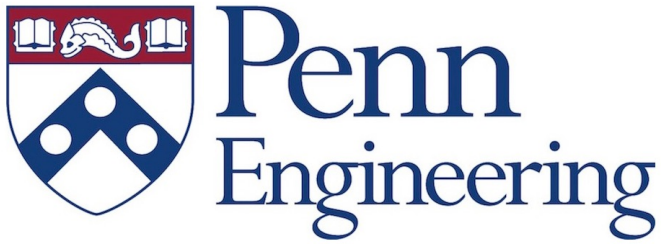
# CIS 7000-04 / ESE 6800 Spring 2025: Intro To Imitation and Reinforcement Learning

Tue, Jan 21, 2025

Instructor: Dinesh Jayaraman

Assistant Professor, CIS



# Module Goals (about 3 lectures)

**We assume that:**

- You understand machine learning at the level of an introductory class e.g. CIS 5190
- You don’t know much about imitation and reinforcement learning
- You know a little bit about robots

**At the end, you will know:**

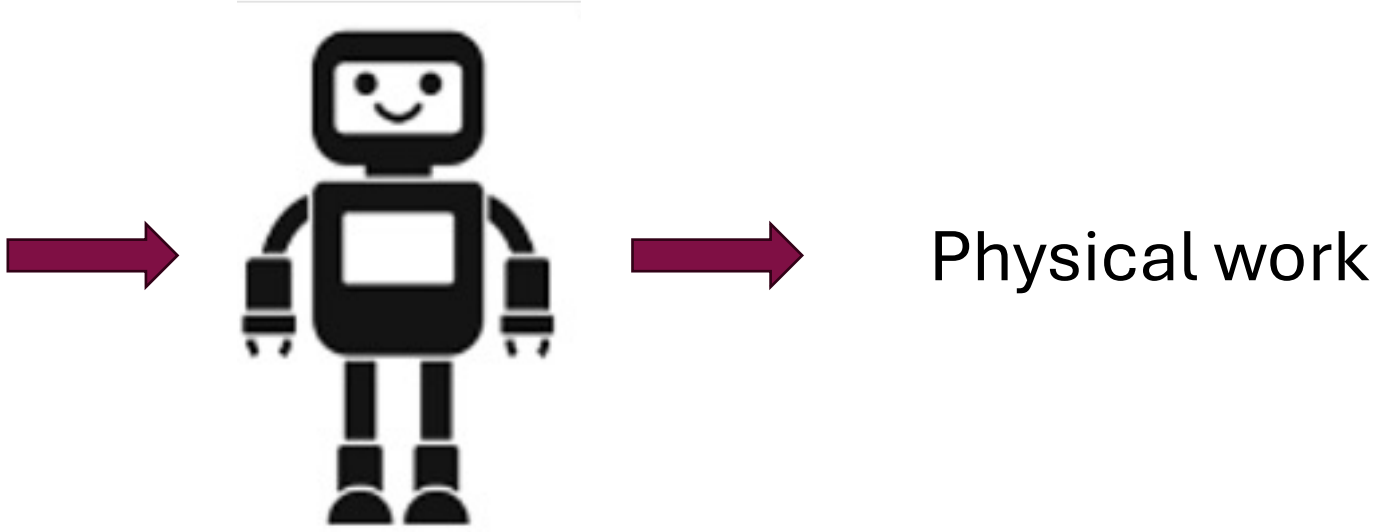
- How to cast the problem of robotic control as a sequential decision making problem
- What the key ideas of imitation learning and reinforcement learning are, and sketches of some basic methods for each
- Broad landscape of imitation and RL, and names and self-study references for common approaches

# Plan

- What Does Robot Control Look Like? Perception-Action Loops
- How Markov Decision Processes Model Robot Control
- What If Dynamics and Reward are Unknown
  - Imitation Learning
  - Reinforcement Learning
    - Policy Gradients
    - Q Learning, DDPG …
    - Model-based RL (stretch goal)

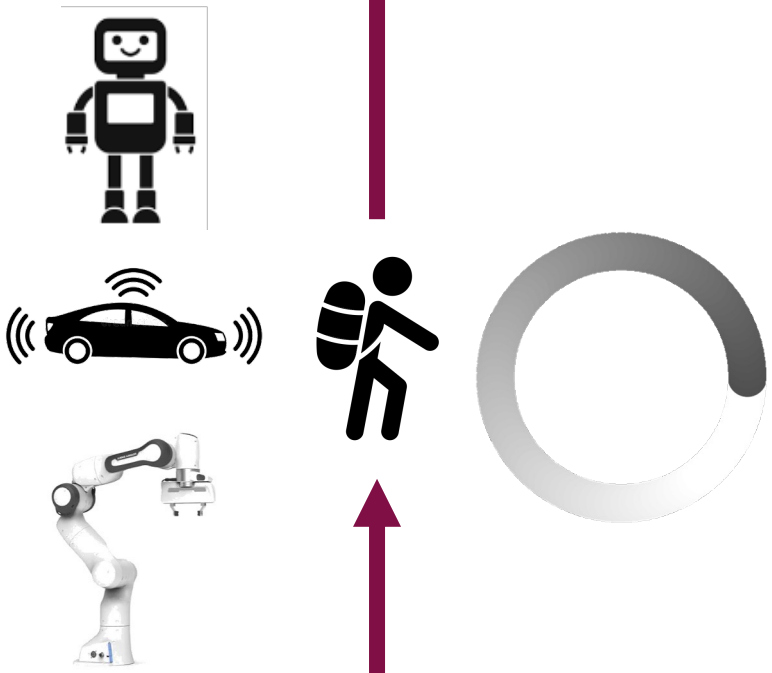
# Robots: Information → Physical Work

Information about the world



Physical work

# (Robotic) Agent ↔ World: Perception & Action

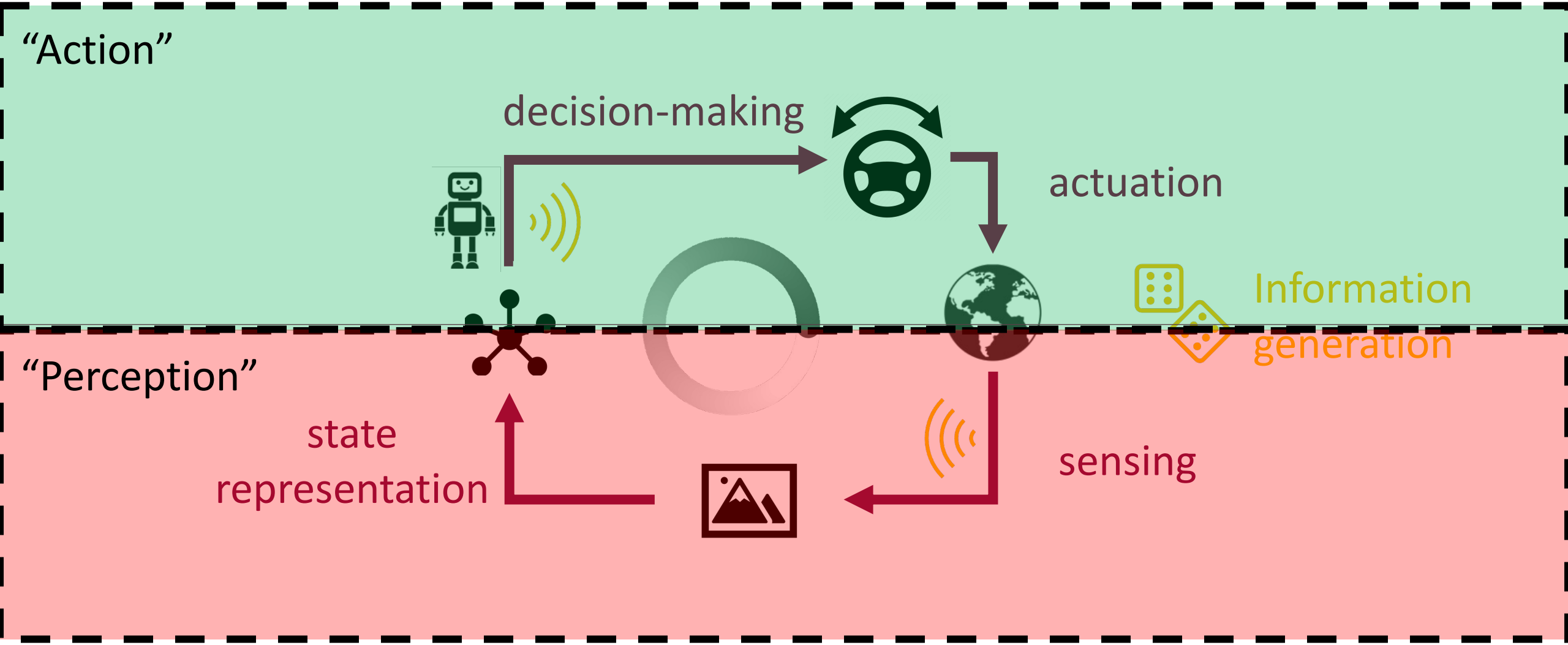


action

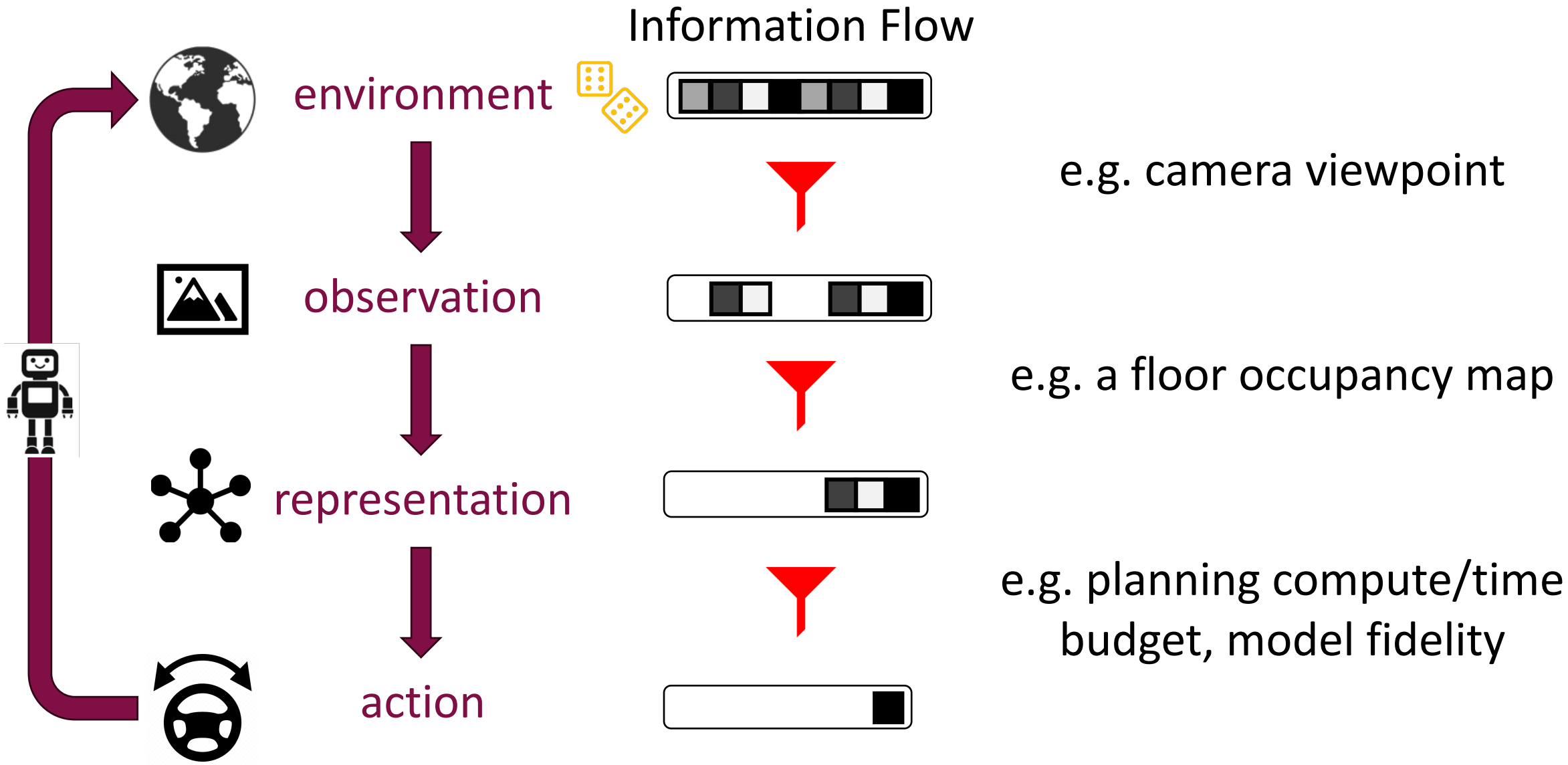
perception

Information generation

# Zooming Into Perception-Action Loops



# Selective Processing of Information

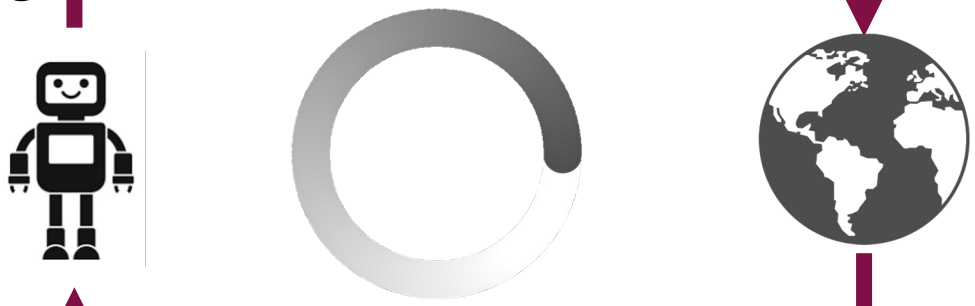


# The Perception-Action Loop ("Fully Observed")

action

… decides how to move

… actually moves, influencing the environment to evolve.



Agent

Environment

… "sees" the state $s_t$

… is in one of several "states"

$s_t \in S$

perception

… evolves into a new "state"

$s_{t+1} \in S$

# What Robots Can Learn From Data

- **Policies:** mappings from perceived state $s_t$ to action commands $a_t$
- **Dynamics models:** models of how agent actions influence the evolution of the environment state
- **Reward functions:** a score indicating how well the robot is performing a task.

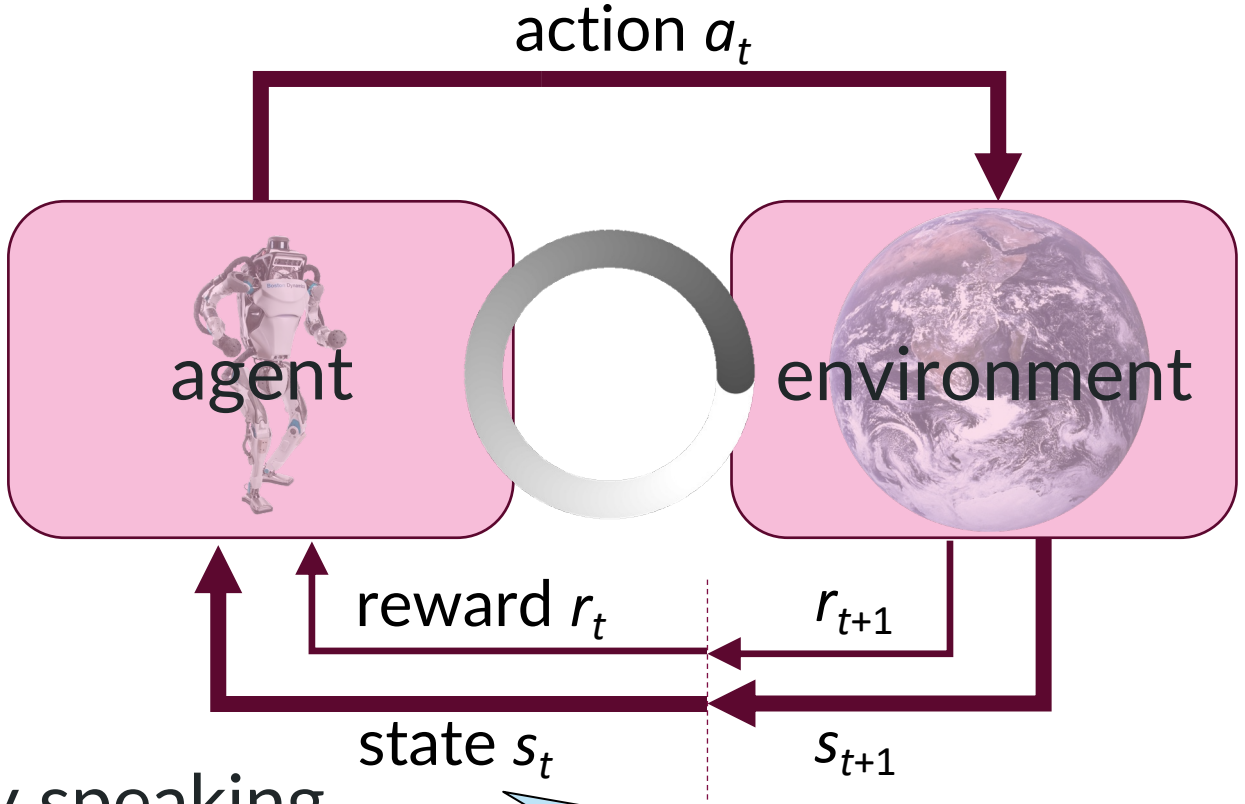
- **State Representations:** an encoding of raw sensory inputs
- **What to Sense**

- **“Common-Sense Knowledge”**

# Casting Robot Control As A Sequential Decision Making Problem

# Markov Decision Process Formulation of Control

- Agent receives observations (state $s_t \in S$) and feedback (reward $r_t$) from the world
- Agent takes action $a_t \in A$
- Agent receives updated state $s_{t+1}$ and reward $r_{t+1}$




- Agent's goal is to maximize, loosely speaking, "expected rewards in the future".

Image: https://robots.ieee.org/robots/pr2/

# MDP Formulation: Notations

An MDP $(S, A, P, R, \gamma)$ is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function or "dynamics model" $P(s' \mid s, a)$
  - Probability $P(s' \mid s, a)$ that $a$ from $s$ leads to $s'$
- Reward function $r_t = R(s, a, s')$
- Discount factor $\gamma < 1$, expressing how much we care about the future (vs. immediate rewards)
- "utility" = *discounted* future reward sum $\sum_t \gamma^t r_{t+1}$
- Goal: maximize *expected* utility

Example

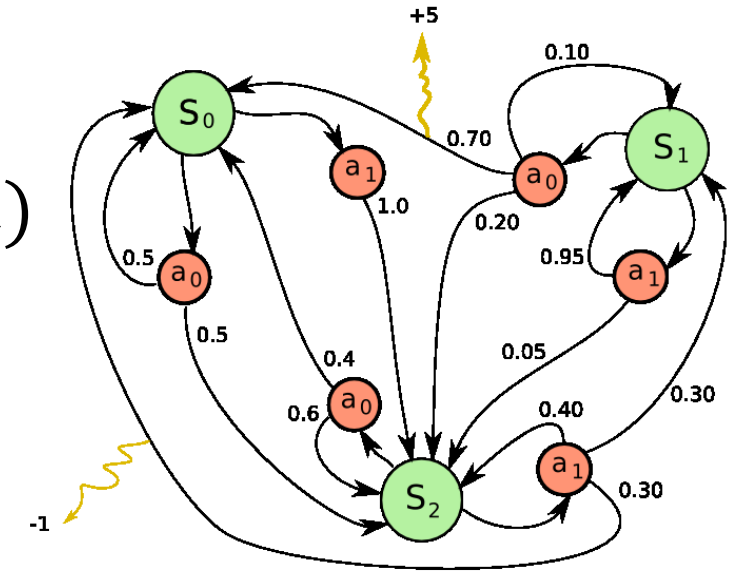


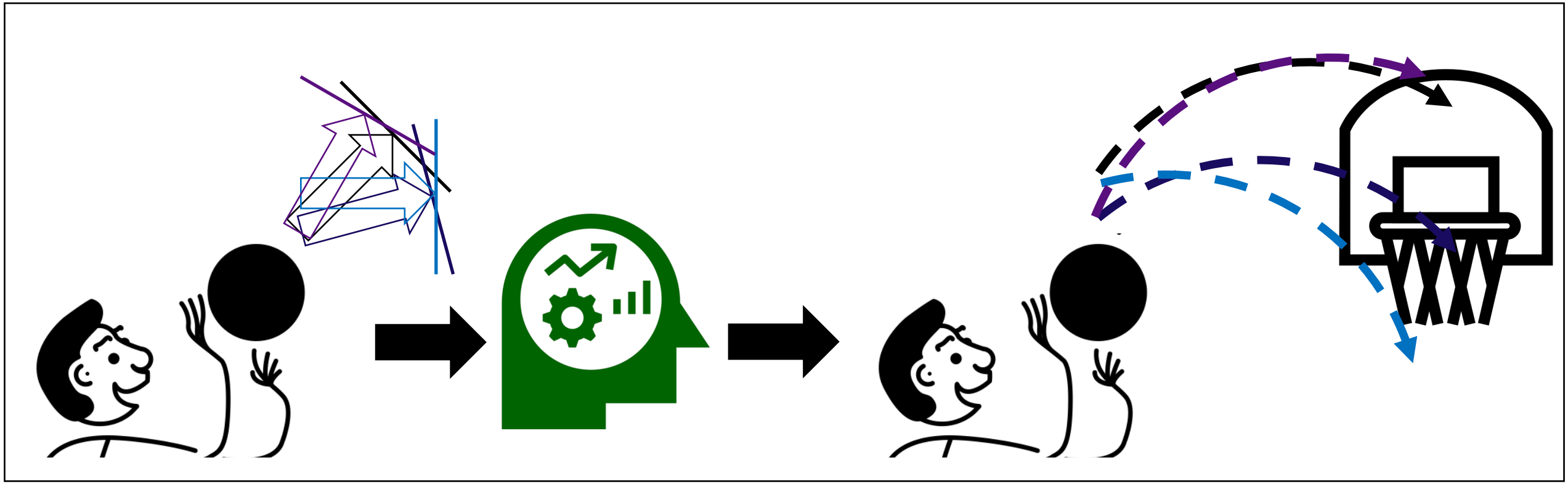

Image: https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

# Robots Make Sequential Decisions Too!

Must make a sequence of decisions to
maximize some success measure/"reward",
which is a cumulative effect of the full sequence.


| | | |
|---|---|---|
| **Actions $a_t$:** | muscle contractions | motor current or torque |
| **Observations $s_t$:** | sight, smell | camera images |
| **Reward $r_t$:** | food | average speed |

# Traditional Model-Based Planning (an example)



1. Sample actions and forecast their effects using $P(s'|s,a)$.
2. Select action with best forecasted outcome $\sum_t \gamma^t r_{t+1}$.

If you can predict how the environment will evolve in response to an action (and score those predictions), you can select good actions.

# Traditional Model-Based Controllers

- Broadly, traditional approaches rely on having near-accurate predictive "models" of the environment. E.g. LQR (linear quadratic regulators), MPC (model predictive control), H-infinity control, Policy Iteration (dynamic programming) …

- The quality of the synthesized controller depends directly on the quality of the environment model.

# Towards Imitation and Reinforcement Learning

- But in practice, it is too strong of an assumption that $\boldsymbol{P}(.)$ and $\boldsymbol{R}(.)$ are known in advance.

- If we assume no knowledge of $P(.)$ and $R(.)$, we are in the position of an agent dropped into a completely unknown environment with a completely unknown task. How could this even work?

Sidenote: In real problems, we have *some* knowledge of $P(.)$ and $R(.)$, but this assumption helps study algorithms for the most general setting.

# Towards Imitation and Reinforcement Learning

- **How might learning even work with unknown dynamics and rewards?**
  - **Reinforcement Learning:**
    - No idea about either $P(.)$ or $R(.)$ at the start of training, but we do have the ability to try things out in the world, each time experiencing the effects of $P(.)$ and $R(.)$.
    - Key Unique Issues:
      - Exploration: How to acquire useful experience to select a good policy?
      - Credit Assignment: How to figure out which parts of experience were "good" (likely to lead to good outcomes) and which were "bad"?
    - More on these later
  - **Imitation Learning:**
    - No idea about either $P(.)$ or $R(.)$, but largely circumvents above RL issues by having a teacher show us the way!
    - Rather than figure out both how to gather experiences and how to optimize policies based on them, we *only* optimize policies.

# Side Note 1: **Partially Observed MDPs (POMDPs)**

- Most robotics problems are *partially observed* i.e., you cannot observe the full Markov state (or you can only observe it noisily).
  - E.g. an autonomous vehicle cannot see a pedestrian beyond a neighbor car.

- Handling this correctly requires a change to the basic MDP formulation; we will not cover this in this tutorial, but instead return to it as our readings during the course require.

Reference: Lauri et al, Partially Observable Markov Decision Processes in Robotics: A Survey | IEEE Journals & Magazine | IEEE Xplore. T-RO 2022.

# Side Note 2: Other Assumptions in the MDP framework

- Time discretization -> continuous-time MDPs etc.
- Instantaneous $s_t, a_t$ at time $t$. -> real-time MDP, delay-aware MDPs etc.
- Simple scalar rewards -> constrained MDPs etc.

…

# Imitation Learning Through Behavior Cloning

Solving sequential decision making problems with supervised learning!

# Which year is this from?

***"This review investigates two recent developments in artificial intelligence and neural computation: learning from imitation and the development of humanoid robots. It will be postulated that the study of imitation learning offers a promising route to gain new insights into mechanisms of perceptual motor control that could ultimately lead to the creation of autonomous humanoid robots."***

Schaal, S (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences* **3**:233-242.

# Is Imitation Learning the Route to Humanoid Robots?

**Stefan Schaal**

sschaal@usc.edu
http://www-slab.usc.edu/sschaal

Computer Science and Neuroscience, HNB-103, University of Southern California, Los Angeles, CA 90089-2520
Kawato Dynamic Brain Project (ERATO/JST), 2-2 Hikaridai, Seika-cho, Soraku-gun, 619-02 Kyoto, Japan

**Summary**

This review investigates two recent developments in artificial intelligence and neural computation: learning from imitation and the development of humanoid robots. It will be postulated that the study of imitation learning offers a promising route to gain new insights into mechanisms of perceptual motor control that could ultimately lead to the creation of autonomous humanoid robots. Imitation learning focuses on three important

# Robots then

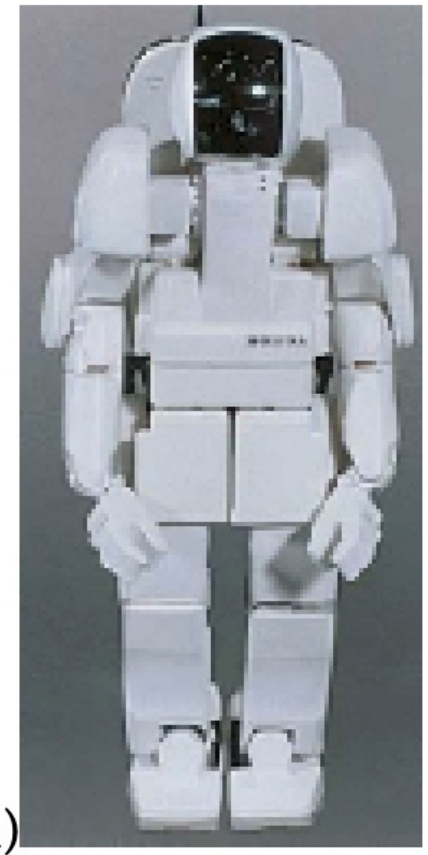
a)  b) 

Figure 1: Honda Humanoid Robot in a) frontal, and b) side view.

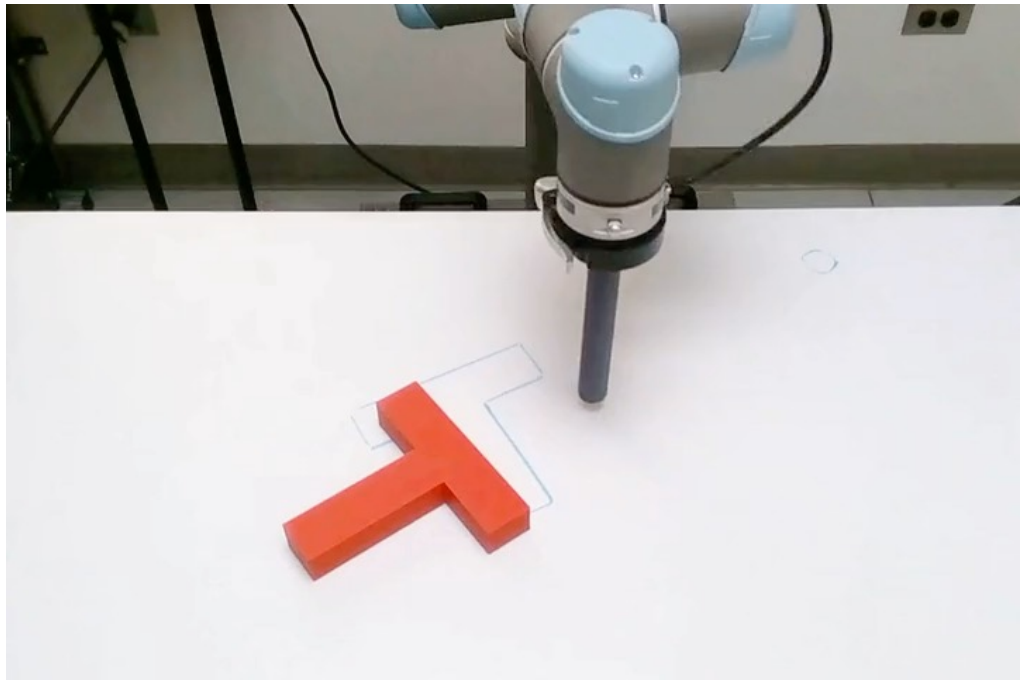
# Fast forward to 2024


Video from Figure.ai

Video from Tesla

# Exciting progress on imitation learning

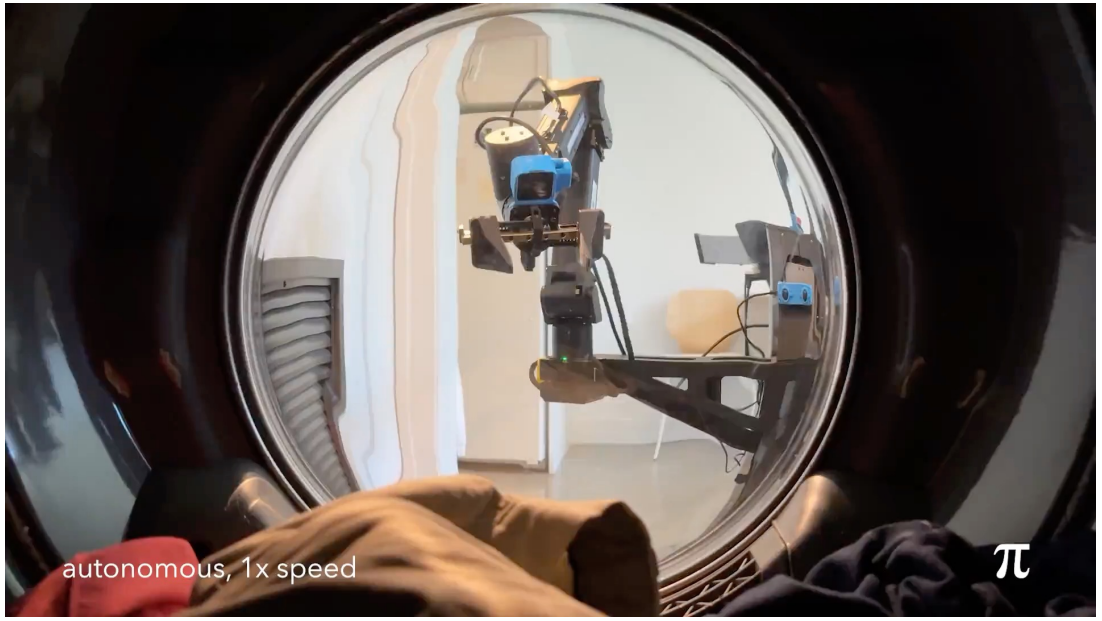
Video from ALOHA (Zhao et al., 2023)



Video from Diffusion Policy (Chi et al., 2023)

# When you have lots of data

Video Credit: DeepMind




Video Credit: Physical Intelligence

# “Policies” for Sequential Decision Making

For any input state of the system, the ML policy model maps it to a decision.

- This motivates the following input-output structure of the model:
  - Input: state observation, like sight and smell for the dog.
  - Output: actions, like muscle contractions.

This mapping from input states to a probability distribution over output actions (or sometimes just a single deterministic action) is called a decision-making “policy”, often denoted $\pi$.

# Supervised learning of Action Policies?

- Given the current "state" $x$, make a decision $\hat{y} = \max_y \pi_\theta(y|x)$.
  - Supervision => labels for "good" decisions that maximize rewards.
  - So, we'd like to have some dataset of (state $x$, good decision $y^*$) pairs. Then we could try running supervised learning just as always.

- For the sequential decision making problem, we will use the notation:
  - state input $s$ instead of $x$,
  - action output $a$ instead of $y$.
  - We will often subscript these items with time indices as $s_t$, $a_t$ etc.

# Behavior Cloning (BC)

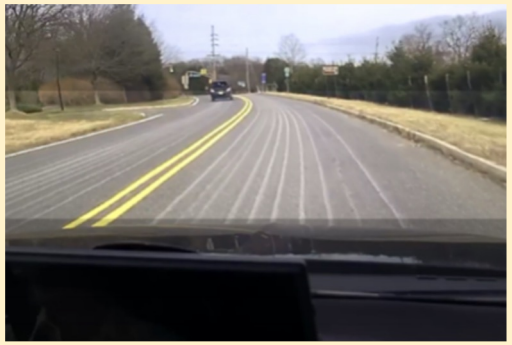
expert

observed states
$s_1, s_2, \ldots, s_H$
$a_1, a_2, \ldots, a_H$
actions

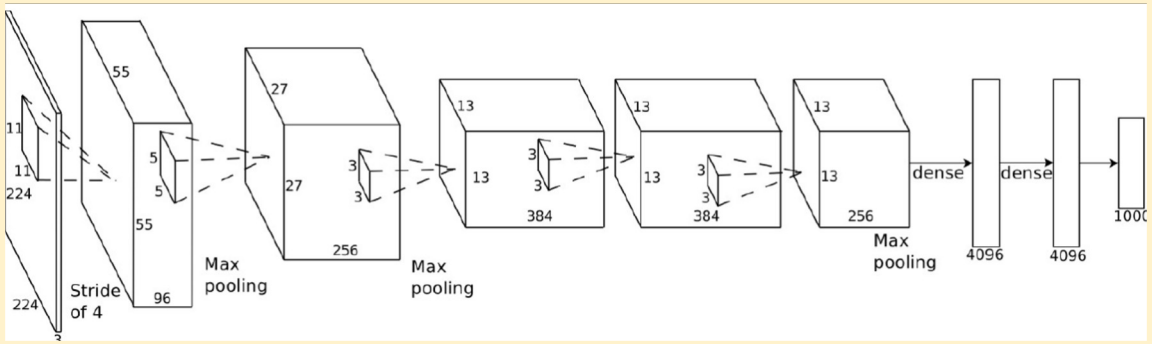
training data

supervised learning

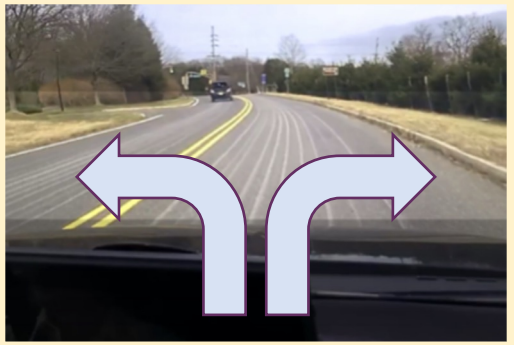
policy
$\pi_\theta(a_t|s_t)$



observed state $s_t$




convolutional network



action $a_t$

An “end-to-end” policy

# Behavior Cloning Objective Function

**Supervised maximum-likelihood objective to map from states to expert actions.**

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

Demonstration data

trajectories

time

Expert actions

**Could minimize by following the gradient:**

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

**Likelihood gradient:** "Change the policy to make these actions more likely".

# Behavior Cloning (BC) Objective Function

**Supervised maximum-likelihood objective to map from states to expert actions.**

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

Demonstration data

trajectories

time

Expert actions

- **Violates i.i.d. assumptions of supervised learning:** learner's prediction affects future input observations / states during execution of the learned policy. Will return to this later.

- **No connection to the task reward function?** Best, we can say is, if $R(.)$ is bounded e.g. [0, -1], then a policy with error rate $\epsilon$ on the expert data incurs a reward penalty $< O(T^2\epsilon)$

**Does this really work?**

# ALVINN:
# AN AUTONOMOUS LAND VEHICLE IN A NEURAL NETWORK

Dean A. Pomerleau
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

## ABSTRACT

ALVINN (Autonomous Land Vehicle In a Neural Network) is a 3-layer back-propagation network designed for the task of road following. Currently ALVINN takes images from a camera and a laser range finder as input and produces as output the direction the vehicle should travel in order to follow the road. Training has been conducted using simulated road images. Successful tests on the Carnegie Mellon autonomous navigation test vehicle indicate that the network can effectively follow real roads under certain field conditions. The representation developed to perform the task differs dramatically when the network is trained under various conditions, suggesting the possibility of a novel adaptive autonomous navigation system capable of tailoring its processing to the conditions at hand.

proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

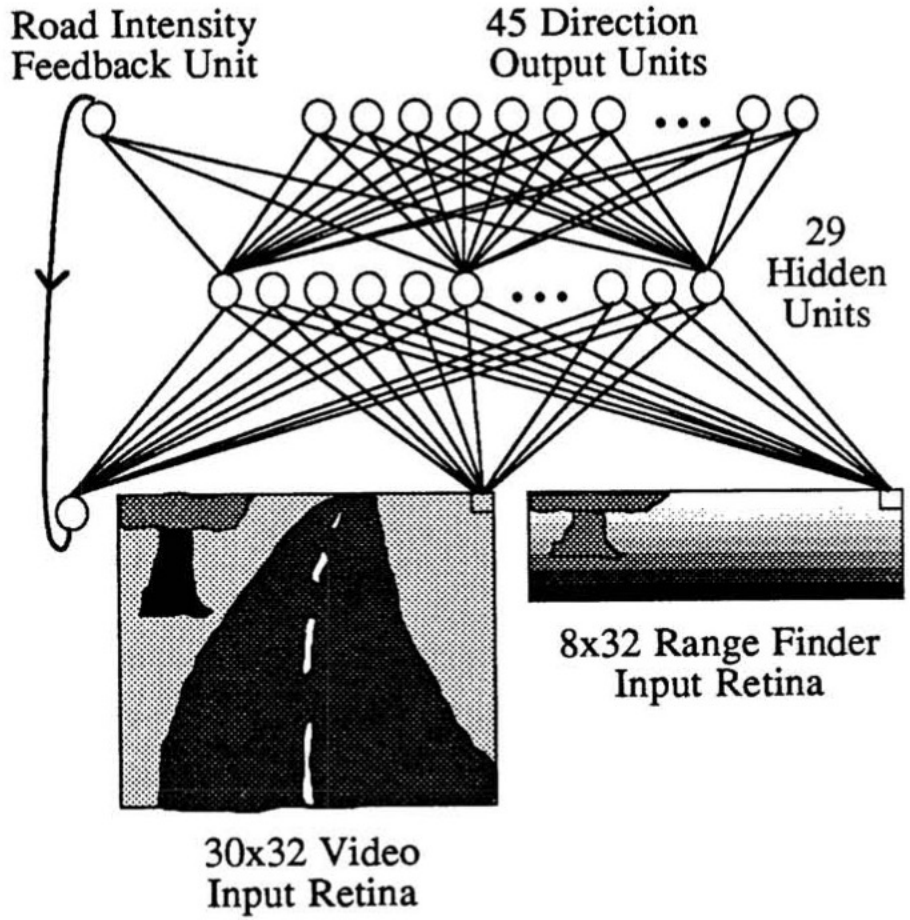



Figure 1: ALVINN Architecture

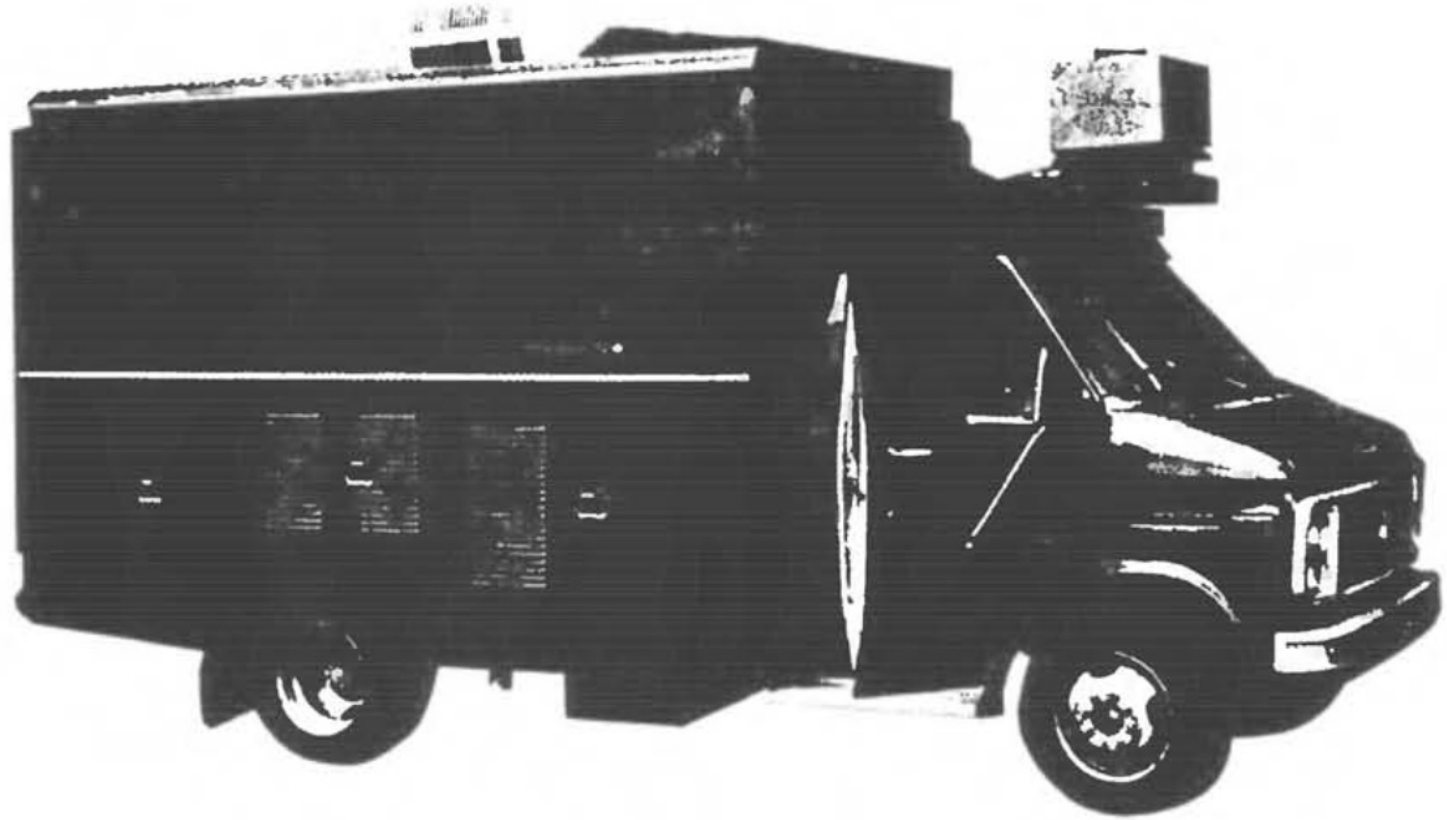


Figure 3: NAVLAB, the CMU autonomous navigation test vehicle.

“the network can accurately drive the NAVLAB at a speed of 1/2 meter per second along a 400 meter path through a wooded area of the CMU campus under sunny fall conditions”

proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

# “No Hands Across America” – 1995!

[No Hands Across America Home Page]



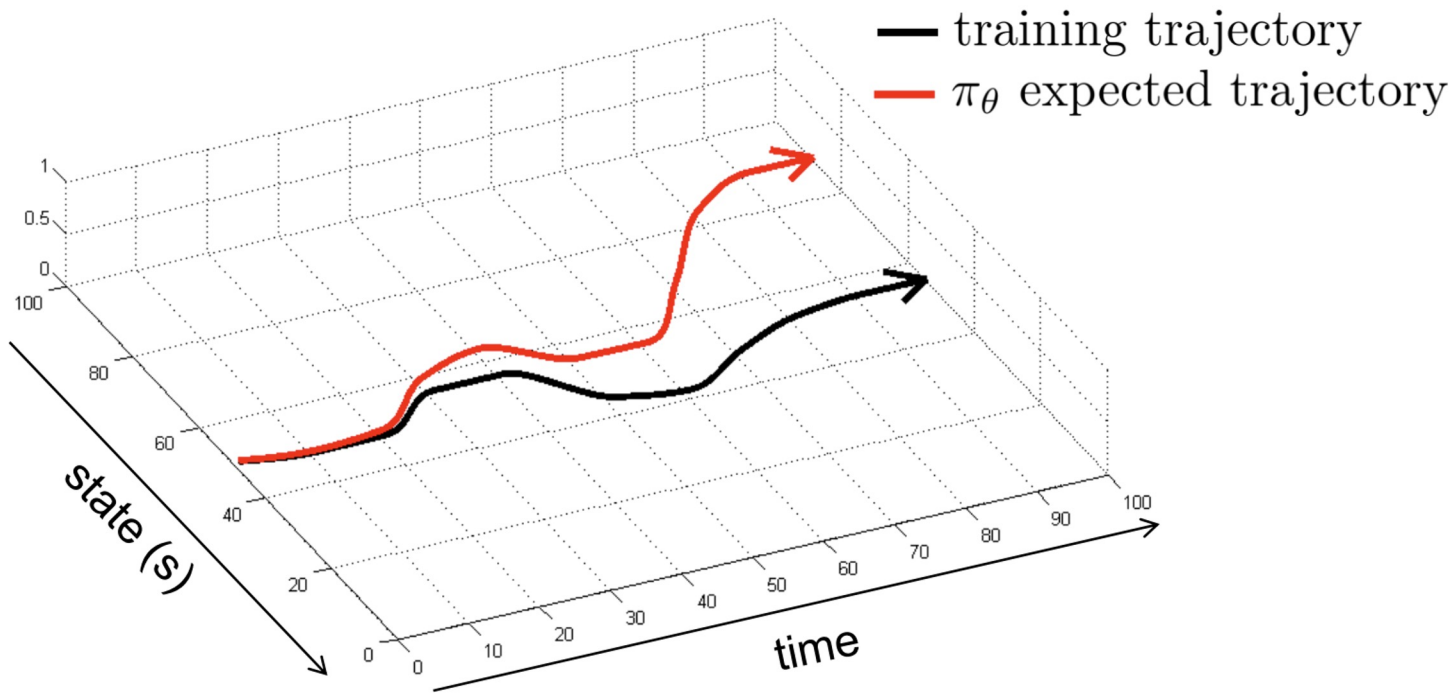
“On July 23, 1995 Research Scientist Dean Pomerleau and Ph.D. student Todd Jochem, both from the Robotics Institute of Carnegie Mellon University in Pittsburgh, PA, will begin a trans-continental journey in their 1990 Pontiac Trans Sport. If all goes well, the two will end up in San Diego, CA on July 30. What makes this trip special is that the vehicle will drive itself most of the way.”

**TRIP COMPLETE !!!**

**2797/2849 miles (98.2%)**

# Distribution Shift in BC

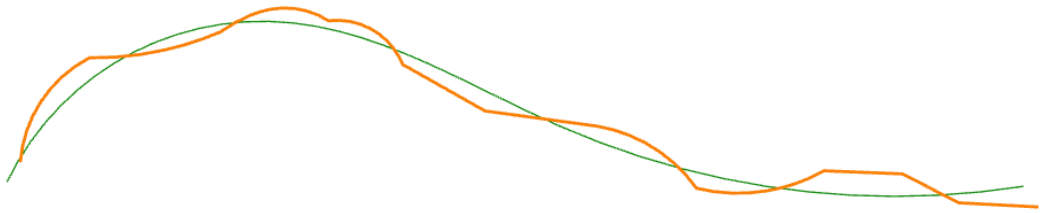
The policy is trained on *demonstration data* that is different from the data it encounters in the world.



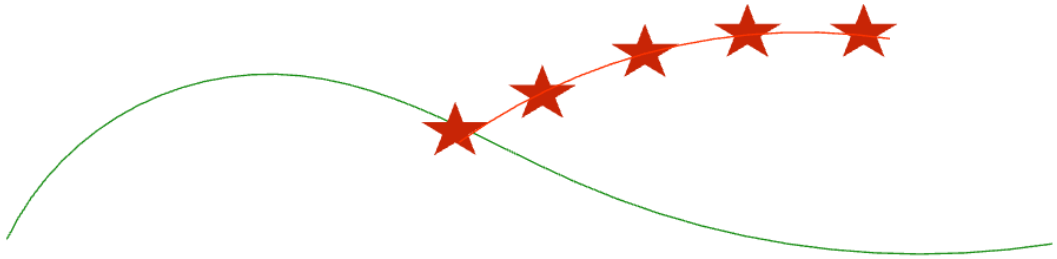

The cloned policy is imperfect; this leads to "compounding" errors, and the agent soon encounters unfamiliar states, leading to failure.

Note how these errors arise from ignoring the the *sequential, interconnected* nature of the task. Past decisions influence future states!

# Compounding Error In Behavior Cloning



Independent-in-time errors



Compounding error

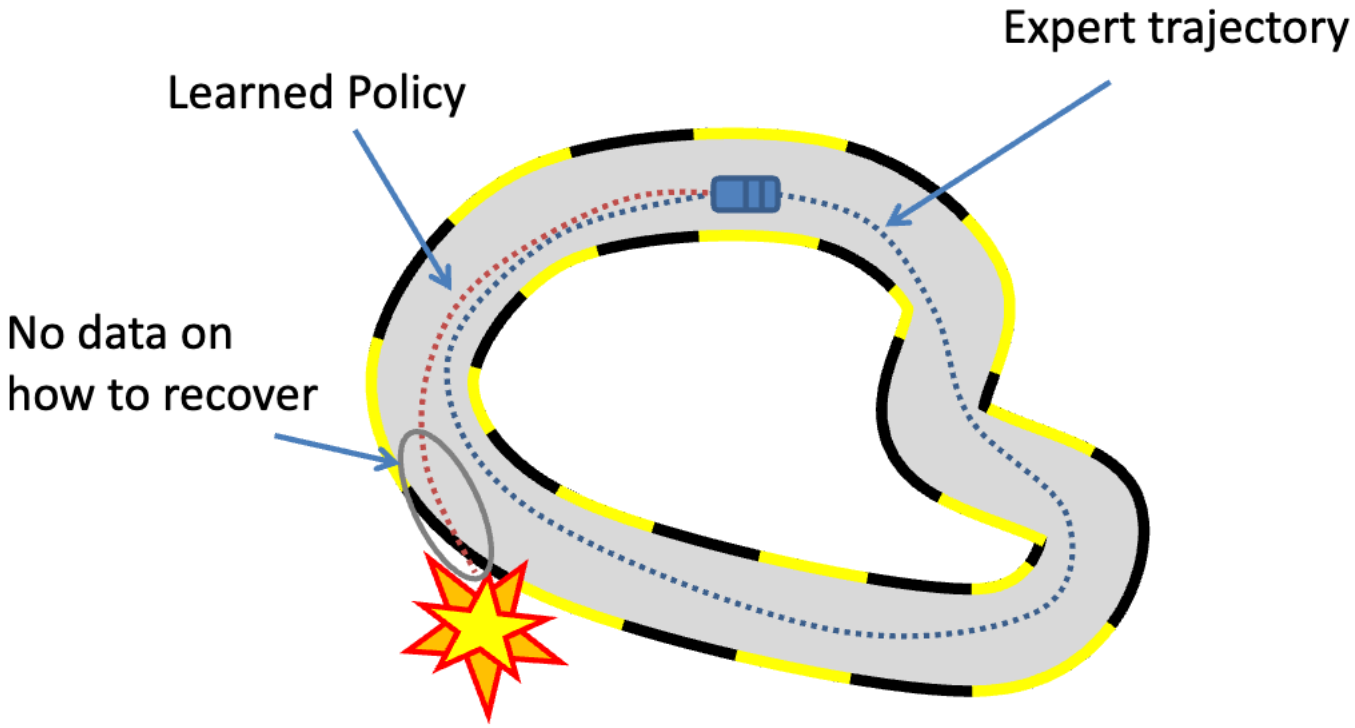
Images: Katerina Fragkiadaki

Ross and Bagnell 2010, Efficient Reductions for Imitation Learning

“When driving for itself, the network may occasionally stray from the center of road and so must be prepared to recover by steering the vehicle back to the center of the road.”

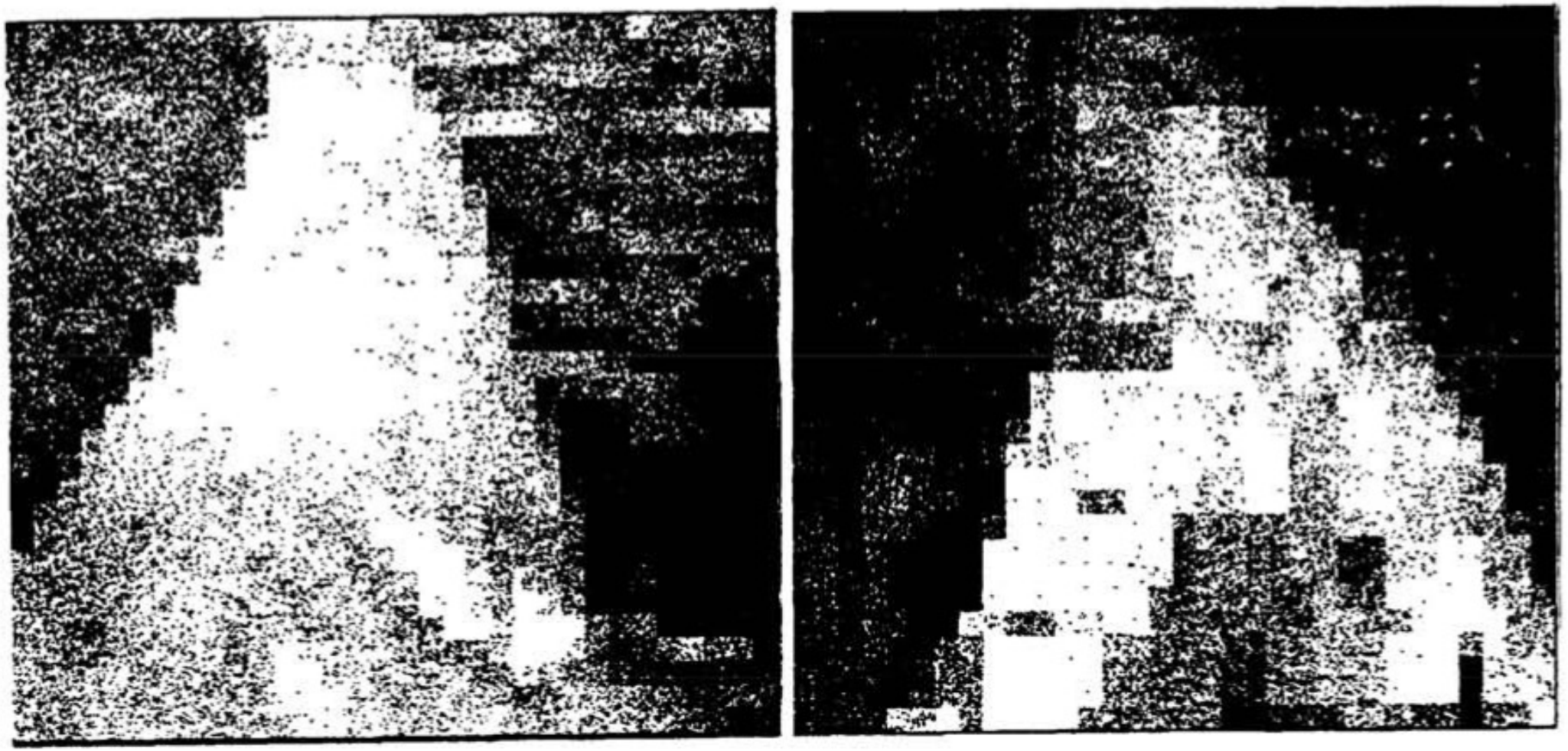
- Pomerleau ‘89

# What's Missing: "Recovery" Data




Images: Katerina Fragkiadaki

# ALVINN involved Simulated(!) Recovery Data

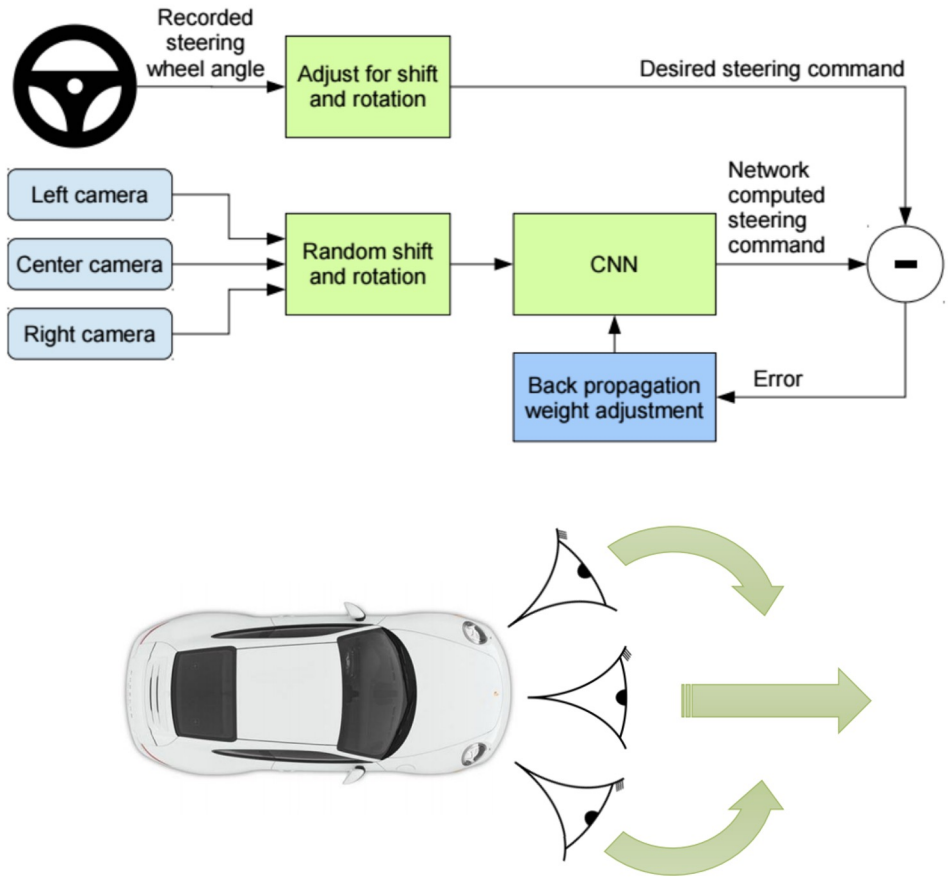


Real Road Image

Simulated Road Image

proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf

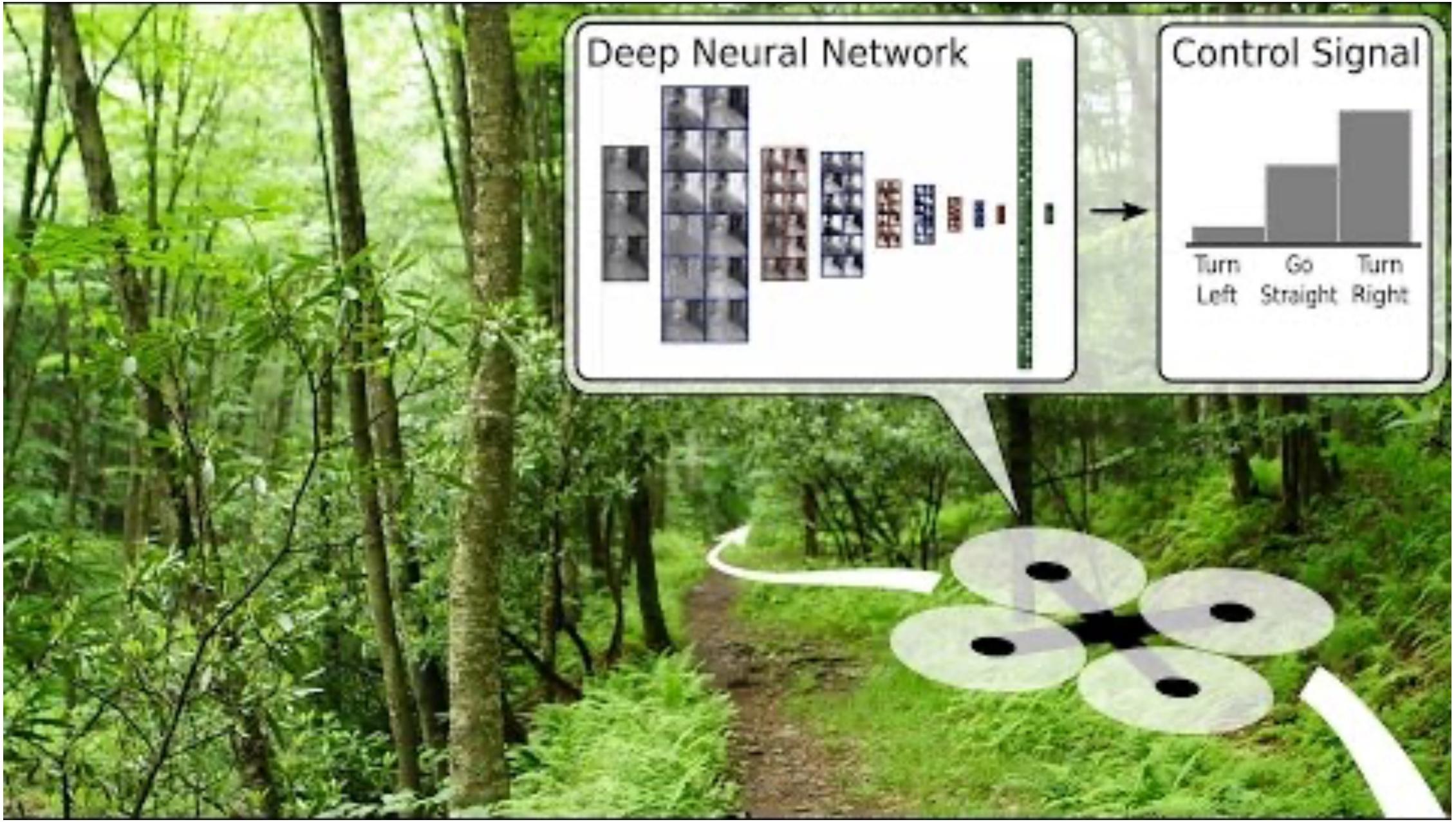
# Clever Recovery Data Collection For Navigation




[Dave-2 A Neural Network Drives A Car - YouTube]

**Hack to handle limited distributional shift for this specific MDP …** exploits the geometry of the state and action space.

Bojarski et al. 2016




[Quadcopter Navigation in the Forest using Deep Neural Networks]

Giusti et al, RA-L 2016

# *Interactive Online* BC: DAGGER [Ross et al, 2011]

A general trick for handling distributional shift: requery expert on new states encountered by the initial cloned policy upon execution, then retrain.

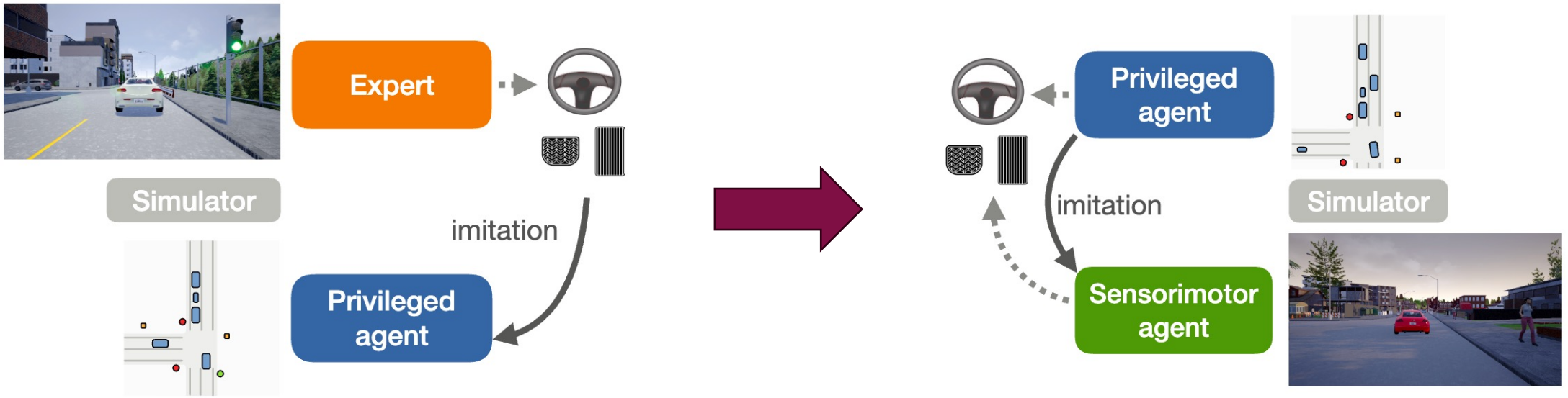
1. Train $\pi_\theta(a_t|s_t)$ from expert data $\mathcal{D} = \{s_1, a_1, \ldots, s_N, a_N\}$
2. Execute / "Rollout" $\pi_\theta$ to get dataset $\mathcal{D}_\pi = \{s_1^{new}, \ldots, s_M^{new}\}$
3. Ask expert to label each state in $\mathcal{D}_\pi$ with actions $a_t^{new}$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Assumes it is okay to keep asking the expert all through the training process.
"Queryable experts". Might not always be practical.

See also (for an offline, non-interactive variant): Lee et al 2017, DART: Disturbances for Augmentic Robot Trajectories

# Other Policies as “Experts”

- Rather than humans, sometimes the “expert” may be another policy. E.g. a policy trained from privileged observations, or a hand-scripted policy, or even a “traditional” model-based policy operating on the system state.
- E.g. ”learning by cheating” [Chen et al, 2020] in simulation:
  - first train an expert policy to operate from simulator state rather than images (e.g. with reinforcement learning, or even a hand-coded program)
  - Train a vision-based policy to imitate the expert.



# BC is akin to how language models are (first) trained

- Language models (LMs) e.g. the GPT family perform "next-token prediction"
  - At test time, they autoregressively predict the next token conditioned on their own previous outputs.

  $LM_\theta(\text{next response token } \hat{r}_i | \text{previous response tokens } \widehat{r_{<i}}; \text{prompt } p)$
  - At training time, they are trained with "teacher forcing" on a human-generated text dataset $\mathcal{D}$.

$$\text{LM Loss} = -\mathbb{E}_{\mathcal{D}}\left[\sum_{i=1,2,\ldots} \log LM_\theta(\hat{r}_i | r_{<i}; p)\right]$$

Compare this to:

$$\text{BC Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

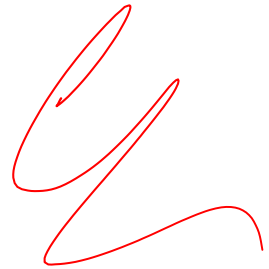
- Incidentally, after pre-training as above, LMs are typically finetuned with "reinforcement learning from human feedback" (RLHF).

# CIS 7000-04 / ESE 6800 Spring 2025: Intro To Imitation and Reinforcement Learning Part II

Thu, Jan 23, 2025

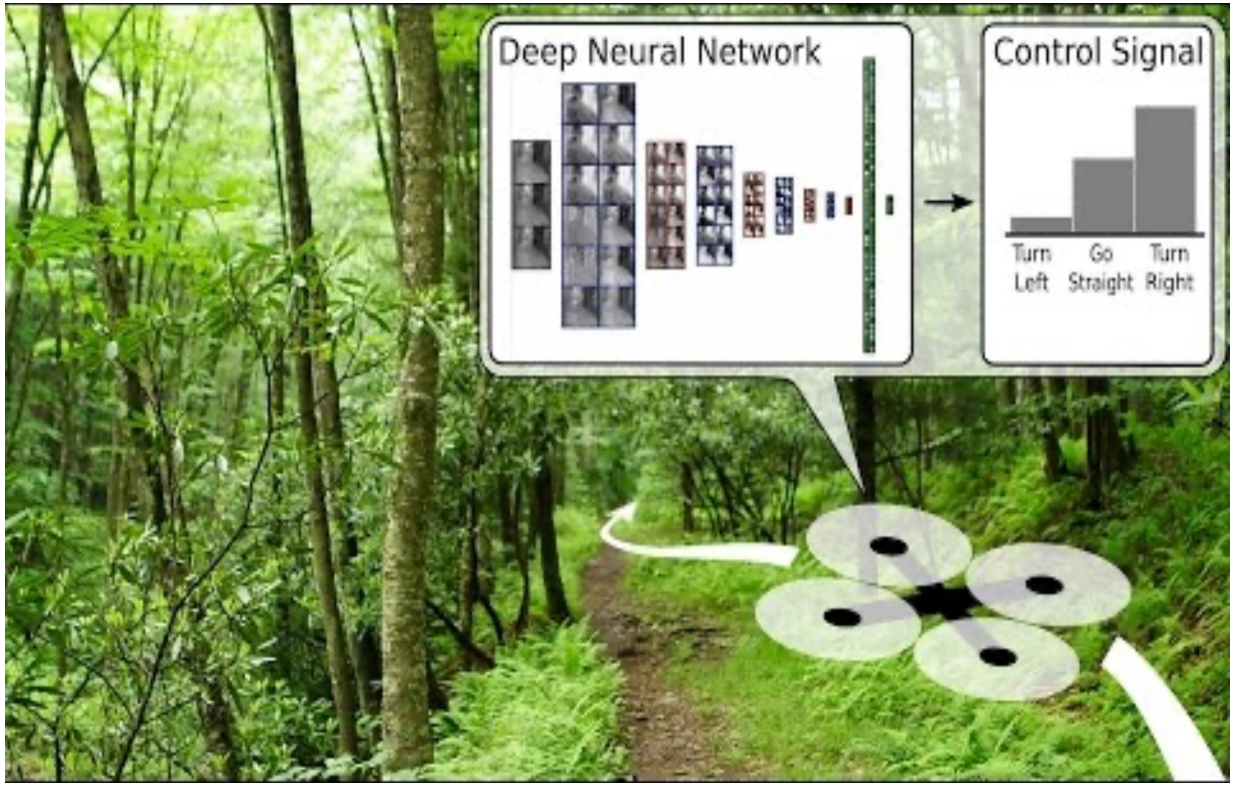
Instructor: Dinesh Jayaraman

Assistant Professor, CIS



Penn Engineering

# Recap (with working videos this time)




[Quadcopter Navigation in the Forest using Deep Neural Networks](#)
(3:15)

[Dave-2 A Neural Network Drives A Car - YouTube](#)

Bojarski et al. 2016

# Recap: Where we left off

- Naïve BC treats policy learning as a supervised learning problem: "expert" provides demos first, and the learner treats the demo as an unordered bag of state-action pairs to learn $\pi(a_t|s_t)$ from.
  - Can be very useful already, but prone to distribution shift -> compounding errors

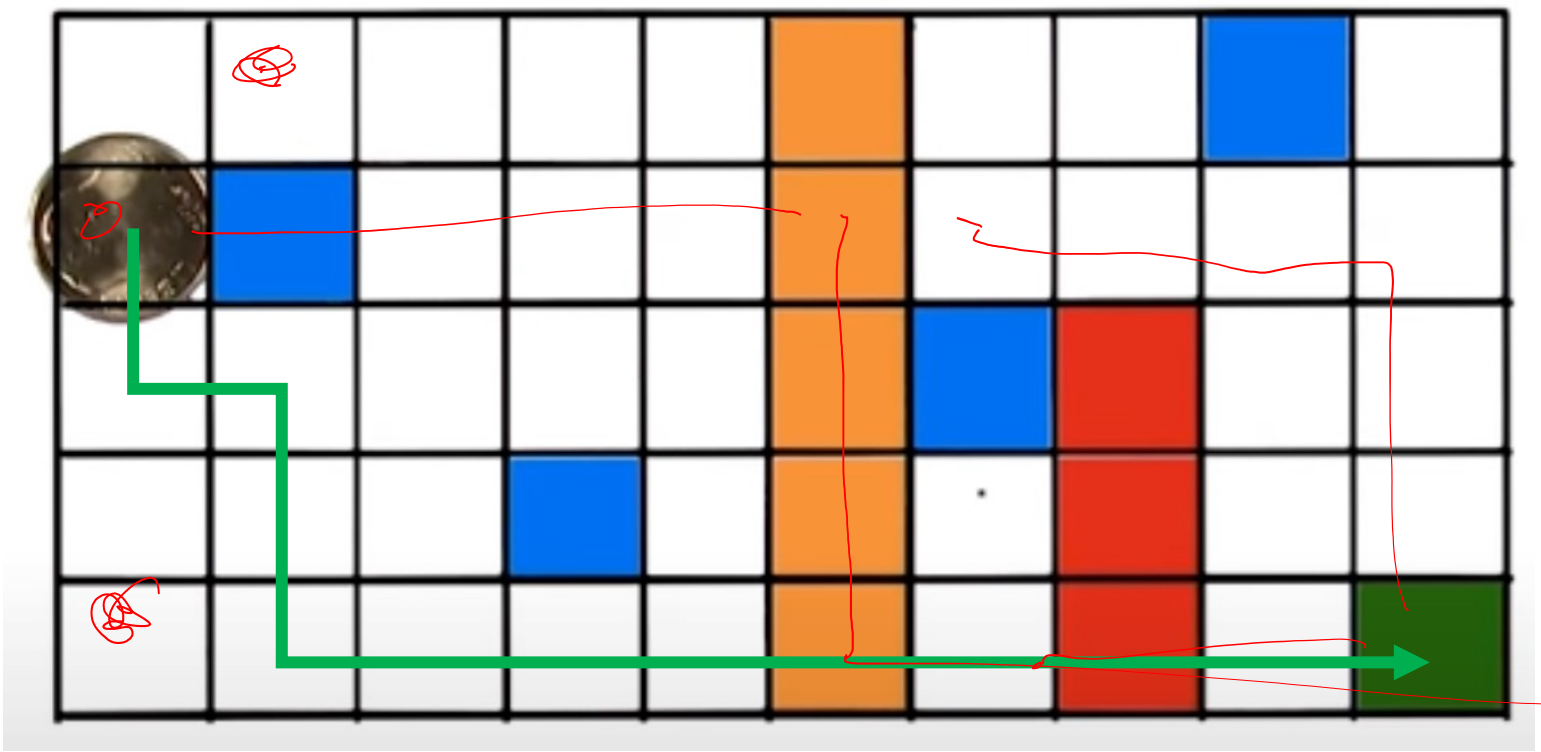
- Improvements to provide recovery data:
  - Simulating recovery data, cleverly gathering recovery data alongside "good" data, interactive *online* BC (i.e. DAGGER), etc.

# Plan for Today

- Wrapping up imitation learning: very brief overview of inverse RL
- Onwards to Reinforcement Learning:
  - From plain BC To Rewards-Informed BC (“offline RL”)
  - … To *Online Policy Gradient Approaches for RL*
    - *The difficulty of online RL*
    - *Policy gradient expression (w/ proof sketch?)*
    - *Monte-Carlo approximation and the REINFORCE algorithm*
  - … Reducing variance through a ”critic” → value functions

# Other Ways to Do Imitation: Inverse RL

- BC might not generalize beyond demonstrations. Instead learn explicitly about the "reward" function that the demonstrator is trying to maximize?
  - This is called "inverse reinforcement learning"



Would you conclude that this agent likes / dislikes:

- Blue squares?
- White squares?
- Orange squares?
- Red squares?
- Green square?

Knowing such *rewards* could inform more generalizable imitation, e.g. starting from a different location than expert. This usually involves RL.

# Stanford Helicopter Stunts with Inverse RL (2008)

Autonomous Helicopters Teach Themselves to Fly Stunts – YouTube (2:35)

Andrew Ng, Pieter Abbeel et al, 2008

# Injecting reward information into BC

BC objective:

$$-\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\log\pi_\theta(a_{i,t}|s_{i,t})\right)$$

Ignores the true objective of the MDP, which is indicated by the MDP reward function

Optimal demonstration data

Optimal actions

Reward-weighted regression objective:

$$-\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\log\pi_\theta(a_{i,t}|s_{i,t})\left(\sum_{\tau=t,t+1,\ldots}\gamma^{\tau-t}\,r_{i,\tau}\right)\right)$$

(Non-optimal) demonstrated data

(Non-optimal) demonstrated actions

Reward returns: "how good was this action?"

# Injecting Reward Information into BC

Reward-weighted regression objective:

$$-\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\log\pi_\theta\left(a_{i,t}\middle|s_{i,t}\right)\sum_{\tau=t,t+1,\ldots}\gamma^{\tau-t}\,r_{i,\tau}\right)$$

(Non-optimal) demonstrated data

(Non-optimal) demonstrated actions

Reward returns: "how good was this action?"

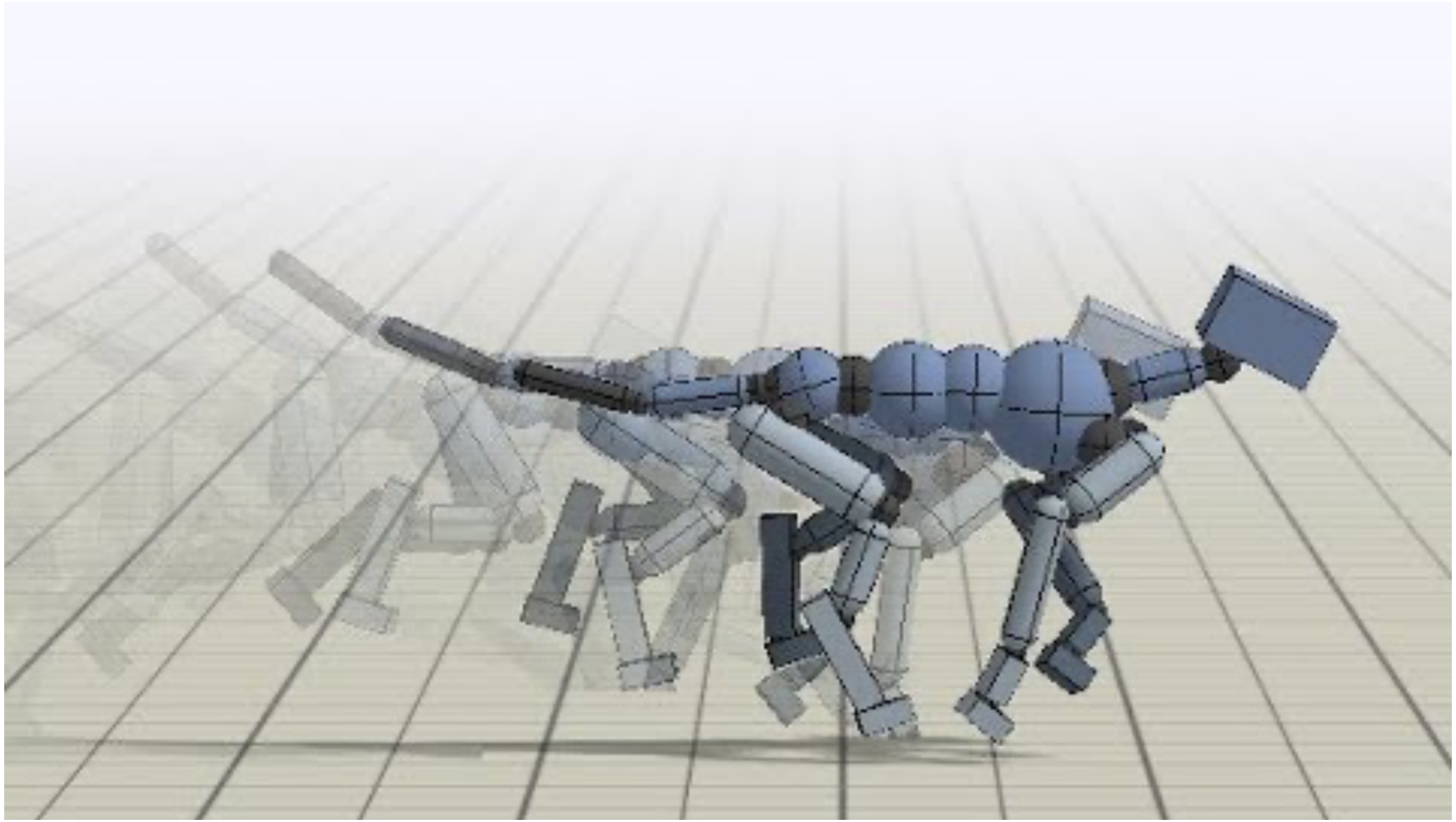
Peters & Schaal, Reward-weighted regression, ICML 2007

Jan Peters et al, Relative Entropy Policy Search, AAAI 2010

Led later to "advantage-weighted regression" [Peng et al 2019]

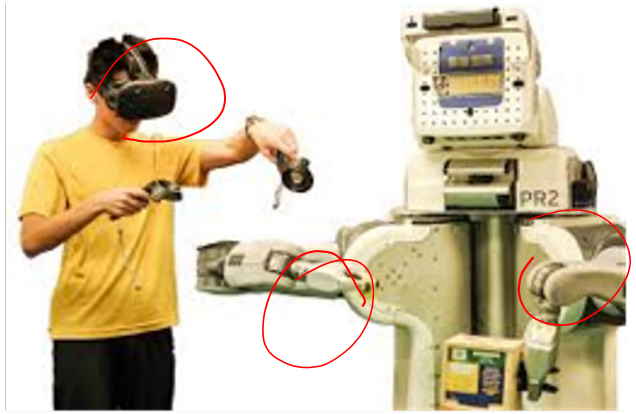
One of a class of "**offline RL algorithms**", that can exploit a pre-recorded dataset of *sub-optimal* behaviors by using rewards to find optimal policies.


RI Seminar: Jan Peters : Motor Skill Learning: From Simple Skills to Table Tennis and Manipulation (29:49)
[2013 results]



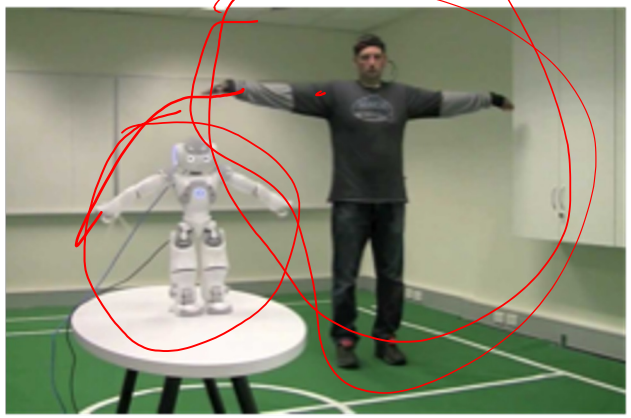
AWR: Advantage-Weighted Regression (1:29)

# Tools to Demonstrate Robot Behavior …




teleoperation

kinesthetic



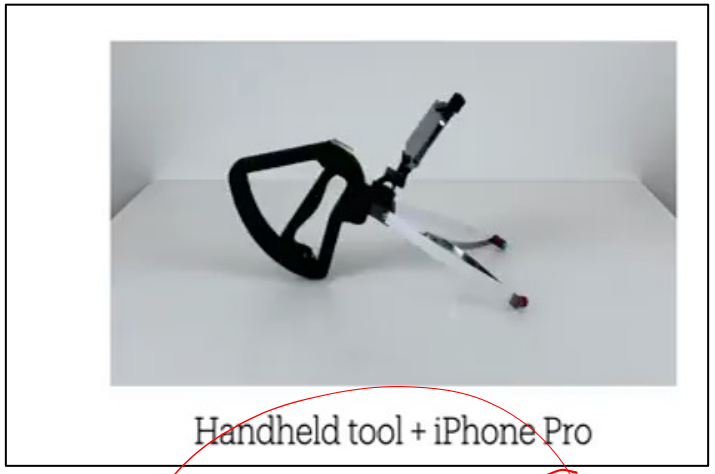
motion capture

web video

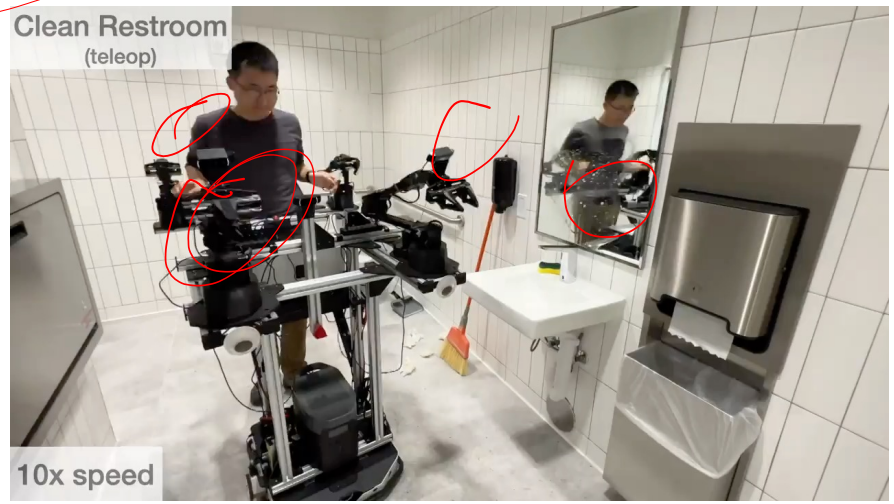
Argall et al, 2009

# Tools To Demonstrate Robot Behavior: Recent Progress

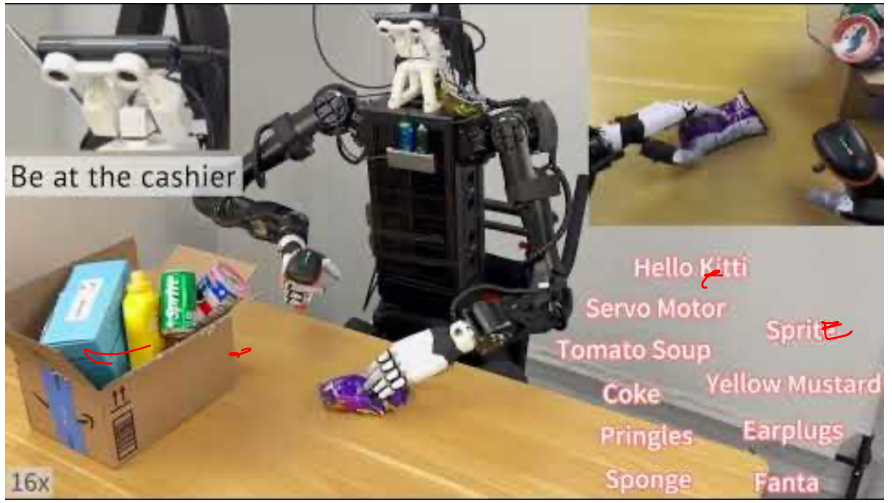
UMI




Stick-V2 (RUM)




Mobile Aloha




Open-TeleVision

# Do we have enough robot data?

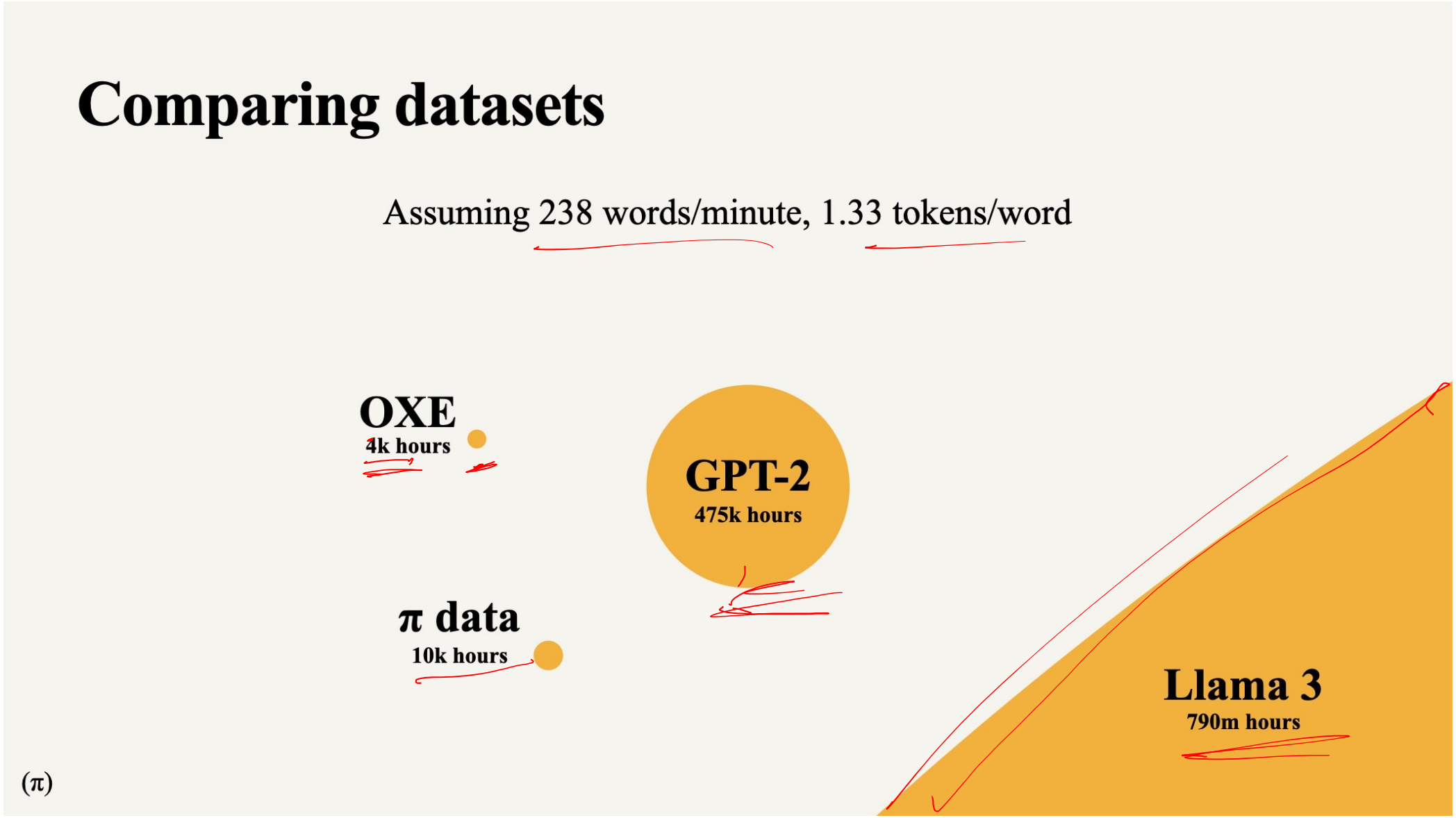



diagram credit: Kevin Black

# Some Key Limitations of Imitation For Robots

- **Data Scarcity**
- **Demo Sub-optimality**
- **Multimodality**
- **Cross-Embodiment**
- **Teacher-Student Discrepancies**
- **Safety**
- …

More on these in detail when we get around to the module on robot imitation later in the semester

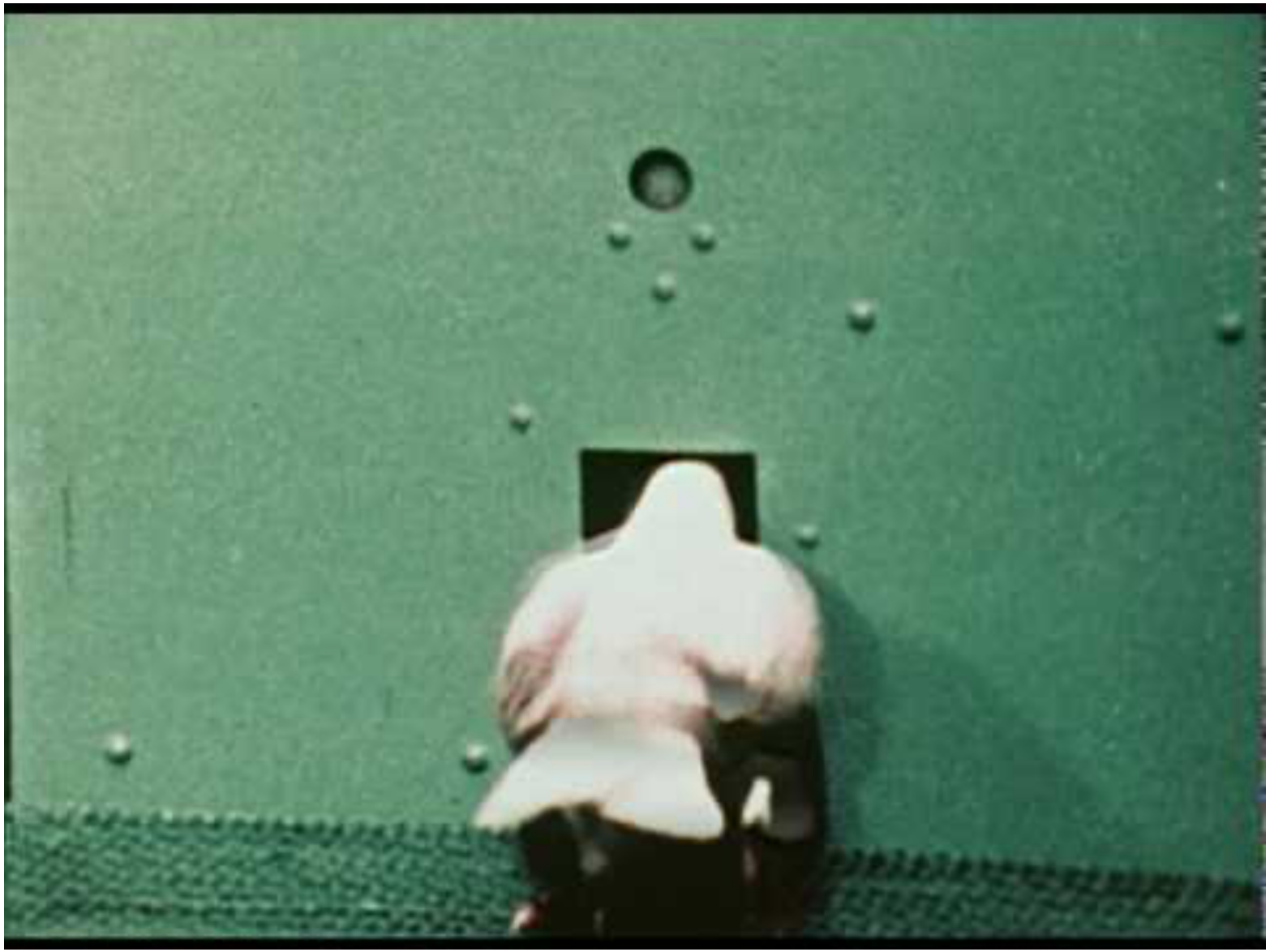
# Going Beyond Imitation

- More broadly, imitation is limited to mimicking experts and cannot discover new solutions. What about solving new problems, like controlling a new robot, or trading on the stock market, or beating the world's best Go player?

- Reinforcement Learning (next) addresses all this more carefully and comprehensively.
  - Imitation can be thought of as short-cutting the data collection part of the broader RL problem, and there are also ways to naturally combine imitation and RL.

# *[Online]* Reinforcement Learning

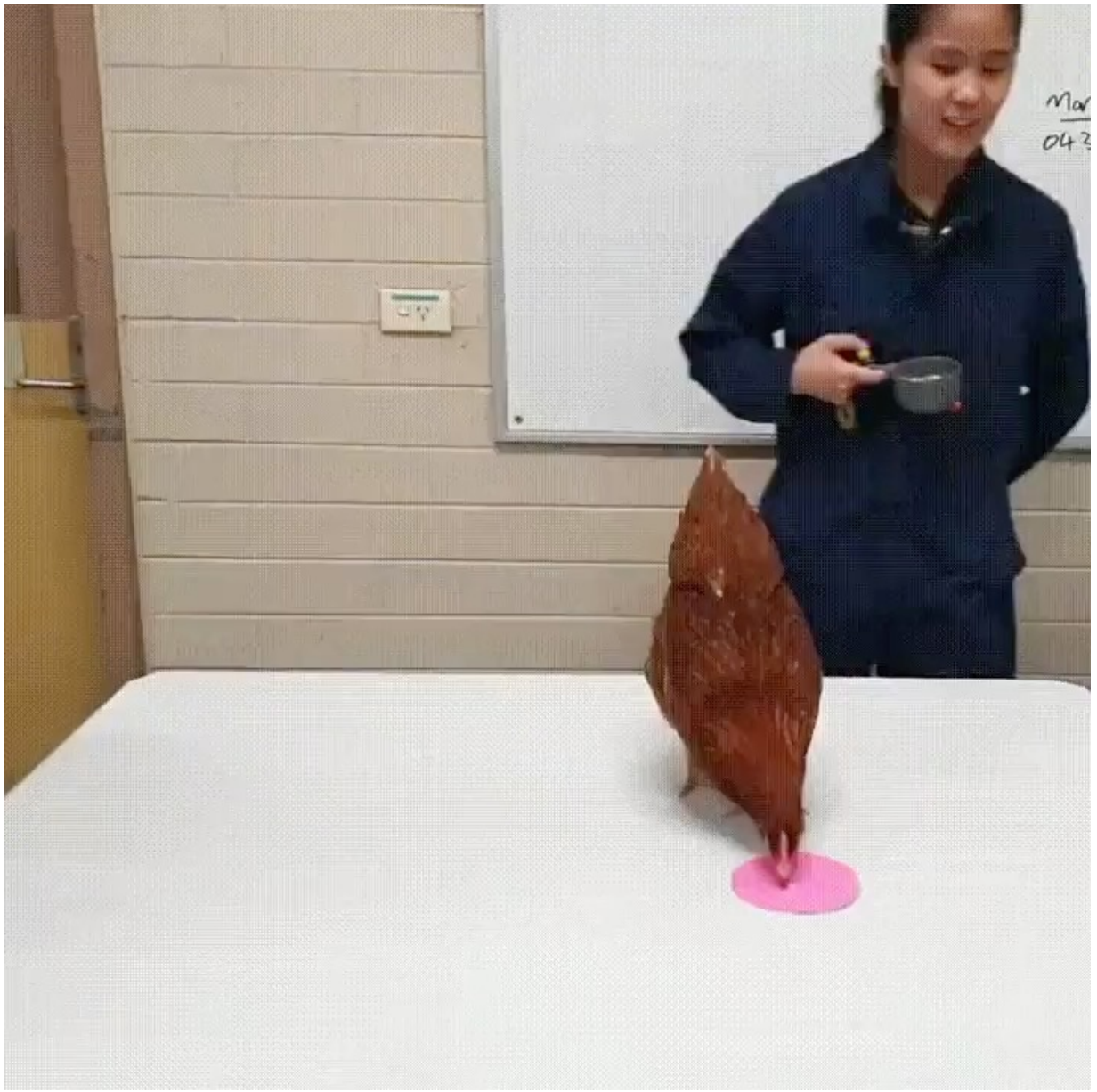
# B. F. Skinner's "Operant Conditioning" (1904– 1990)



(Go to ~ 0:30)

Project Pigeon - Wikipedia

# Learning Through Trial and Error (+ a curriculum)



Source: the interwebs

# Learning Through Trial and Error

The aim of RL is to learn to make sequential decisions in an environment:

- Driving a car
- Cooking
- Playing a videogame
- Controlling a power plant
- Riding a bicycle
- Making movie recommendations
- Navigating a webpage
- Treating a trauma patient

**How does an RL agent learn to do these things?**

- Assume only occasional feedback, such as a tasty meal, or a car crash, or video game points.
- Assume very little is known about the "environment" in advance.
- Learn through trial and error.

# Recap: MDP Formulation Notations

An MDP $(S, A, P, R, \gamma)$ is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function or "dynamics model" $P(s' \mid s, a)$
  - Probability $P(s' \mid s, a)$ that $a$ from $s$ leads to $s'$
- Reward function $r_t = R(s, a, s')$
- Discount factor $\gamma < 1$, expressing how much we care about the future (vs. immediate rewards)
- "utility" = *discounted* future reward sum $\sum_t \gamma^t r_{t+1}$
- Goal: maximize *expected* utility

Example

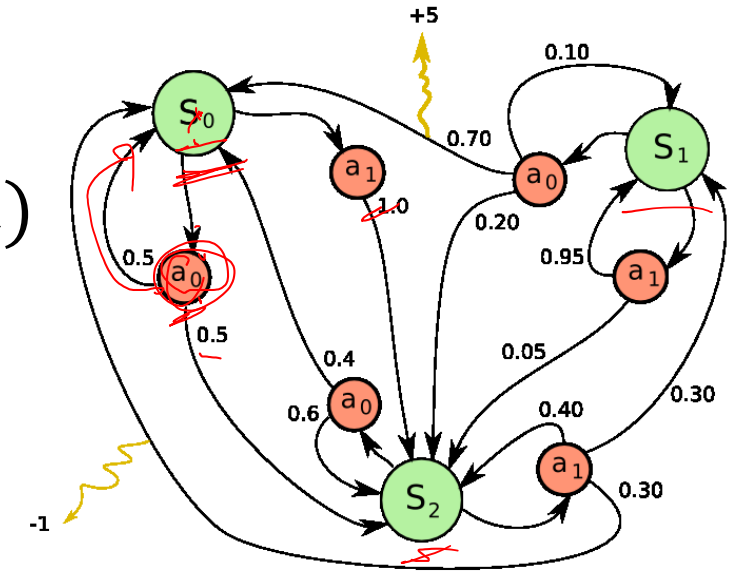



Image: https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

# Notation: Rewards, Returns, Utilities, Discounted …

- "reward": instantaneous reward $r_t$ received at one time instant from the environment
- "return": sum of (future) rewards
  - Sometimes also called "cumulative reward", "utility", etc.
  - By default includes the discount factor i.e. return = $\sum_t \gamma^t r_{t+1}$
    - sometimes called the "discounted return" to make this explicit and distinguish from "(undiscounted) return" = $\sum_t r_{t+1}$.

**Outside of this class**, lots of confusion, beware! For example:

- Sometimes "reward" ↔"return"
- Sometimes "return" is by default undiscounted, etc.

# Online RL is hard!
# Entering An Unknown Gridworld

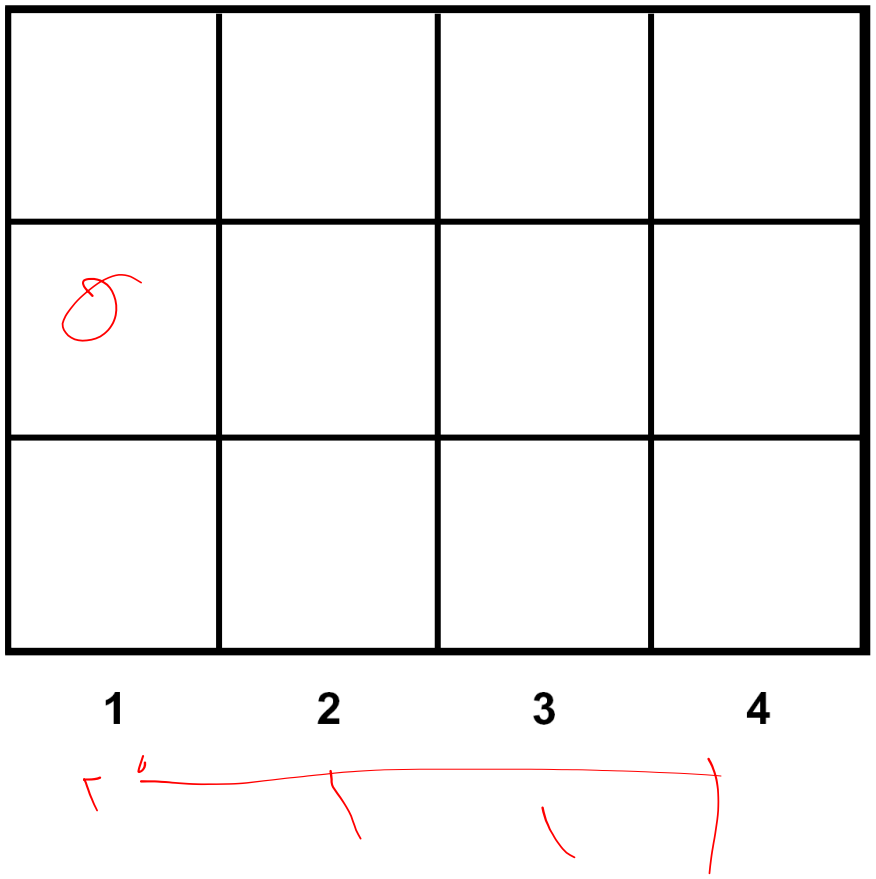
In the shoes of an Online RL agent

# Sample RL environment: Grid World

- The agent's state is one cell $s = (x, y)$ within the grid $x \in \{1,2,3,4\}$ and $y \in \{1,2,3\}$.
- The agent can execute 4 actions: $a =$"W", "E", "S", "N"

For the moment, this is all that that the RL agent knows about the environment. In particular, it does not know:

- $P(s'|s, a)$
  - Which cell would it move to, if it executes an action from a cell? (e.g. $a =$"N" from $s = (1, 2)$)
  - The result might even be non-deterministic.
- $R(s, a, s')$
  - What is the instantaneous reward it would get if it moved from $s = (1,2)$ to $s' = (1,3)$ by executing action $a =$"N"?

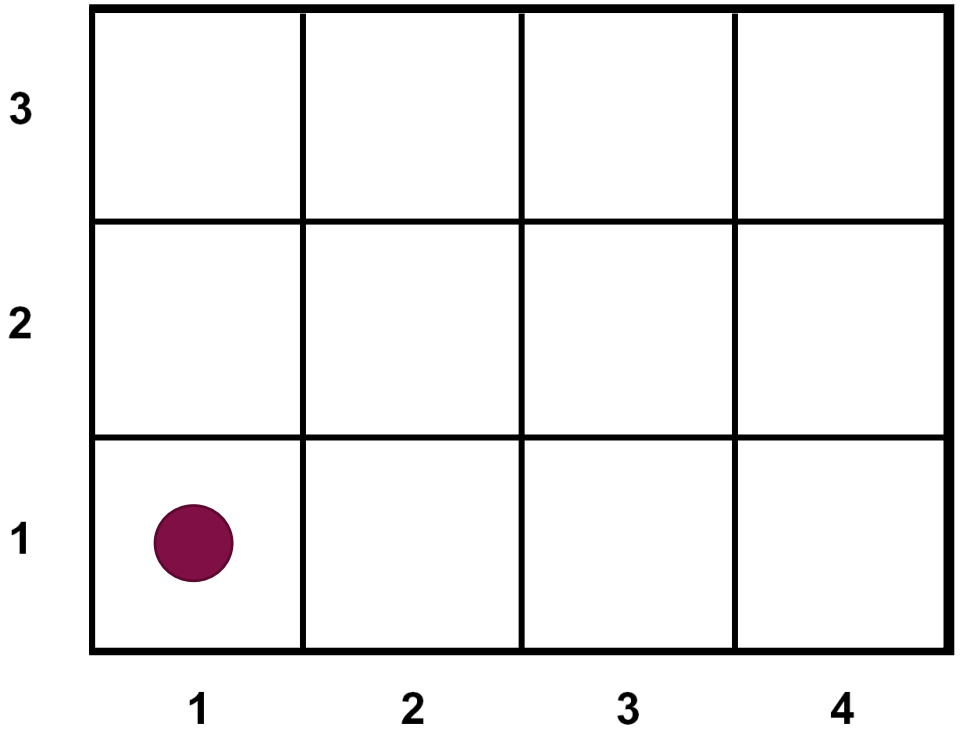


# A random trajectory of an RL agent

Time t=0

s=(1,1)

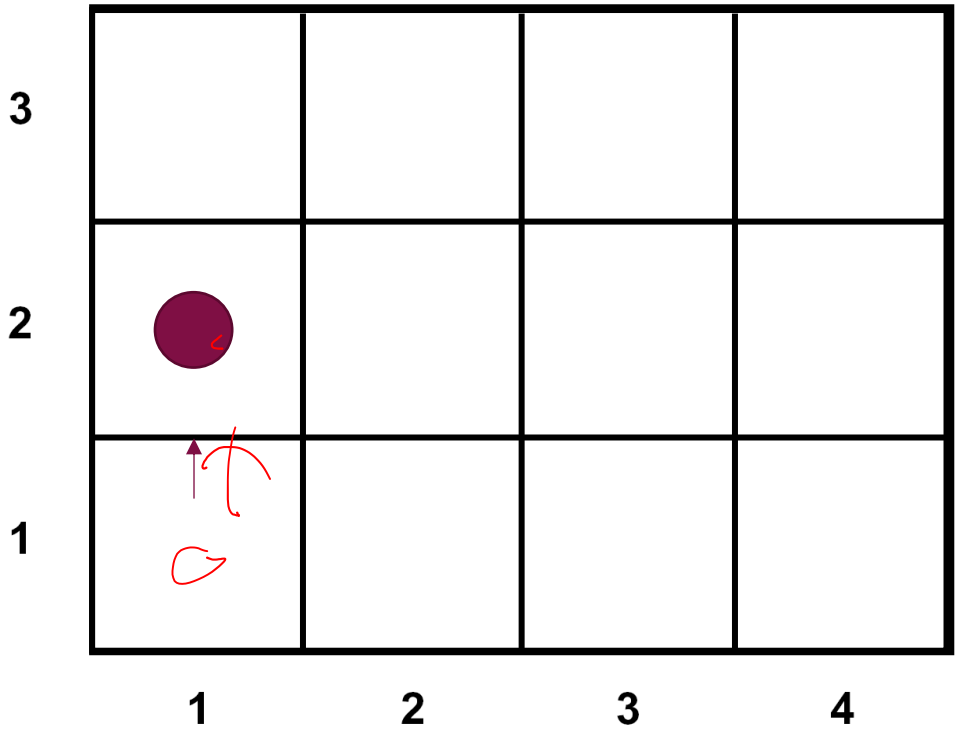
Action= “N”



# A random trajectory of an RL agent

Time t=0

s=(1,1)
Action= “N”
s’=(1,2)
Reward = -0.03

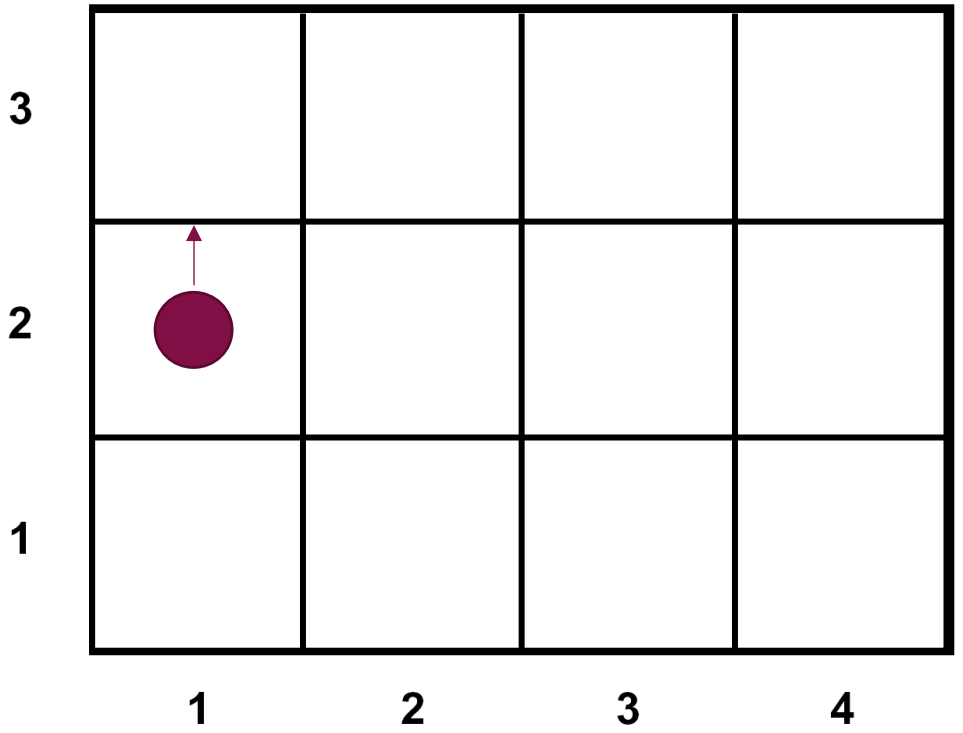



Time step t=0 over

# A random trajectory of an RL agent

Time t=1

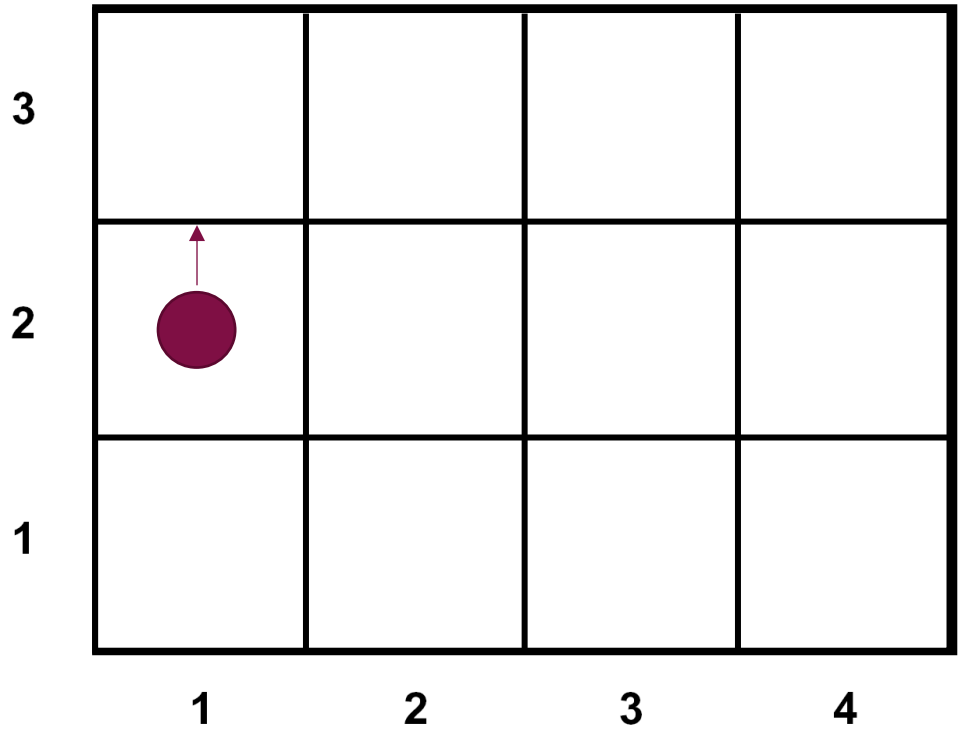
s=(1,2)
Action= “N”
s’=?
Reward = ?



# A random trajectory of an RL agent

Time t=1

s=(1,2)
Action= “N”
s’=(1,2)
Reward = -0.03

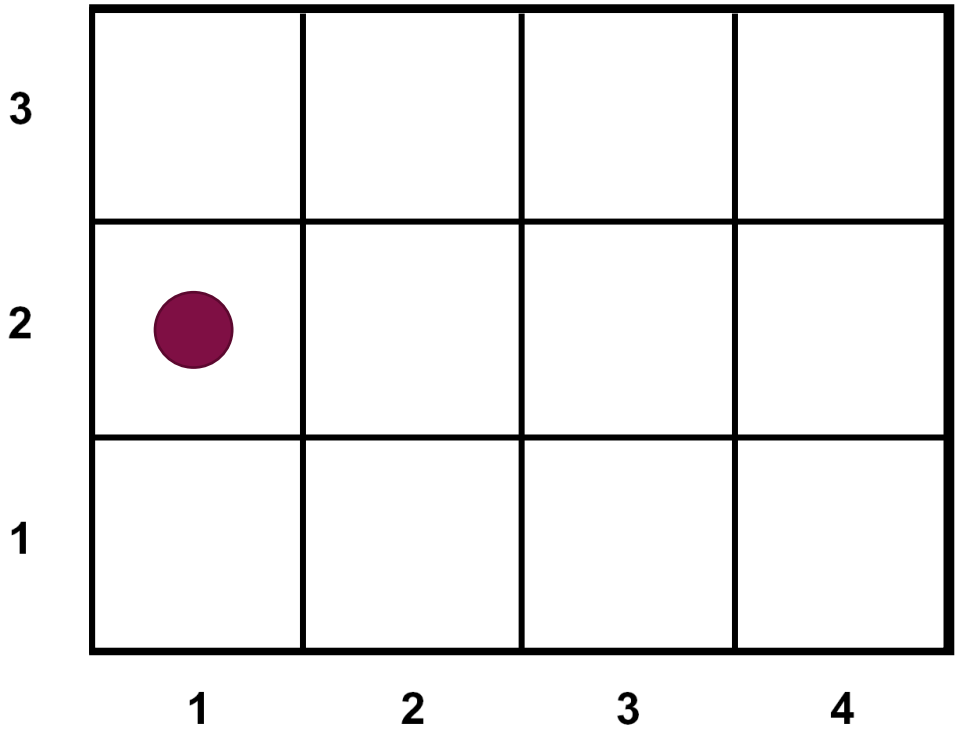



Time step t=1 over

# A random trajectory of an RL agent

Time t=2

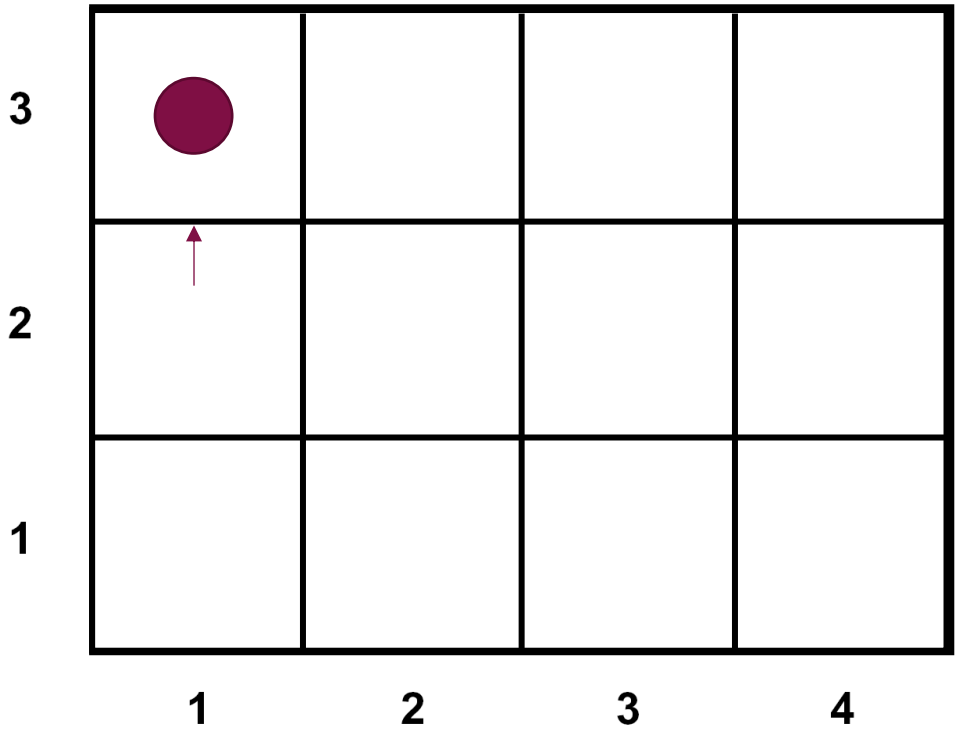
s=(1,2)
Action= “N”
s’=?
Reward = ?



# A random trajectory of an RL agent

Time t=2

s=(1,2)
Action= “N”
s’=(1,3)
Reward = -0.03

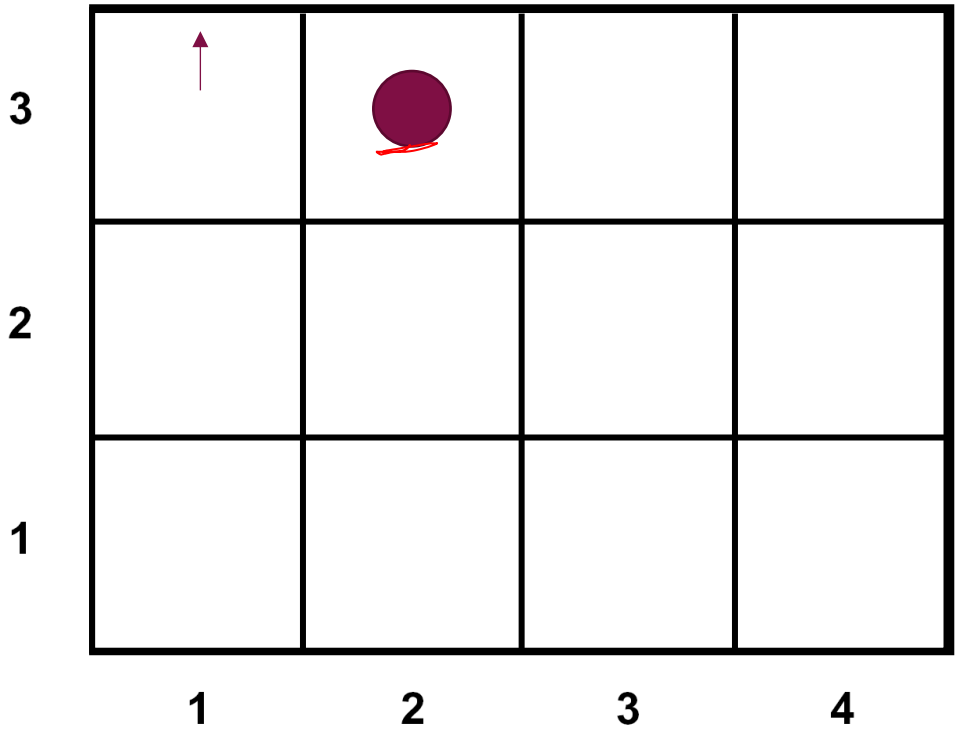



Time step t=2 over

# A random trajectory of an RL agent

Time t=3

s=(1,3)
Action= “N”
s’=(2,3)
Reward = -0.03

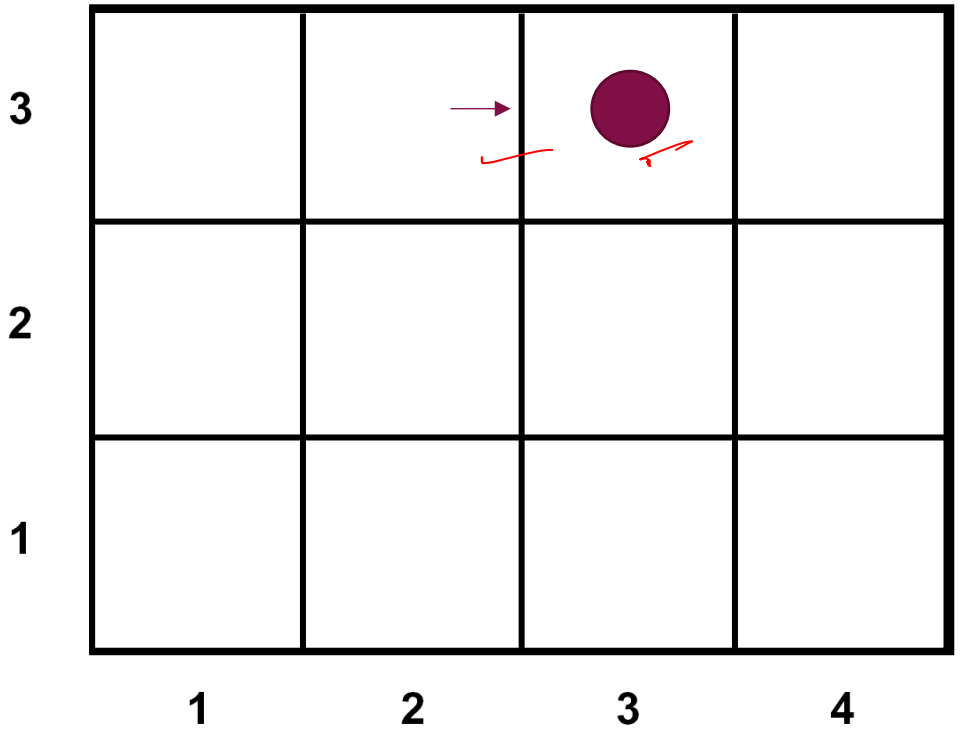



Time step t=3 over

# A random trajectory of an RL agent

Time t=4

s=(2,3)
Action= “E”
s’=(3,3)
Reward = -0.03

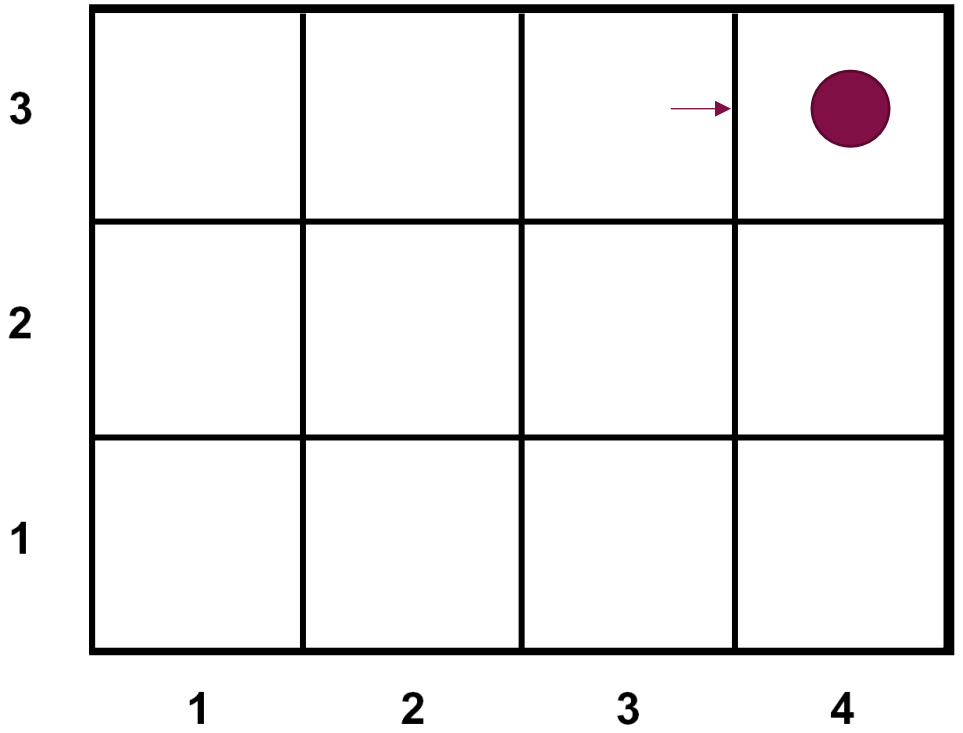



Time step t=4 over

# A random trajectory of an RL agent

Time t=5

s=(3,3)
Action= “E”
s’=(4,3)
Reward = -0.03




Time step t=5 over

# A random trajectory of an RL agent

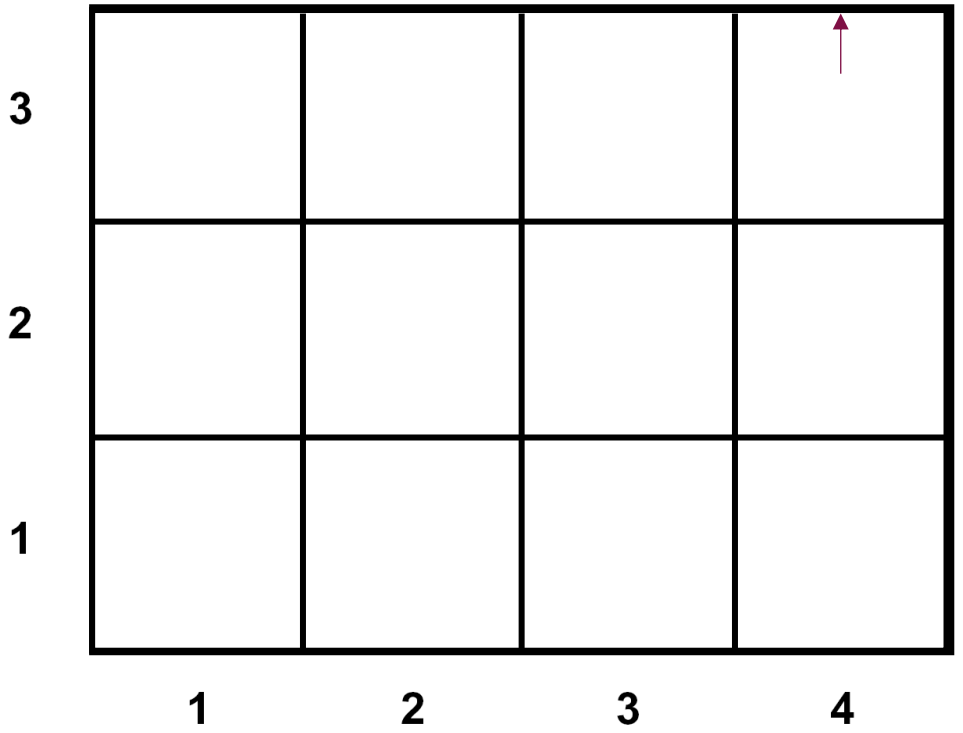
END

s=(4,3)

Action= “N”

s’= special state “END”

Reward = +1

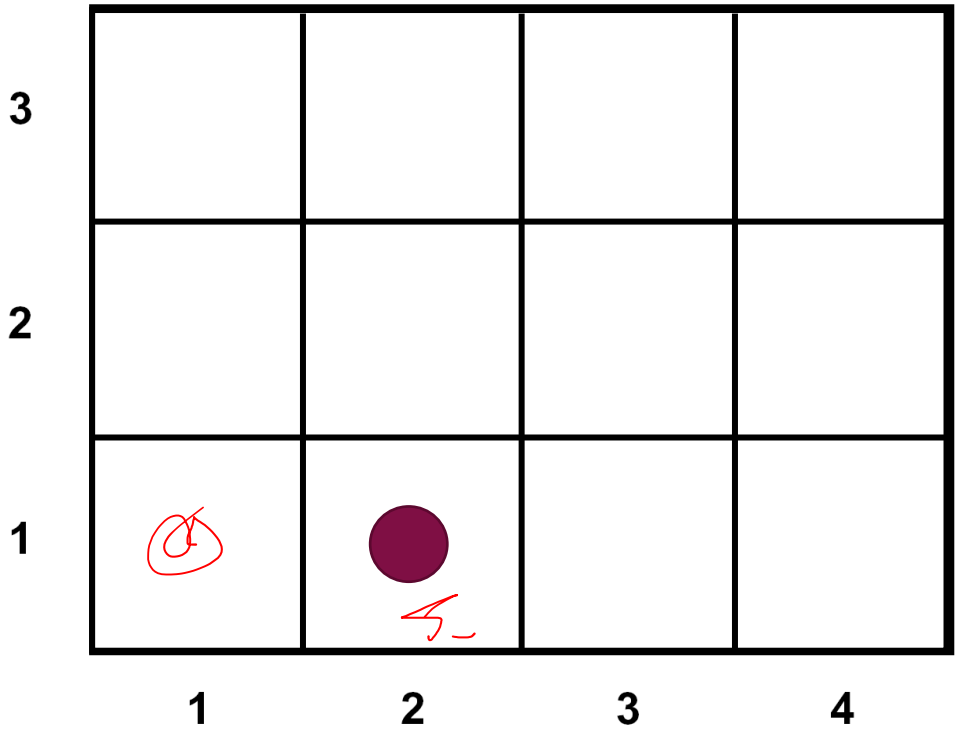



One “episode”/“trial” of our “episodic task” is over.

Next, the agent respawns in the environment. “Reset”

# Reset

Another episode begins!
Time t=0




s=(2,1)
Action=?
s'= ?
Reward = ?

Note that we have started at a different point in the grid than last time.
In addition to $(S, A, P, R, \gamma)$, there may also be an "initial state probability distribution" $\mu$ over states that the agent is spawned into.

# So, can we maximize rewards in this environment?

- What have we learned about this environment after having acquired this experience?
  - Do we know something about $P, R$?
  - Do we know how to act optimally now?

We have learned some things, but there is still far too much ambiguity.

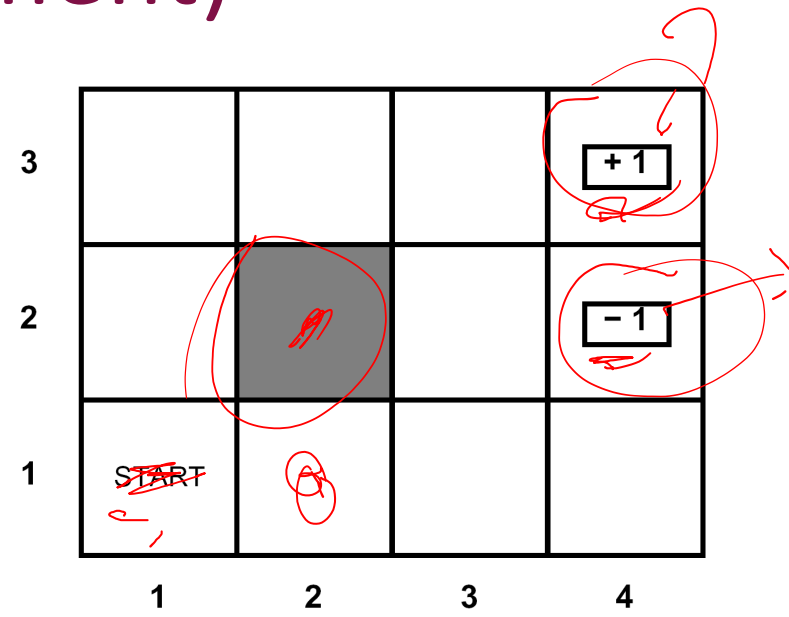
Perhaps with more experience …?

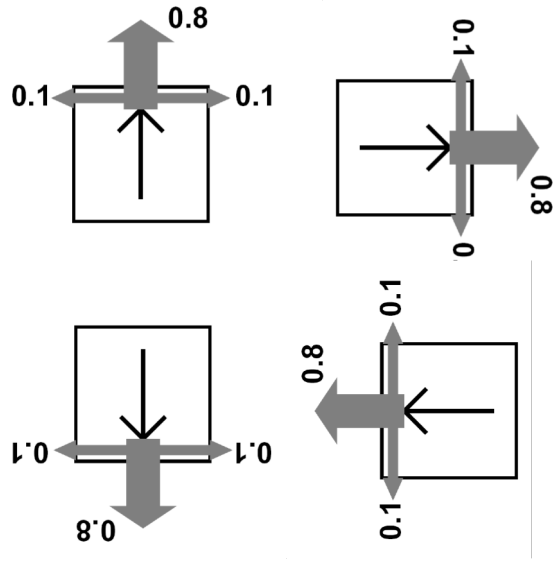
**Indeed, RL algorithms can acquire sufficient experience and learn optimal policies!**

# Gridworld Revealed

# (Behind The Scenes: The Full Environment)

- A grid map with solid / open cells. Agent('s dot) moves between open cells.
- From terminal states $(4,3)$ and $(4,2)$, any action ends the episode, and results in a +1/-1 reward respectively.
- For each timestep outside terminal states , the agent pays a small "living" cost (negative reward): $-0.03$
- The agent actions N, E, S, W correspond to North, East, South, West
  - **But the outcomes of actions are not deterministic!**
    - The dot obeys the commanded motion direction 80% of the time
    - 10% of the time, the dot instead executes a different direction 90° off from the agent command. Another 10% of the time, -90° off.
    - E.g. if dot surrounded by open cells and executing action N, will end up in the northern cell 80% of the time, in the eastern cell 10% of the time, and in the western cell 10% of the time.
  - **The dot stays put if it attempts to move into a solid cell** or outside the world. (Imagine the map is surrounded by solid cells)
- Goal: As always, maximize the sum of discounted future rewards within an episode



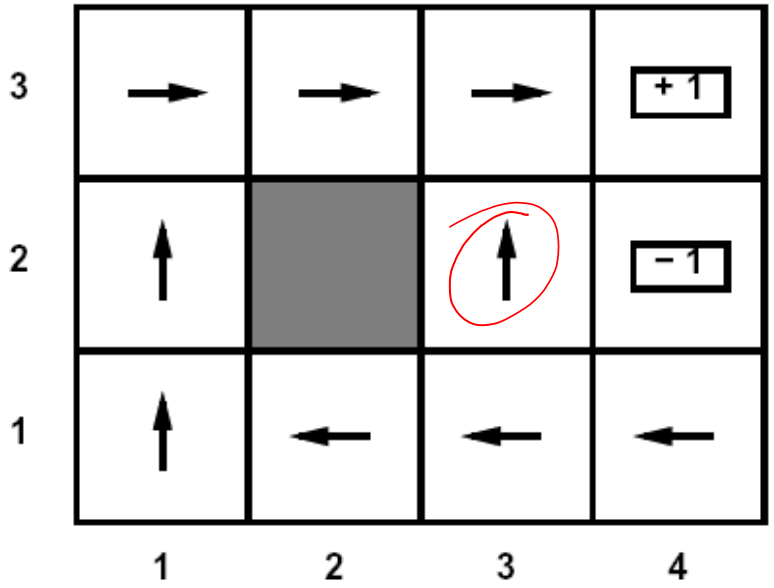



# Desired Outcome of RL: Optimal Policies

Goal: given some environment, find the optimal policy $\pi^*(s): S \rightarrow A$

- "Optimal" $\Longrightarrow$ Following $\pi^*$ maximizes expected return $\sum_t \gamma^t r_{t+1}$

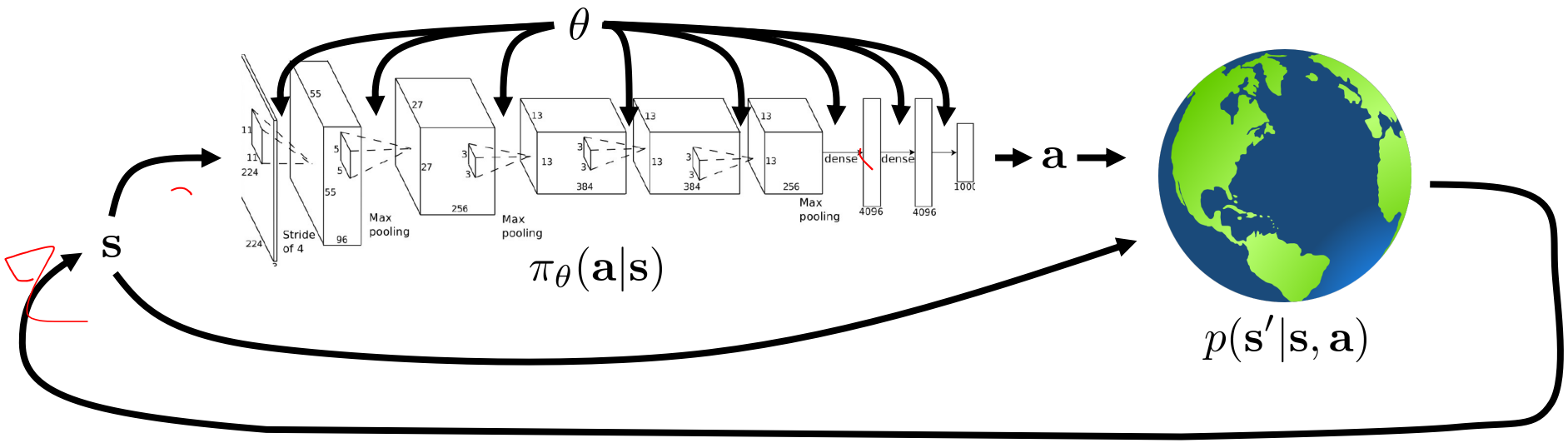
Example optimal policy $\pi^*$




Optimal policy when living cost is
$R(s, a, s') = R(s) = -0.03, \gamma = 1.0$
for all non-terminal states $s$

Based on slide by Dan Klein

# *Parametric* Optimal Policies With Continuous States




# How is RL Different from Supervised Learning (SL)?

SL: Find $h(x): X \rightarrow Y$, that minimizes a loss $L$ over training $(x, y)$ pairs

RL: Find $\pi(s): S \rightarrow A$ that maximizes expected return

## Supervised Learning

- **Training dataset:** curated in advance
- **Labels:** Desired outputs labeled in advance

## Reinforcement Learning

- **Training dataset:** choose your own adventure. Trial-and-error learning.
- **Labels:** No direct action labels, just instantaneous rewards $r_t$ which we need to maximize the sum-over-time of.

Unlike supervised learning, RL can **find solutions that the problem specifier did not already know**!

# Key Problems Specific to RL

In Online RL, we are trying both to gather training data, as well as to optimize policies based on the data we have observed. This leads to unique issues.

- **Exploration vs Exploitation**: Yes, trial-and-error, but what should you try?
  - If only we knew the optimal behaviors, we could go and gather data right around there (this is, in some sense, the imitation learning solution)
  - But you generally don't start out knowing the optimal policy.
- **Credit assignment**: Which actions in a sequence were the good/bad ones?
  - E.g. at what point did you falter in baking a complex cake, and which steps were good?

# Policy Gradients

# Getting Back to The Real Objective

$$\mathbb{E}_{\tau_i \sim \pi_\theta(\tau)} \left[ \sum_t \gamma^t r_{i,t} \right]$$

Trajectories obtained by rolling out the policy

Discounted sum of future rewards within that policy-generated trajectory

Note that this policy training objective is expressed as an expectation **over data generated by the policy**

$\pi_\theta$ induces a trajectory distribution, which induces a reward distribution.

# Recall: BC and Reward-Weighted Regression

BC gradient:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)$$

Optimal demonstration data

Optimal actions

Both objectives are expressed over pre-recorded data rather than policy-generated data!

Reward-weighted regression gradient:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\left(\sum_{\tau=t,t+1,\ldots}\gamma^{\tau-t}\, r_{i,\tau}\right)\right)$$

(Non-optimal) demonstrated data

(Non-optimal) demonstrated actions

Reward returns: "how good was this action?"

# “Vanilla Policy Gradient”

It turns out that the gradient $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\sum_t \gamma^t r_t]$ works out to:

$$\mathbb{E}_{\tau_i \sim \pi_\theta(\tau)} \left[ \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left( \sum_{\tau=t,t+1,\ldots} \gamma^{\tau-t} r_{i,\tau} \right) \right]$$

Policy-generated trajectories

Policy-generated actions

Reward returns: “how good was this action?”

**We generate our own data during learning … this is trial-and-error learning!**

**“Make good stuff more likely, and bad stuff less likely”**

(Proofs if time)

Policy Gradient Algorithms | Lil'Log

Williams 1992 REINFORCE

Sutton, McAllester, Singh, Mansour 1999

Peters & Schaal, Motor skills with policy gradients, 2008

$$\mathop{E}_{\tau \sim \pi_\theta(\tau)} \left[ r(\tau) \right] \qquad \sum_t \gamma^t r_{t+1}$$

unknown

$$\pi_\theta(\tau) = \mu(s_0) \, \pi_\theta(a_0|s_0) \, P(s_1|s_0, a_0) \, \pi_\theta(a_1|s_1) \, P(s_2|s_1, a_1) \cdots \pi_\theta(a_n|s_n) \, P(s_{n+1}|s_n, a_n)$$

unknown.

$$J(\theta) = \underset{\tau \sim \pi_\theta(\tau)}{\mathbb{E}}\left[ r(\tau) \right] = \int \pi_\theta(\tau)\, r(\tau)\, d\tau$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau)\, r(\tau)\, d\tau$$

$$= \int \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\, r(\tau)\, d\tau$$

$$= \underset{\pi_\theta(\tau)}{\mathbb{E}}\left[ \quad \right]$$

$$f(\theta)\, \nabla_\theta \log f(\theta) = \frac{1}{f(\theta)}\, \nabla_\theta f(\theta)$$

# “Vanilla Policy Gradient”: Monte Carlo Estimate

$$\mathbb{E}_{\tau_i \sim \pi_\theta(\tau)} \left[ \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \left( \sum_{\tau=t,t+1,\ldots} \gamma^{\tau-t} r_{i,\tau} \right) \right]$$

Estimated through sampling trajectories $\tau$ by executing policy $\pi$ in the environment.

$$\frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \left( \sum_{\tau=t,t+1,\ldots} \gamma^{\tau-t} r_{i,\tau} \right) \right)$$

Policy-generated trajectories

Policy-generated actions

Reward returns: “how good was this action?”

**Good estimates often require large $N$!**

Policy Gradient Algorithms | Lil'Log

Williams 1992 REINFORCE

Sutton, McAllester, Singh, Mansour 1999

# Vanilla Policy Gradient vs. Reward-Wtd Regression

Reward-weighted regression gradient:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_{i,t}\middle|s_{i,t}\right)\left(\sum_{\tau=t,t+1,\ldots}\gamma^{\tau-t}\, r_{i,\tau}\right)\right)$$

(Non-optimal) demonstrated data

(Non-optimal) demonstrated actions

Reward returns: "how good was this action?"

Policy gradient:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(a_{i,t}\middle|s_{i,t}\right)\left(\sum_{\tau=t,t+1,\ldots}\gamma^{\tau-t}\, r_{i,\tau}\right)\right)$$

Policy-generated trajectories

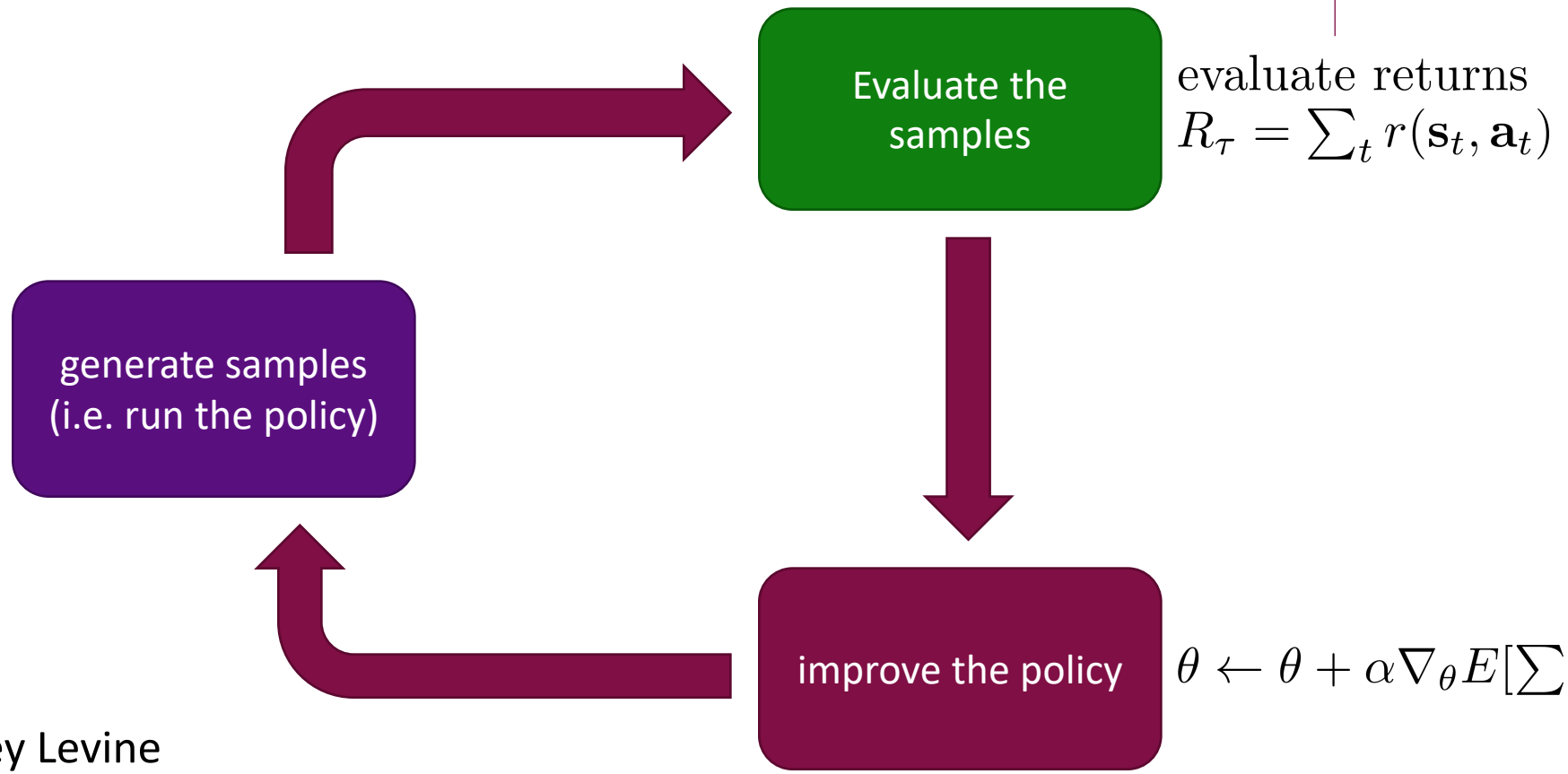
Policy-generated actions

Reward returns: "how good was this action?"

# The basic policy gradients algorithm: REINFORCE

REINFORCE algorithm:

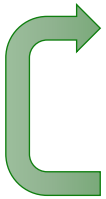
1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$




Slide: Sergey Levine

# Reward function need not be differentiable!

REINFORCE algorithm:



1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

In supervised learning, when we optimized an objective using gradient descent, we needed the objective to be differentiable w.r.t. to the parameters $\theta$.

In RL, this is not true any more. See how the update term involves no derivative of the reward function!

2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$

# “On-Policy” Learning

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left( \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

- The policy gradient increases the likelihood of those past actions that yielded good future returns ***when later actions were generated from the current policy.***
- This means you can only ever compute the policy gradient update on data that is generated *from the current policy.*
  - “On-policy” learning.
  - Expensive in terms of amount of experience required in the environment, because old experience, generated from old policies, is no longer relevant. Need to keep generating fresh new experiences.
  - Also, recall, large $N$ is required for getting good gradient estimates i.e. *lots of fresh new experiences.*

# The “Sample Complexity” of Online RL

- Recall that in supervised learning, we often updated a neural network tens of millions of times on small mini-batches to find a solution.
- Online RL requires *generating a whole new dataset for each update*, generated by executing the current policy many times.
  - **Often too sample-inefficient to run on real-world robots, and instead deployed in highly parallelized simulators.** More on this when we discuss “sim-to-real RL” methods later in the course.
- Gradient descent works best when step sizes are small, but if each gradient step is very expensive (as in online RL), one might consider more aggressive updates. Tweaking learning rates like one might in supervised learning doesn’t work too well.
  - **TRPO** [Schulman et al 2015], **PPO** [Schulman et al 2017], **etc.** propose ways to set the step size at each update as large as possible without breaking optimization.

# Whither Exploration?

- **Exploration in RL**: Which actions to execute in the world to most efficiently learn an optimal policy?
  - But with on-policy RL, do we really have a choice? Remember, our updates can only be computed from trajectories sampled from the current policy $\pi_\theta$ at each stage of training!
- **Two solutions:**
  - $\pi_\theta$ is inherently stochastic, because it is probabilistic, so it does automatically perform different actions each time it is executed, and therefore induces some exploration.
  - Explicitly add an **"exploration bonus"** to the reward, e.g. entropy

$$r_t \leftarrow r_t + \lambda H\big(\pi_\theta(a_t|s_t)\big)$$

which incentivizes more uncertain policies, inducing more exploration. $\lambda \to 0$ during training.

# Popular Implementations

- RL implementation details can be hard to get right. Good to start with **popular repositories: OpenAI stable-baselines, CleanRL etc.**

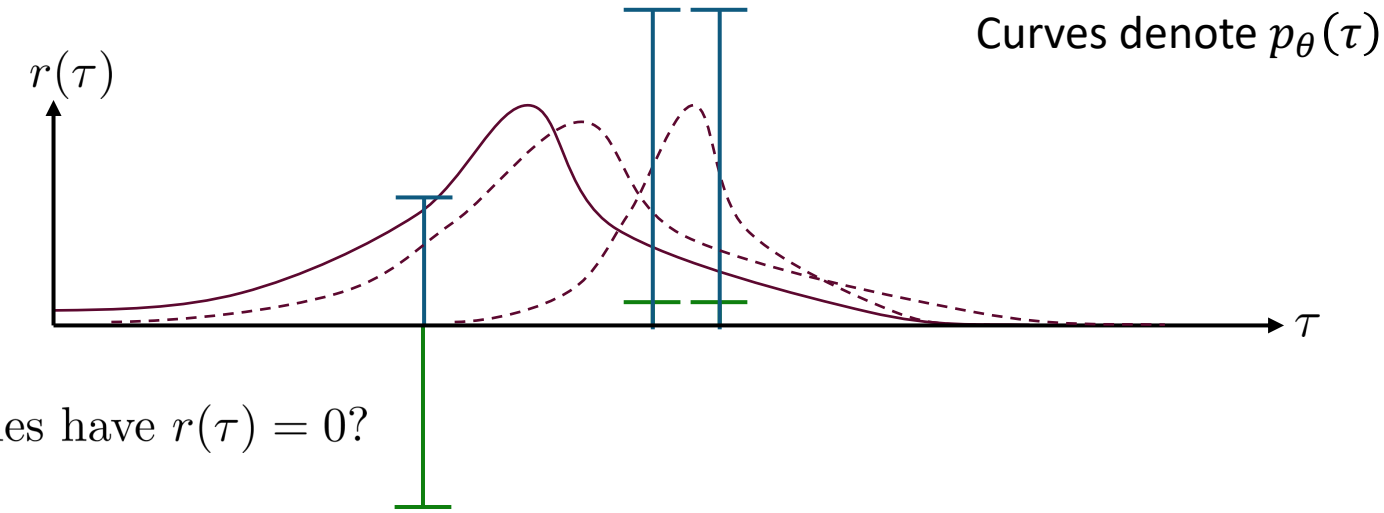
# Reducing Policy Gradient Variance with Baselines

# Sensitivity To Constant Reward Offsets

Consider what should happen if rewards for an MDP were all incremented by a constant value $r$. Should the optimal policy change? Should a policy gradient change?

No to both, but look at the current policy gradient!

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$




even worse: what if the two "good" samples have $r(\tau) = 0$?

# Baselines

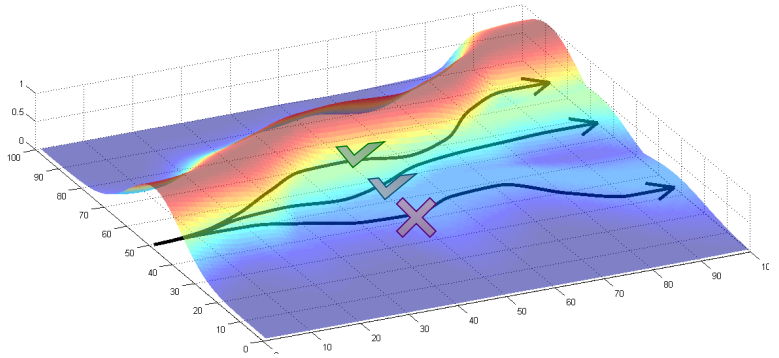
a convenient identity

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

but… are we *allowed* to do that??



$$E[\nabla_\theta \log \pi_\theta(\tau) b] = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) b \, d\tau = \int \nabla_\theta \pi_\theta(\tau) b \, d\tau = b\nabla_\theta \int \pi_\theta(\tau) d\tau = b\nabla_\theta 1 = 0$$

subtracting a baseline is *unbiased* in expectation

- Keeps expectation fixed, but reduces the variance of the policy gradients!

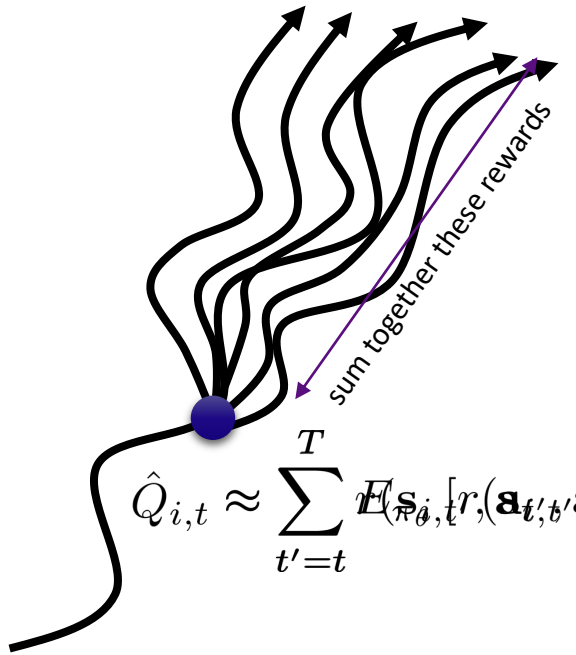
average reward is *not* the best baseline, but it's pretty good!

# Improving the policy gradient "critic"

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\sum_{t'=1}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)$$

"reward to go"

$\hat{Q}_{i,t}$




$\hat{Q}_{i,t} \approx \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]$

$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

can we get a better estimate?

$Q(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t]$: true *expected* reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})(Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t}))$$

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t,\mathbf{a}_t)]$$

# State & state-action value functions

Next up: Value function learning!

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$: total reward from taking $\mathbf{a}_t$ in $\mathbf{s}_t$

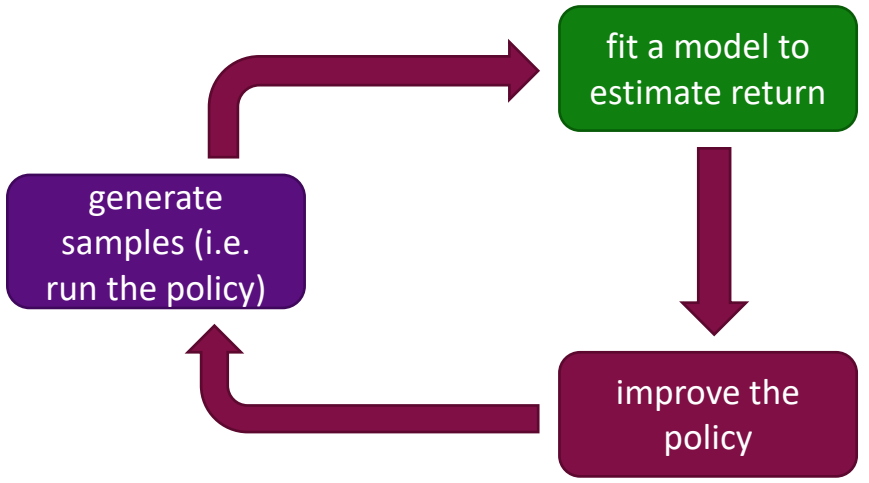
$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$: total reward from $\mathbf{s}_t$

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how much better $\mathbf{a}_t$ is

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

the better this estimate, the lower the variance

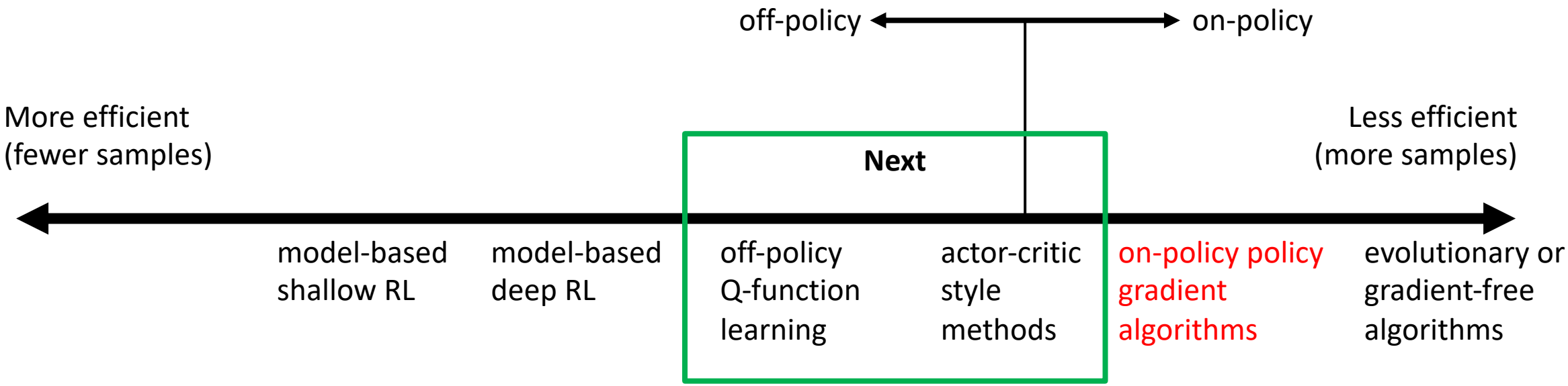
fit $Q^\pi$, $V^\pi$, or $A^\pi$




$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

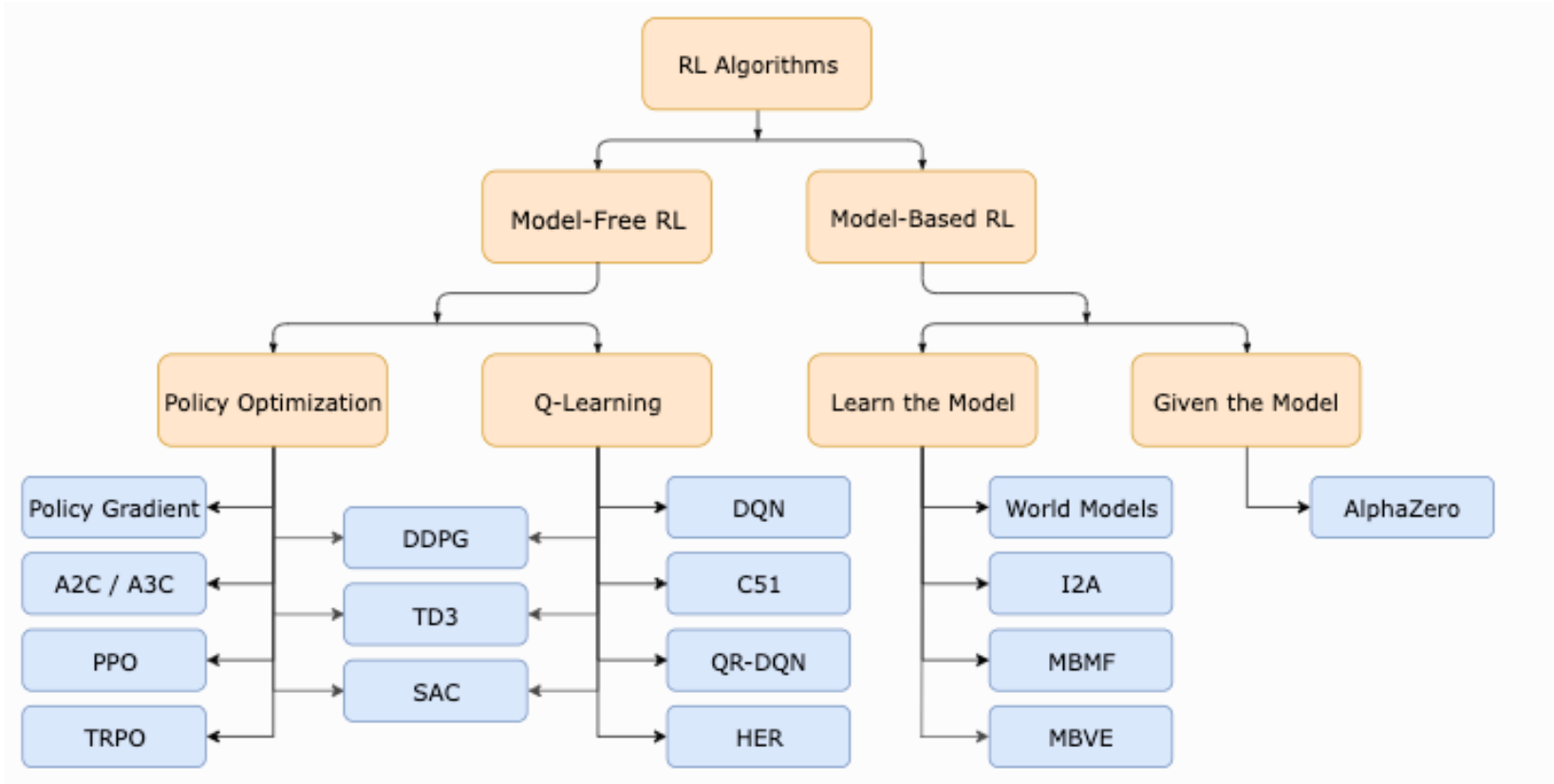
# The Landscape of RL Algorithms

# Other RL Algorithms




But policy gradients are among the most *stable* approaches that work most broadly, and take limited wall clock time even though many samples.

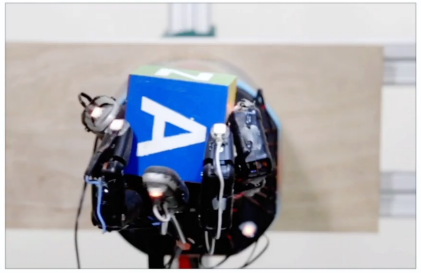
Next: Q learning and actor-critic approaches!


RL Algorithms
Model-Free RL
Model-Based RL
Policy Optimization
Q-Learning
Learn the Model
Given the Model
Policy Gradient
A2C / A3C
PPO
TRPO
DDPG
TD3
SAC
DQN
C51
QR-DQN
HER
World Models
I2A
MBMF
MBVE
AlphaZero

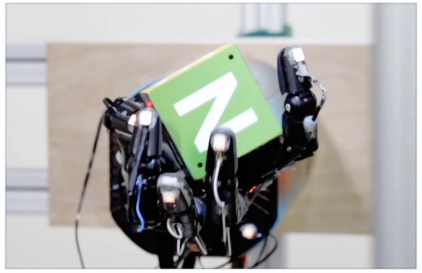
# Examples of RL On Robots

# Robotics

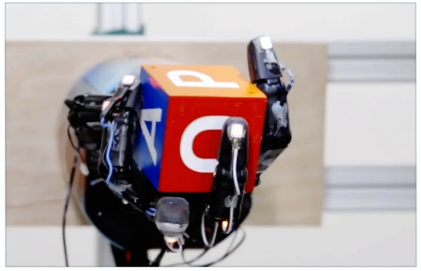
Robotics



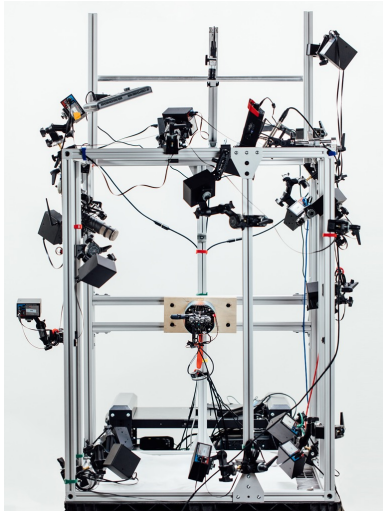
FINGER PIVOTING



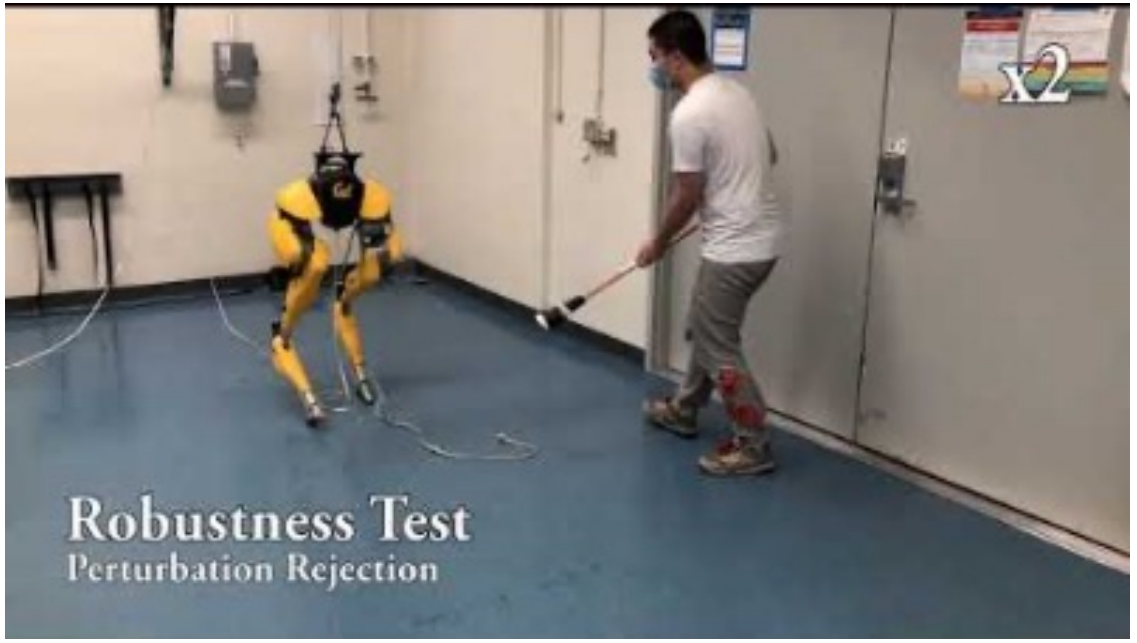
SLIDING



FINGER GAITING



Open AI Dactyl (2018)




Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots, 2022

# More RL for Robotics

- Guiness world record in 100 meters by biped robots (Oregon State University)
- Learned quadrupedal locomotion in challenging environments (ETH Zurich)
- Autonomous Navigation of Stratospheric Balloons (Google AI), blog (was real, just Google canceled the whole project.. sadly..)
- Not yet perching (article), but soon? Just for inspiration..
- Video games; car racing in video games, competing with humans

- Vision-based autonomous drone racing (video, UZH RPG)
- Commanding robots using natural language to perform tasks (SayCan project, Google)
- behavioral cloning/imitation learning (not RL) is doing well with transformers in the kitchen (Google)
- Yet it is not enough to learn to drive well
- Quadruped learns to walk in the park in 20 minutes, model-free (UC Berkeley)
    - More of this
- Still, dexterous manipulation is not easy.. (Berkeley, Meta, UW)
- Visual Navigation (Berkeley)
- In the need for resets (Berkeley)

Credit: Csaba Szepasvari

# RL For Robots: Challenges

- How sample-efficient and stable is RL optimization?
- Does it make sense to ignore that we may know dynamics / physics $P(s'|s,a)$?
  - Simulation, residual learning, robot-aware learning etc.
- Where do you get episode resets from?
  - Reset-free RL etc.
- Where do you get rewards from?
  - Learning from examples, demonstrations etc.
- Is it fair to treat time as discrete?
- Is it fair to treat $s_t, and\ a_t = \pi(s_t)$ as happening at the same instant?
  - Delay-aware methods

# RL For Robots: Challenges

- Partial observability: is the Markov state actually available to the agent?
  - No! This is particularly important in the context of this class!
- How are demonstrations provided?
- Will learning by trial and error damage my robot / other equipment / me?
- Do I have to learn from scratch for each robot?
- Non-stationarity, e.g. my robot deteriorates over time?

**How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned**

1–22

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

**Julian Ibarz[1], Jie Tan[1], Chelsea Finn[1,3], Mrinal Kalakrishnan[2], Peter Pastor[2], Sergey Levine[1,4]**

arxiv.org/pdf/2102.02915.pdf