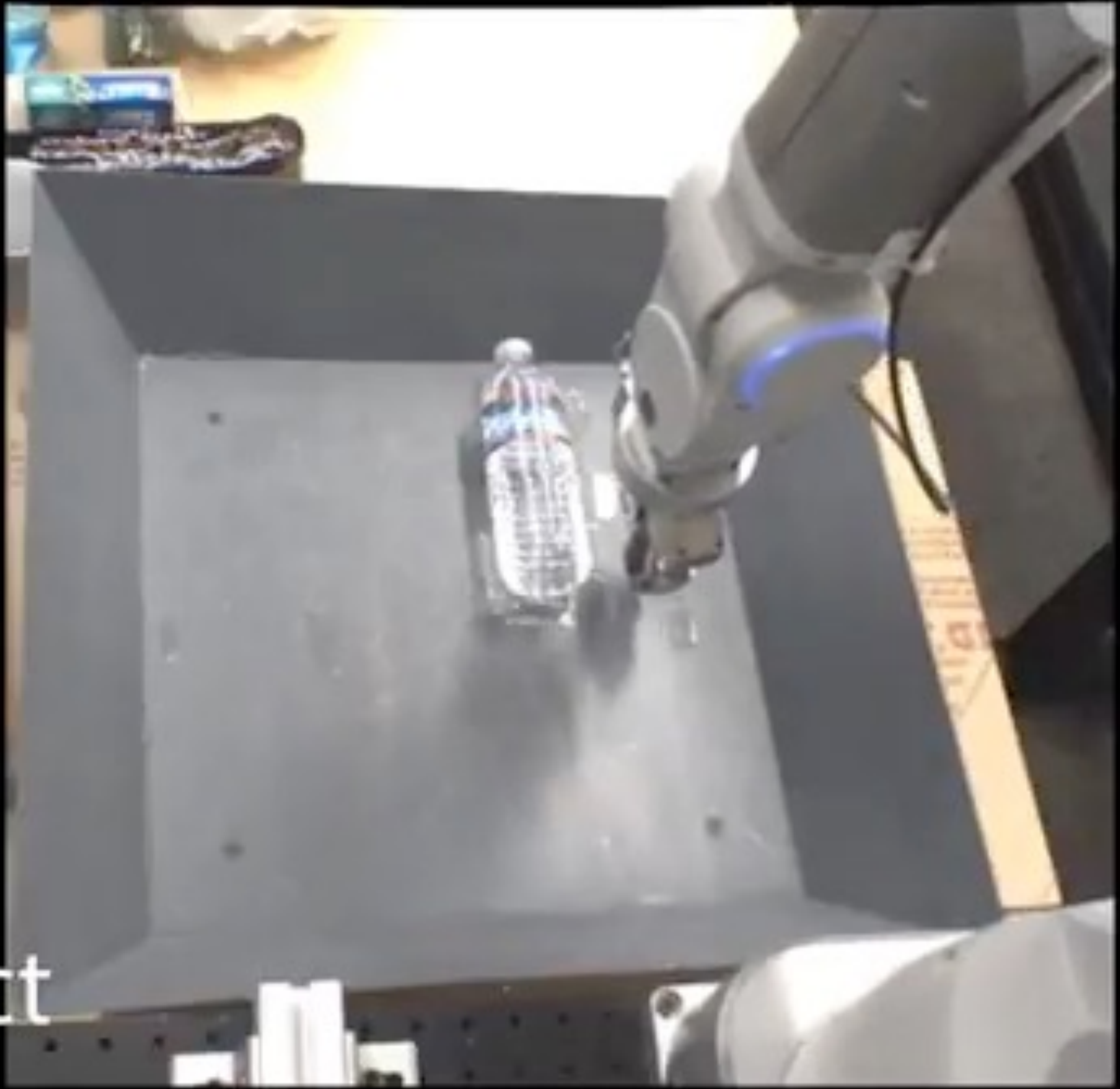
Key question: what should be the form of s_t?

Pixel Dynamics - Deep Visual Foresight



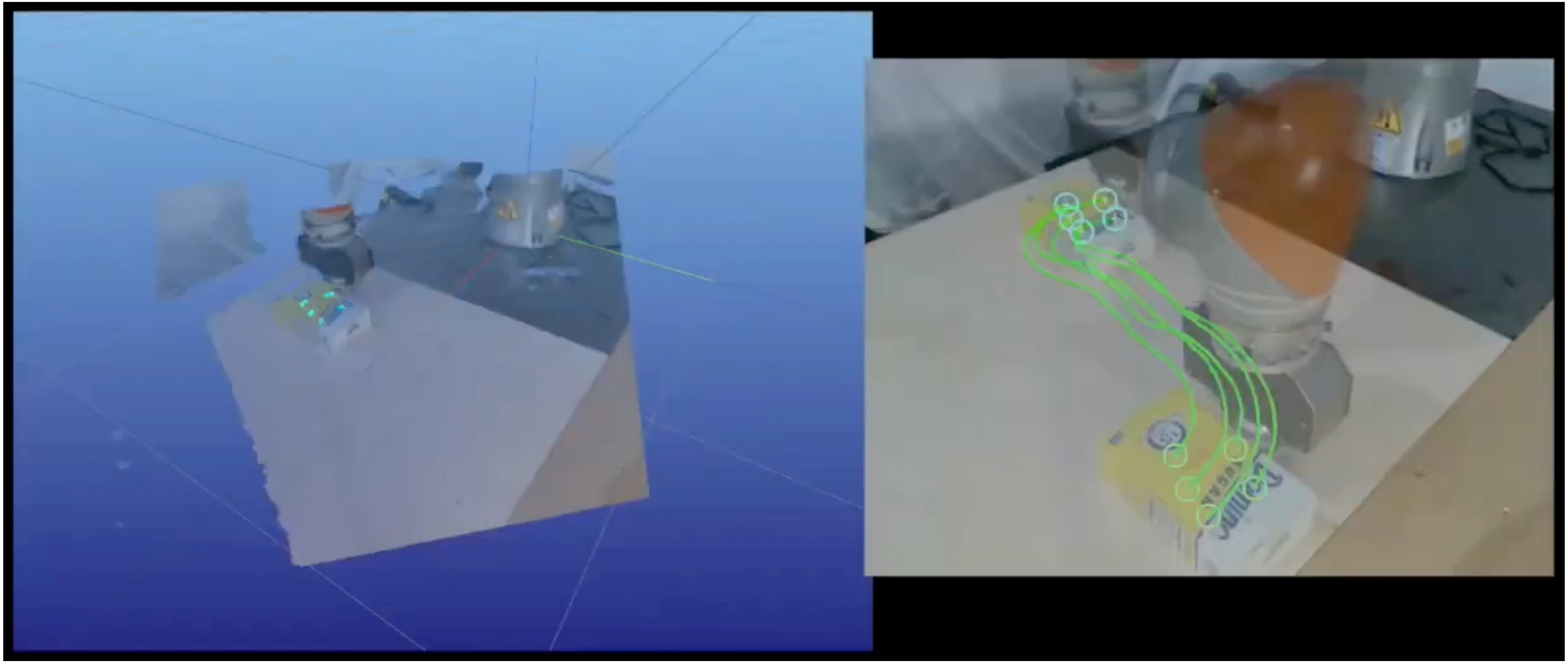
Finn and Levine, "Deep Visual Foresight for Planning Robot Motion", ICRA 2017



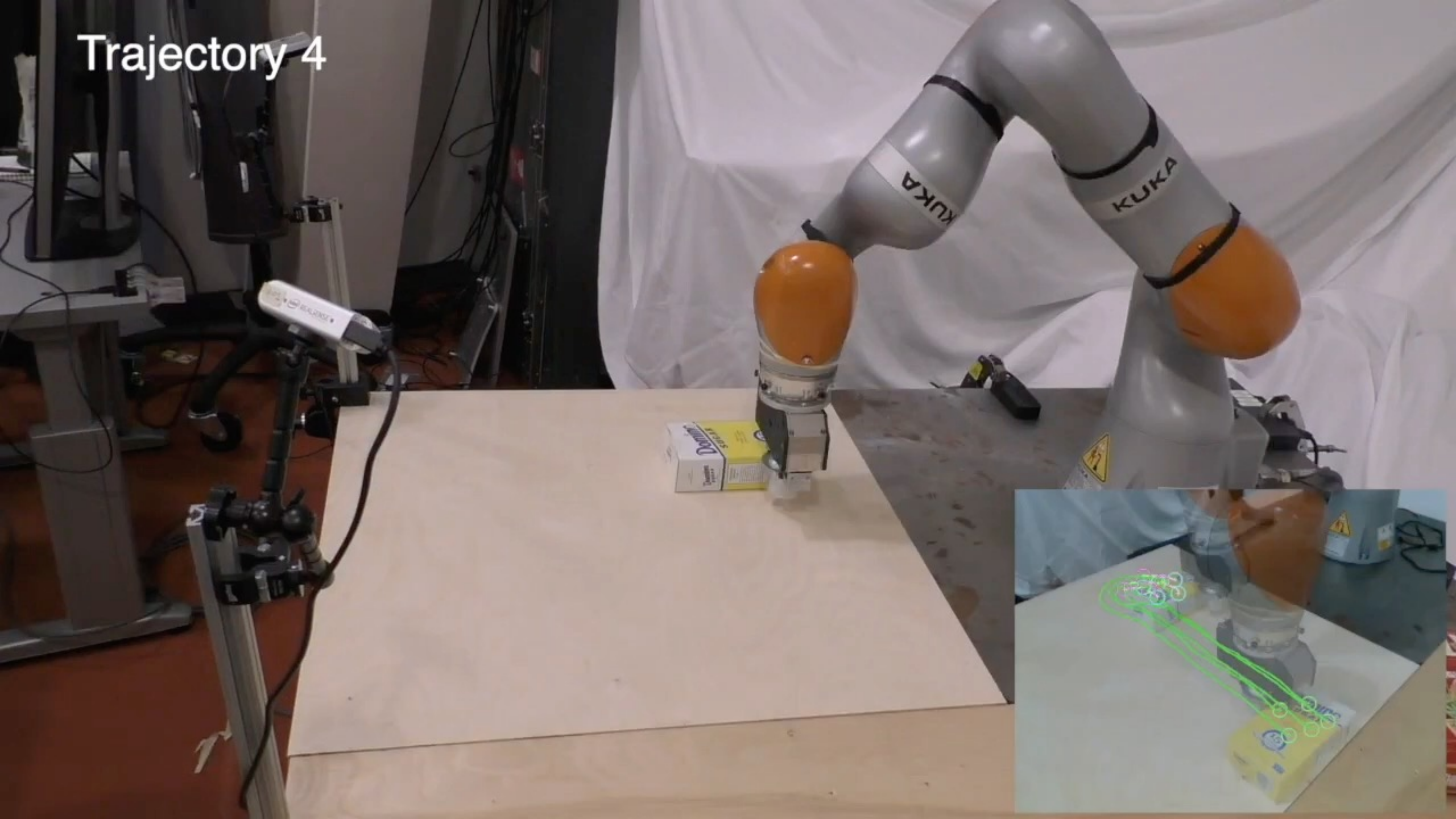


transparent object

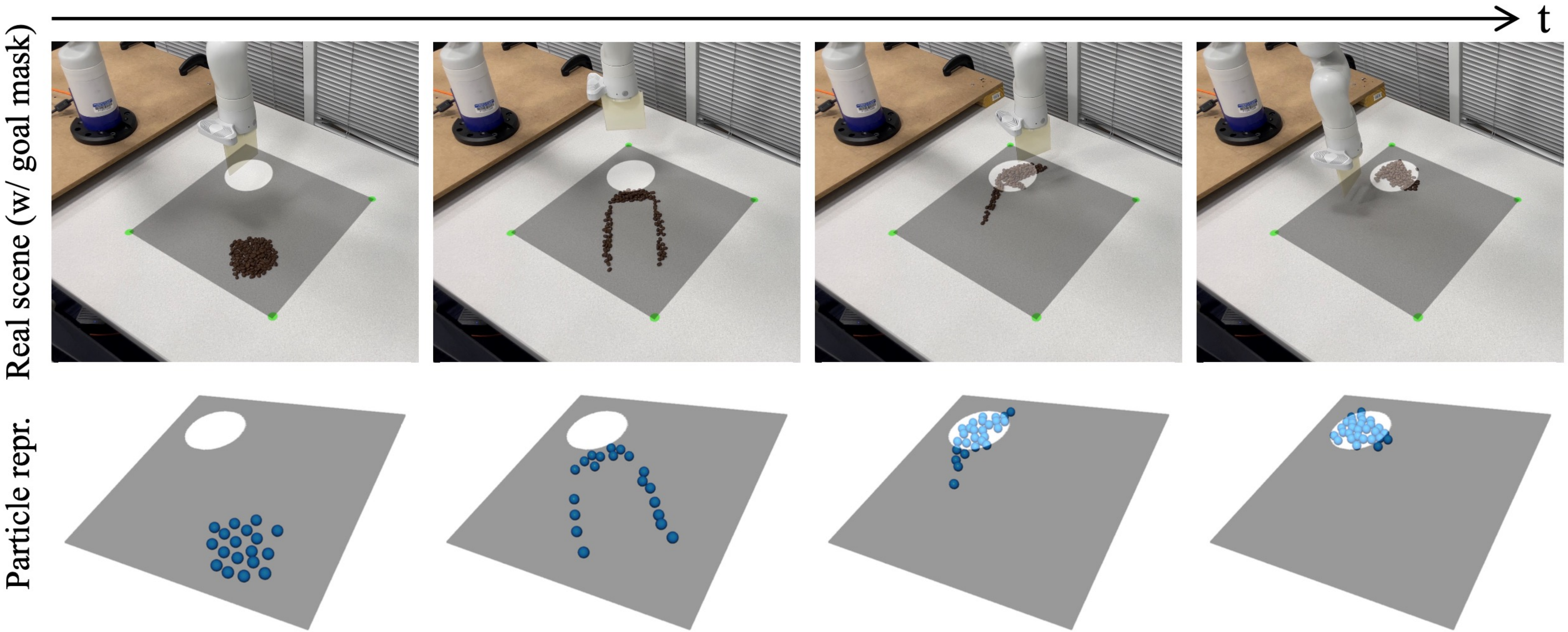
Keypoint Dynamics



Manuelli, Li, Florence, Tedrake, “Keypoints into the Future: Self-Supervised Correspondence in Model-Based Reinforcement Learning”, CoRL 2020

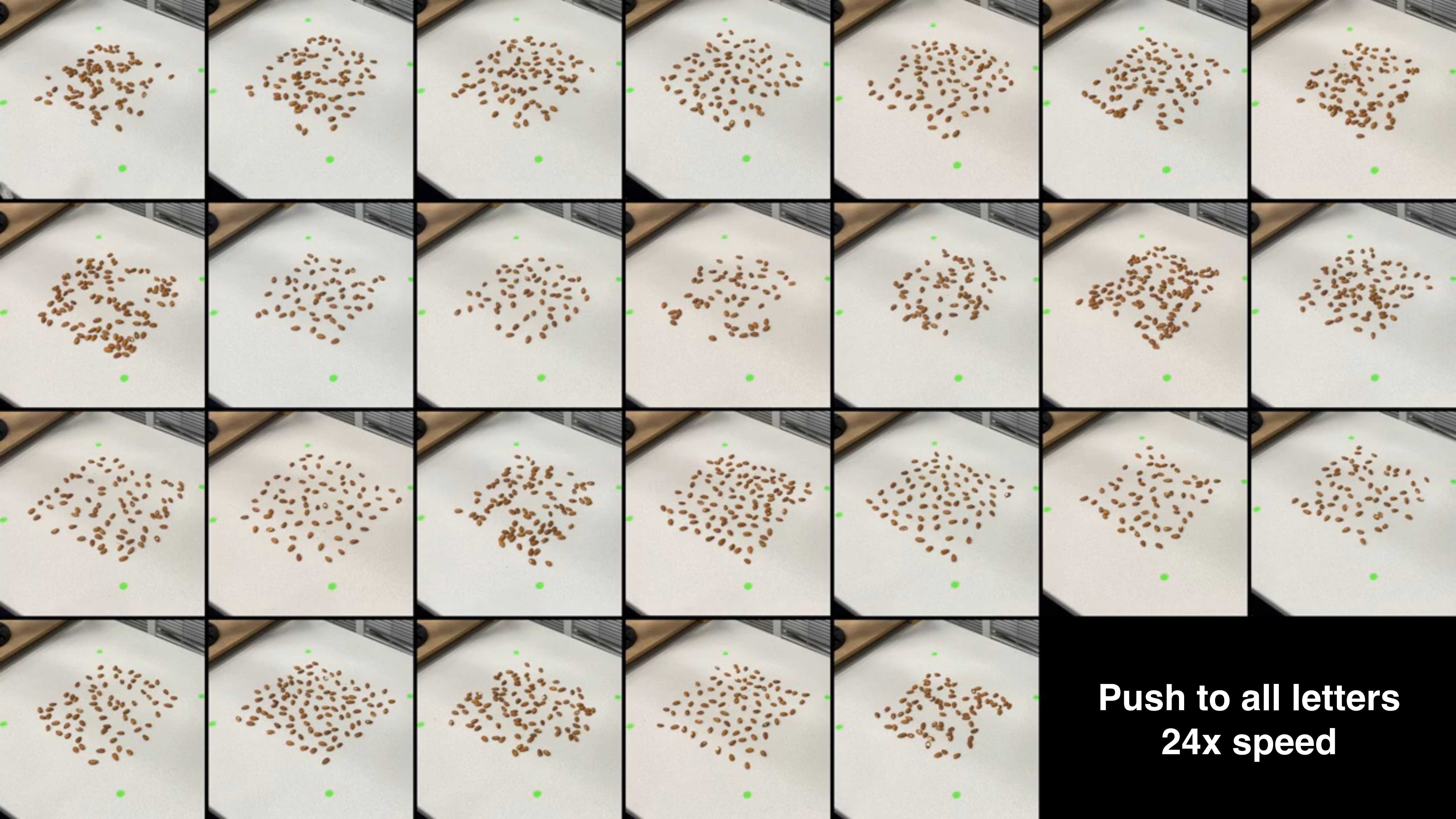

Trajectory 4
KUKA
KUKA
Domino
SUGAR

Particle Dynamics

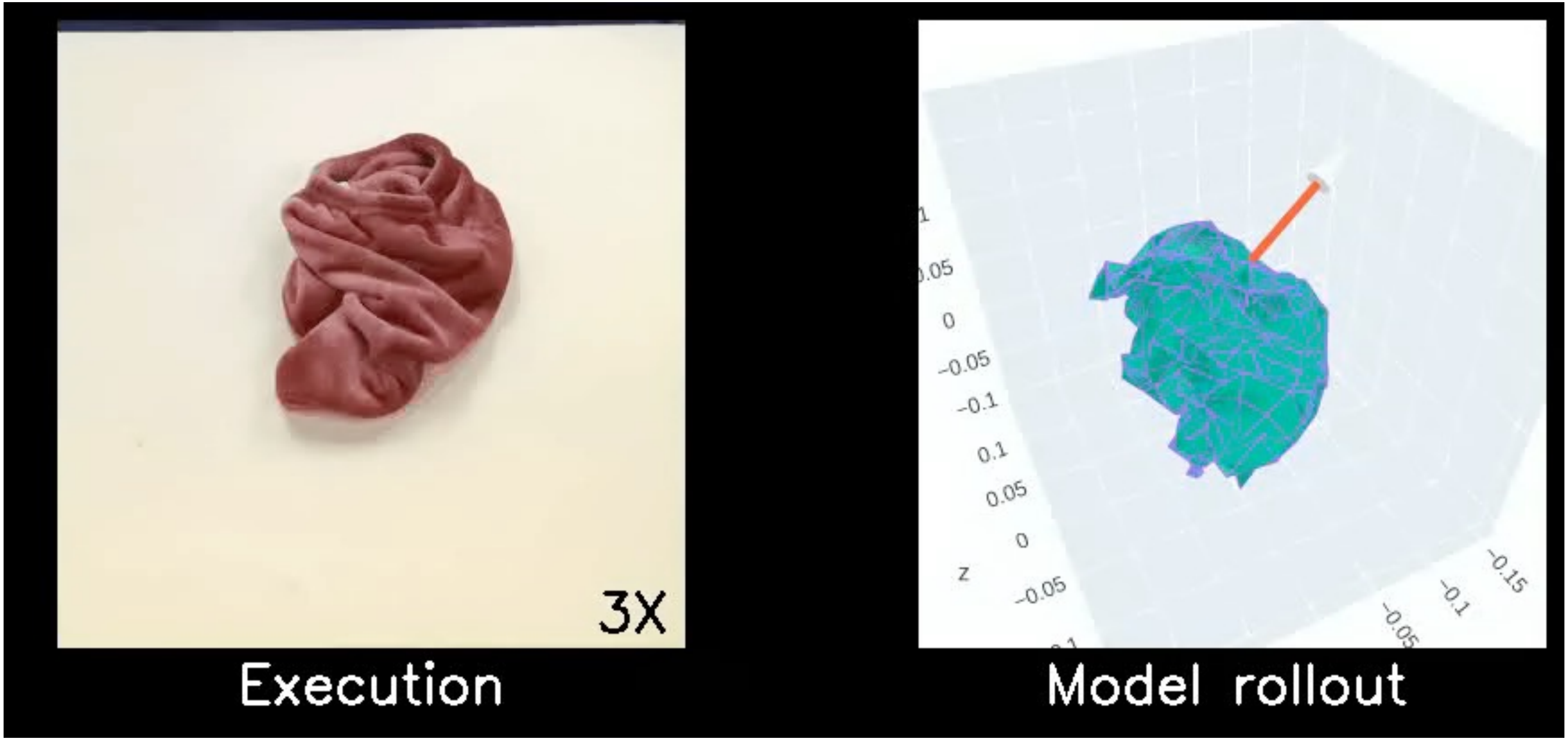


Wang, Li, Driggs-Campbell, Fei-Fei, Wu, “Dynamic-Resolution Model Learning for Object Pile Manipulation”, RSS 2023


Granola
Rice
Carrot
Candy
24x speed


Push to all letters
24x speed

Mesh-Based Dynamics



Huang, Lin, Held, “Mesh-based Dynamics with Occlusion Reasoning for Cloth Manipulation”, RSS 2022

Other approaches

Model-Based: Learn a model of the world's state transition function $P(s_{t+1}|s_t, a_t)$ and then use planning through the model to make decisions

Other approaches

Model-Based: Learn a model of the world's state transition function $P(s_{t+1}|s_t, a_t)$ and then use planning through the model to make decisions

Actor-Critic: Train an actor that predicts actions (like policy gradient) and a critic that predicts the future rewards we get from taking those actions (like Q-Learning)

Sutton and Barto, "Reinforcement Learning: An Introduction", 1998; Degris et al, "Model-free reinforcement learning with continuous action in practice", 2012; Mnih et al, "Asynchronous Methods for Deep Reinforcement Learning", ICML 2016

Other approaches

Model-Based: Learn a model of the world's state transition function $P(s_{t+1}|s_t, a_t)$ and then use planning through the model to make decisions

Actor-Critic: Train an actor that predicts actions (like policy gradient) and a critic that predicts the future rewards we get from taking those actions (like Q-Learning)

Sutton and Barto, "Reinforcement Learning: An Introduction", 1998; Degris et al, "Model-free reinforcement learning with continuous action in practice", 2012; Mnih et al, "Asynchronous Methods for Deep Reinforcement Learning", ICML 2016

Imitation Learning: Gather data about how experts perform in the environment, learn a function to imitate what they do (supervised learning approach)

Other approaches

Model-Based: Learn a model of the world's state transition function $P(s_{t+1}|s_t, a_t)$ and then use planning through the model to make decisions

Actor-Critic: Train an actor that predicts actions (like policy gradient) and a critic that predicts the future rewards we get from taking those actions (like Q-Learning)

Sutton and Barto, "Reinforcement Learning: An Introduction", 1998; Degris et al, "Model-free reinforcement learning with continuous action in practice", 2012; Mnih et al, "Asynchronous Methods for Deep Reinforcement Learning", ICML 2016

Imitation Learning: Gather data about how experts perform in the environment, learn a function to imitate what they do (supervised learning approach)

Inverse Reinforcement Learning: Gather data of experts performing in environment; learn a reward function that they seem to be optimizing, then use RL on that reward function

Ng et al, "Algorithms for Inverse Reinforcement Learning", ICML 2000

Other approaches

Model-Based: Learn a model of the world's state transition function $P(s_{t+1}|s_t, a_t)$ and then use planning through the model to make decisions

Actor-Critic: Train an actor that predicts actions (like policy gradient) and a critic that predicts the future rewards we get from taking those actions (like Q-Learning)

Sutton and Barto, "Reinforcement Learning: An Introduction", 1998; Degris et al, "Model-free reinforcement learning with continuous action in practice", 2012; Mnih et al, "Asynchronous Methods for Deep Reinforcement Learning", ICML 2016

Imitation Learning: Gather data about how experts perform in the environment, learn a function to imitate what they do (supervised learning approach)

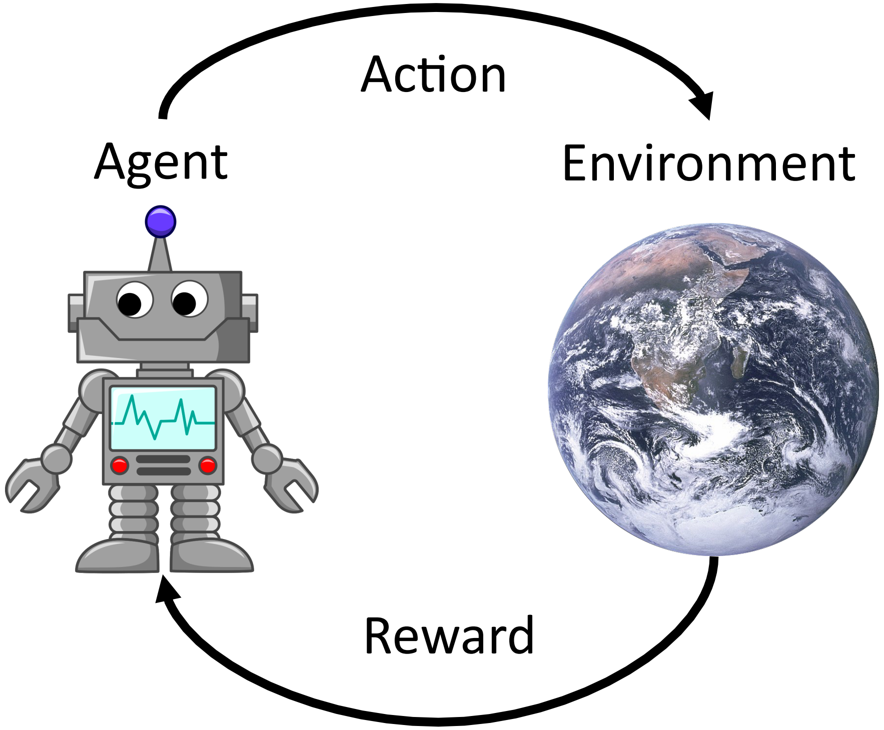
Inverse Reinforcement Learning: Gather data of experts performing in environment; learn a reward function that they seem to be optimizing, then use RL on that reward function

Ng et al, "Algorithms for Inverse Reinforcement Learning", ICML 2000

Adversarial Learning: Learn to fool a discriminator that classifies actions as real/fake

Ho and Ermon, "Generative Adversarial Imitation Learning", NeurIPS 2016

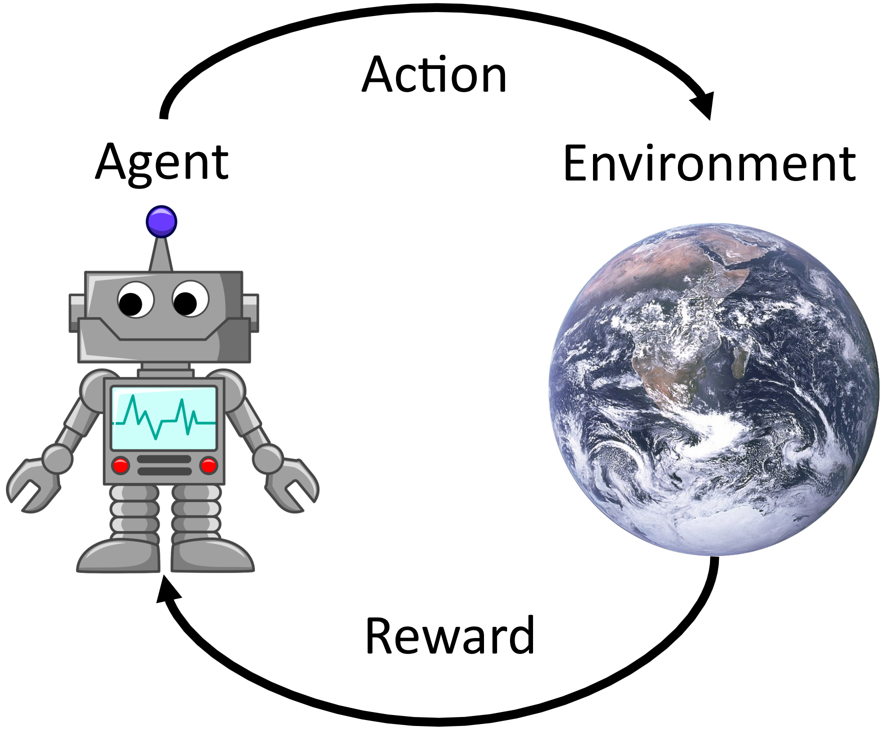
Reinforcement Learning: Interacting With World



Normally we use RL to train **agents** that interact with a (noisy, nondifferentiable) **environment**

Summary: Reinforcement Learning

RL trains **agents** that interact with an **environment** and learn to maximize **reward**



Q-Learning: Train network $Q_\theta(s, a)$ to estimate future rewards for every (state, action) pair. Use <u>Bellman Equation</u> to define loss function for training Q

Policy Gradients: Train a network $\pi_\theta(a \mid s)$ that takes state as input, gives distribution over which action to take in that state. Use <u>REINFORCE Rule</u> for computing gradients

Active research problems in robot learning

What tasks do we work on?

How to get training data (sim)?

How to get large-scale diverse data (real)?

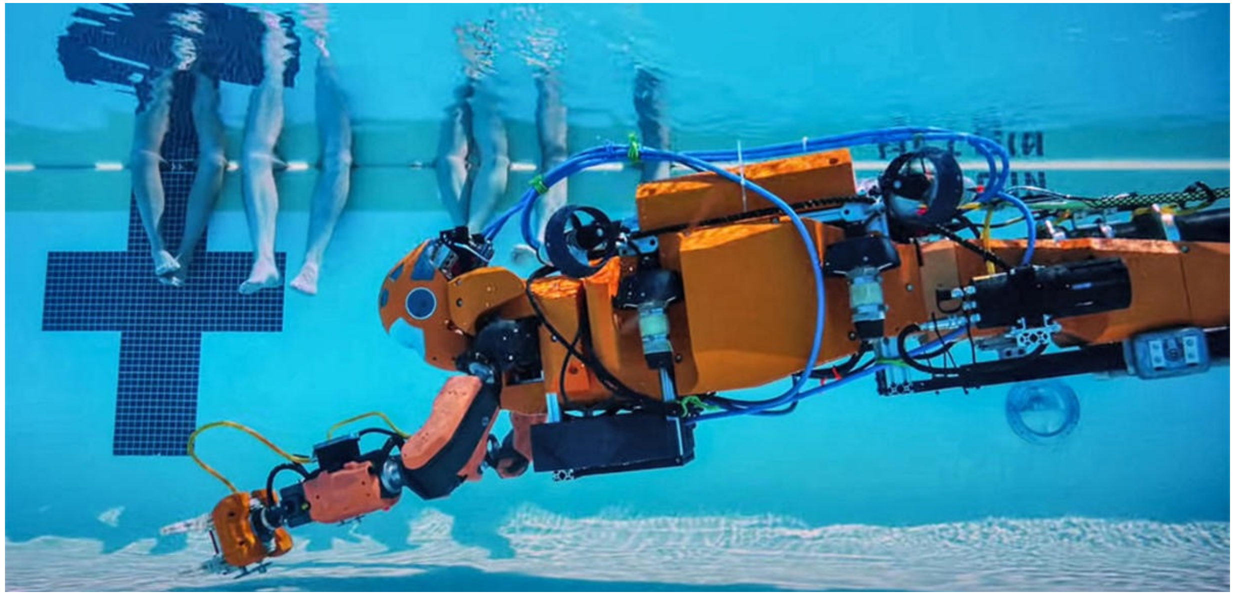
How to achieve successful sim2real transfer?

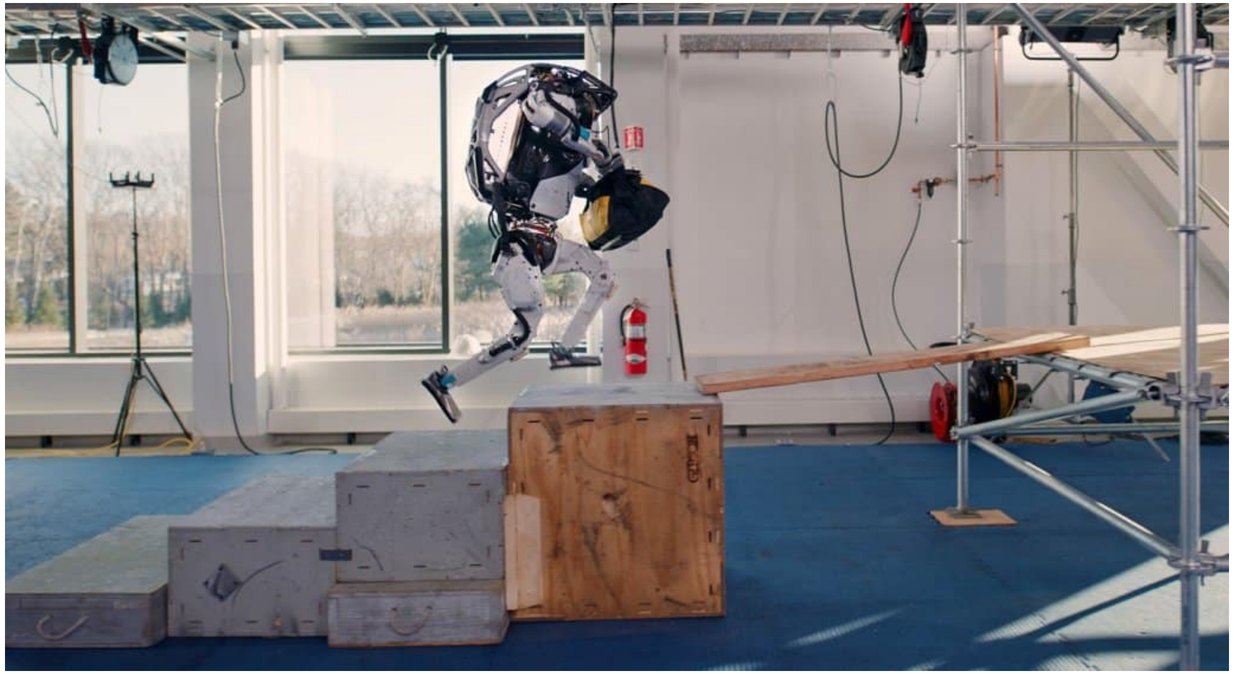
How to interact with humans?

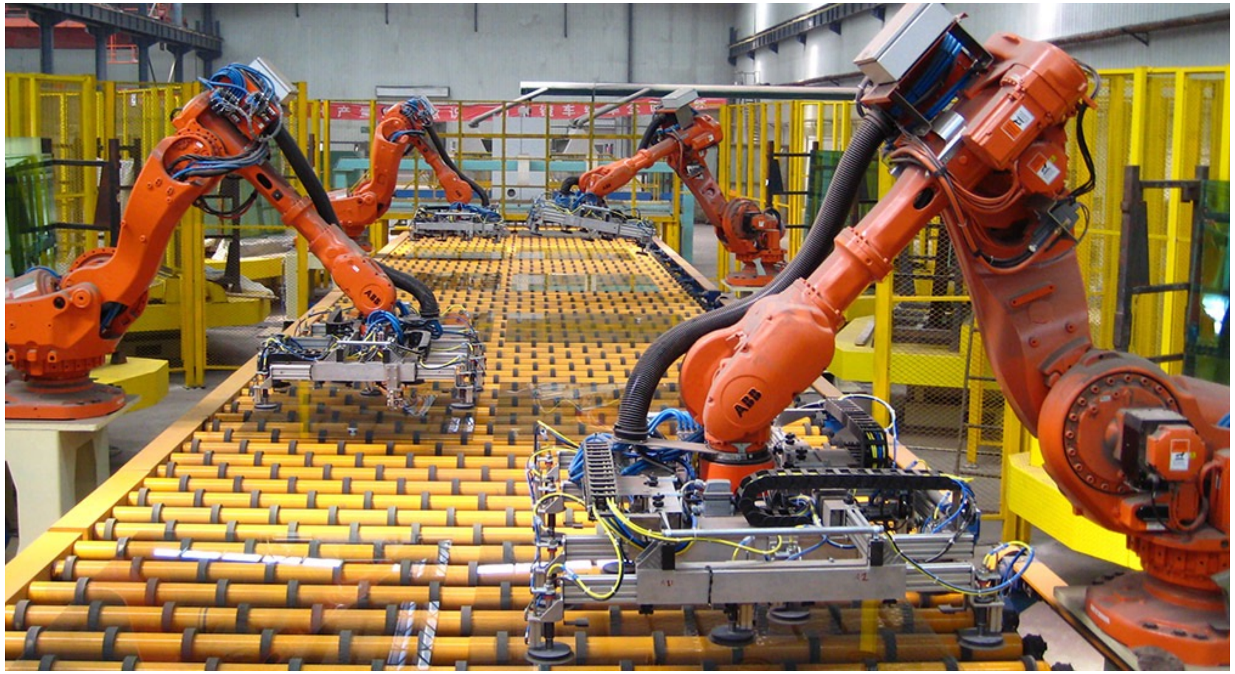
…

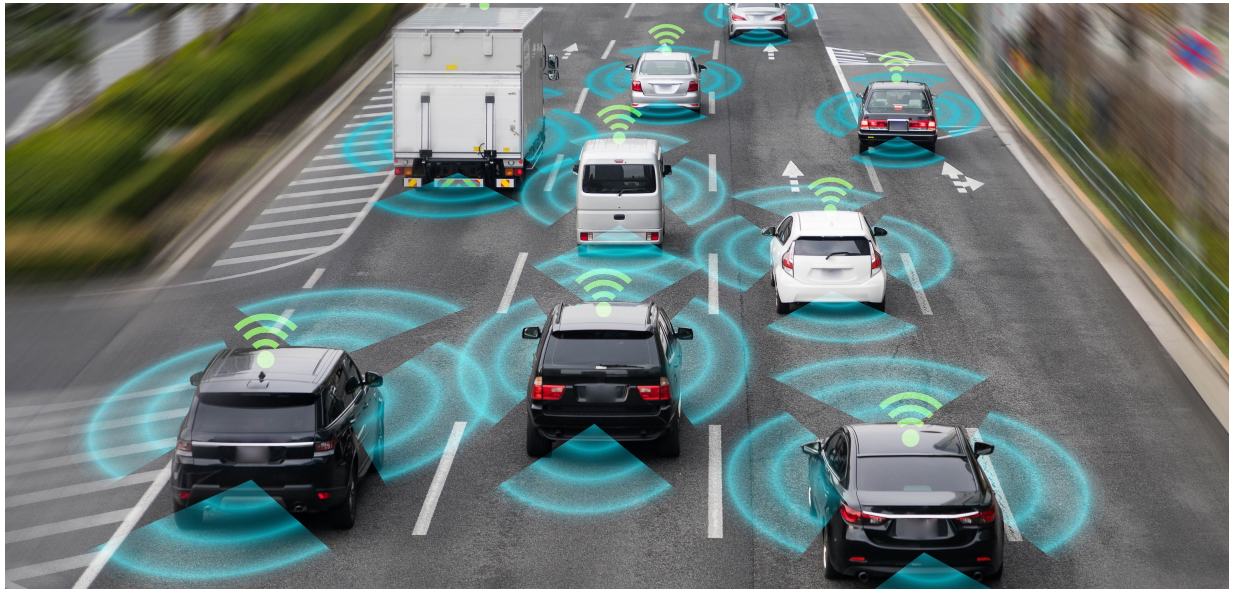
tasks that matter

What would you like a robot to help you with?





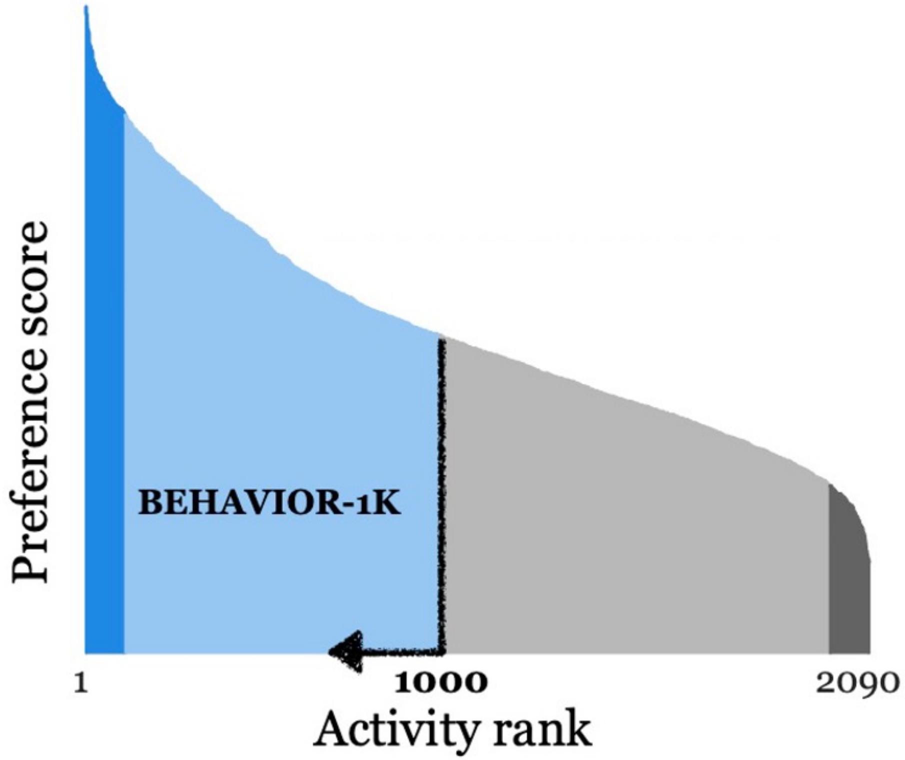




What do people actually want robots to do?

"How much would you benefit if a robot did this for you?"

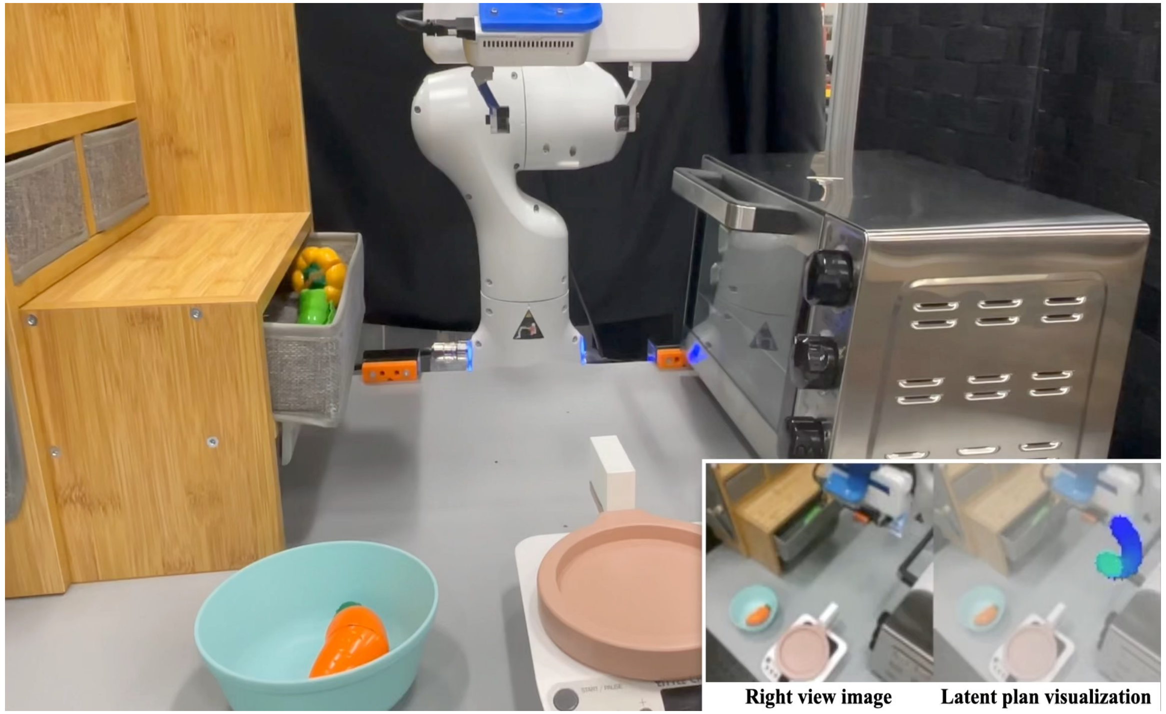
1. Wash floor
2. Clean bathrooms
3. Clean after a wild party
4. Clean floors
5. Mop floors



2086. Throw darts
2087. Mix baby cereal
2088. Buy a ring
2089. Playing squash
2090. Opening presents

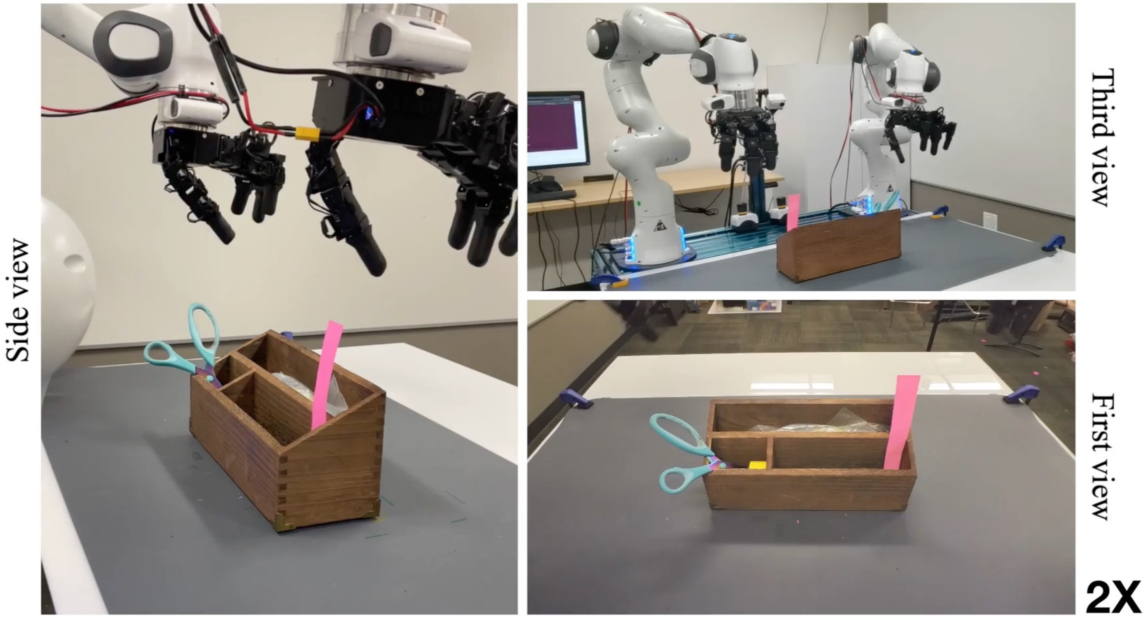

BEHAVIOR
simulating and benchmarking robot tasks that matter to humans
Stanford Vision and Learning Lab

Scalable human data collection for robot learning



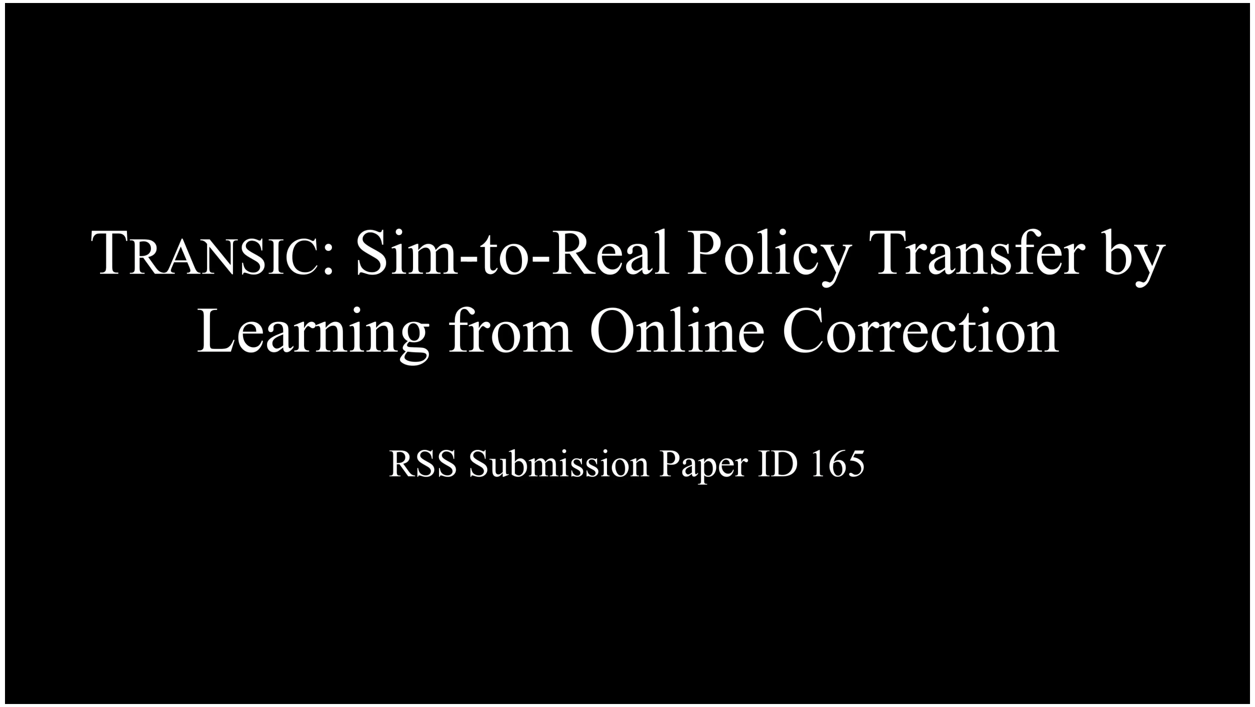
MimicPlay
Wang et al., CoRL 2023

Scalable human data collection for robot learning



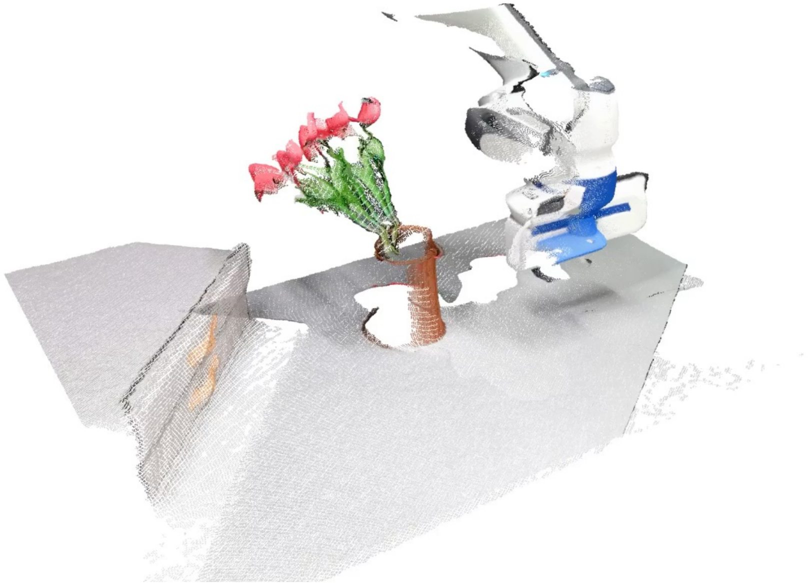
DexCap
Wang et al., RSS 2024

Sim2Real transfer of RL policy



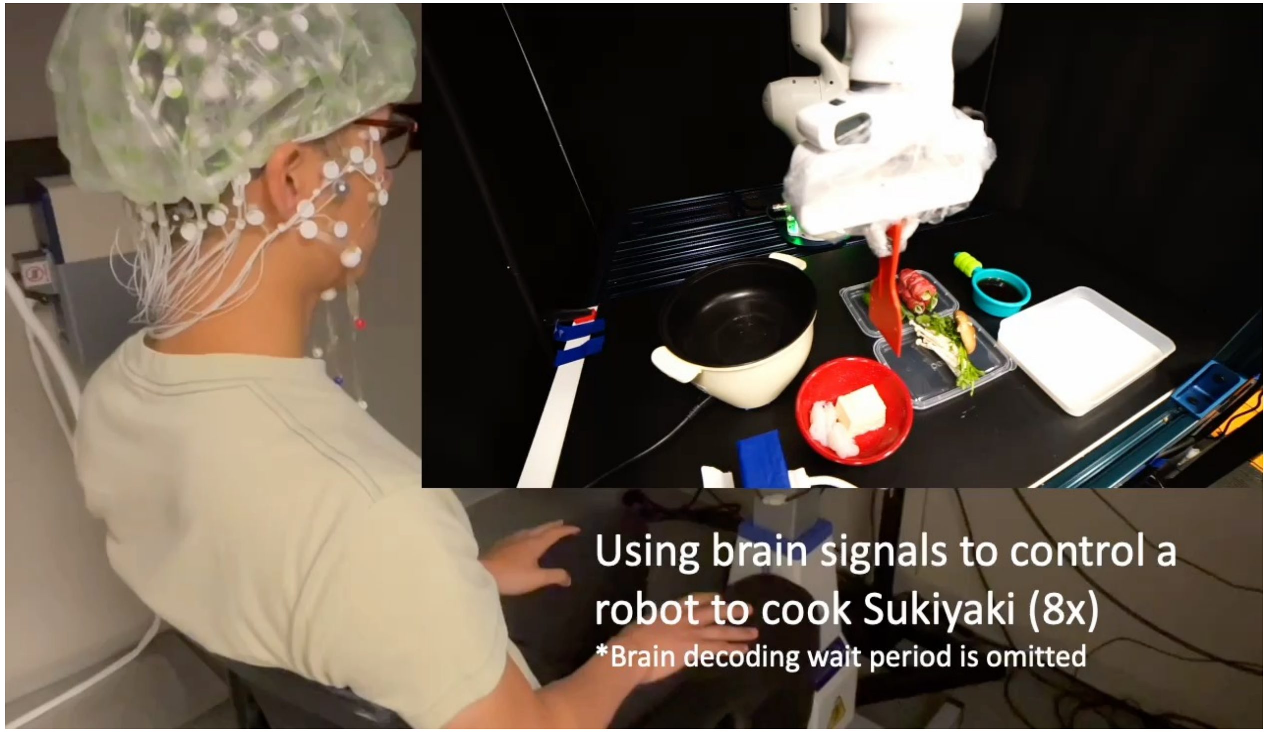
TRANSIC
Jiang et al., arxiv 2024

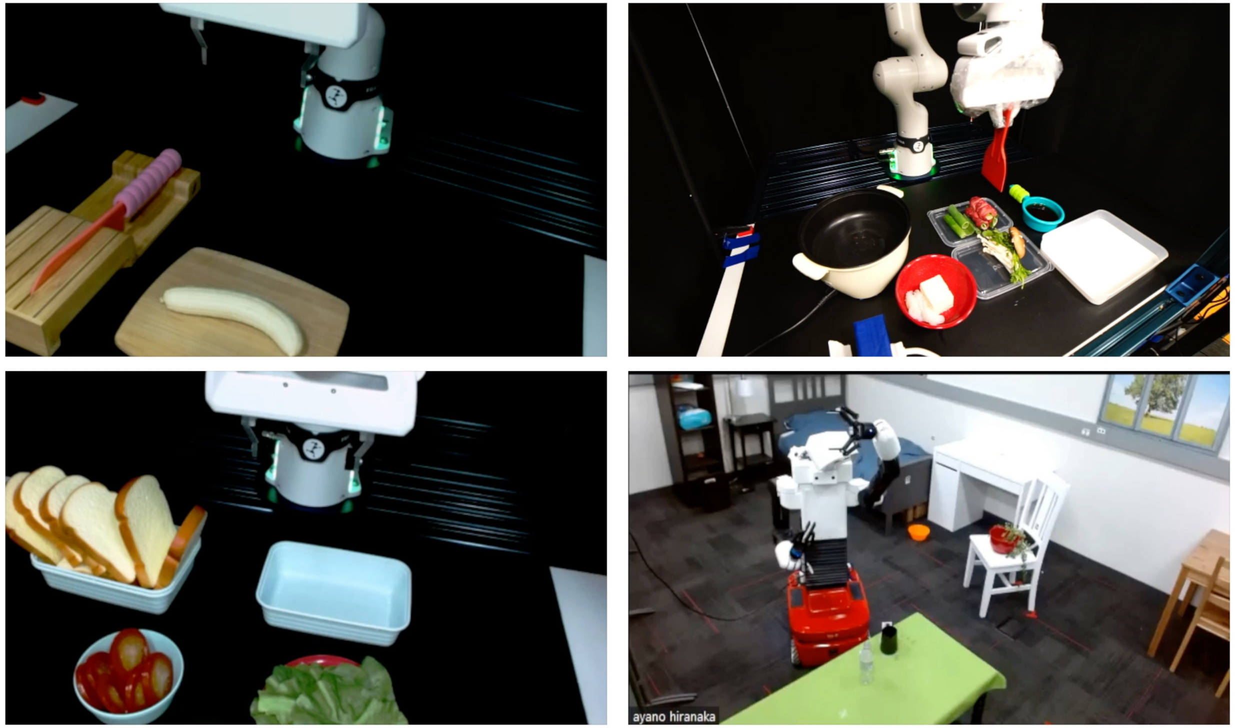
Human-robot interaction through language



VoxPoser
Huang et al., CoRL 2023

Human-robot interaction through brain signals





NOIR
Zhang et al., CoRL 2023

Next time: **Human-Centered AI (Fei-Fei)**