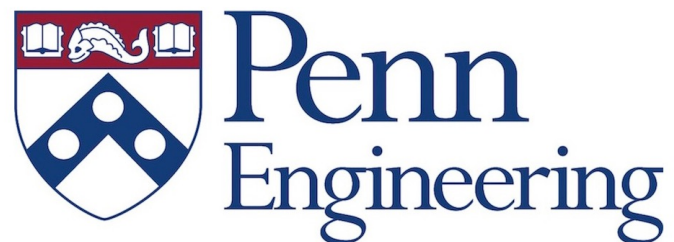


CIS 7000-04 / ESE 6800 Spring 2025: Intro To Imitation and Reinforcement Learning Part II

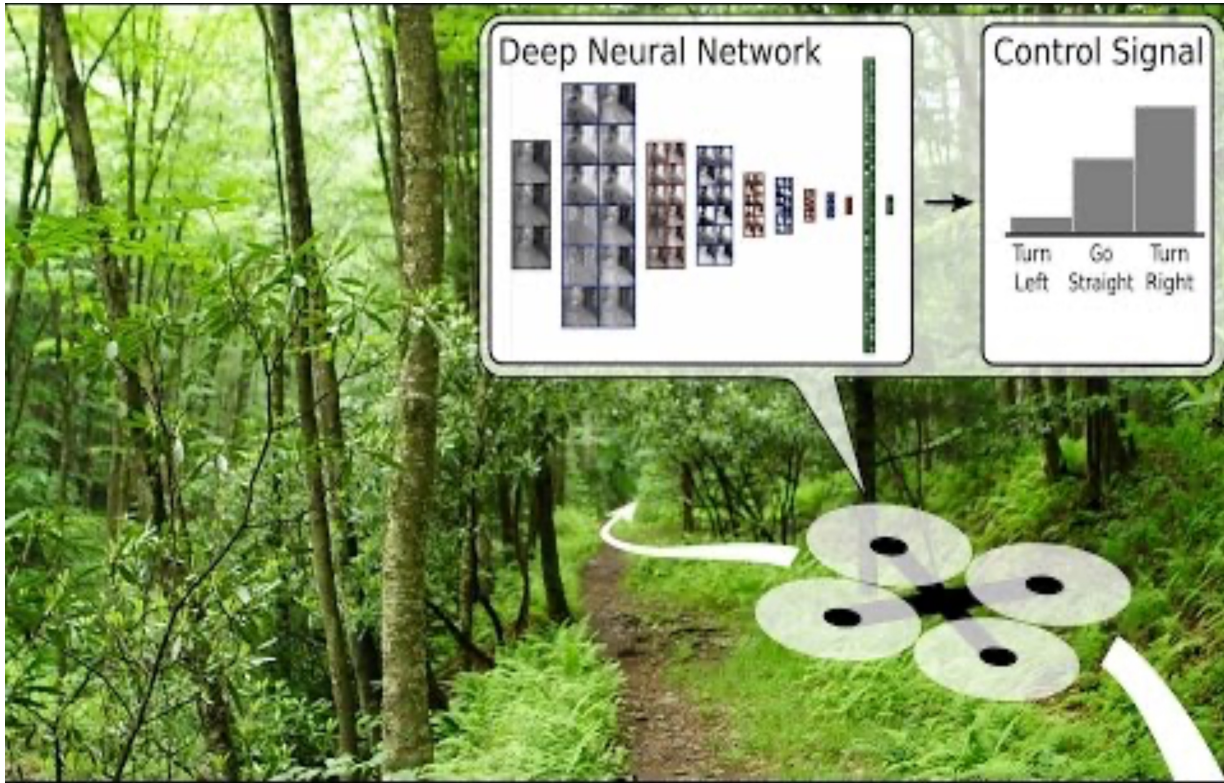
Thu, Jan 23, 2025

Instructor: Dinesh Jayaraman

Assistant Professor, CIS



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](#)

Recap: Where we left off

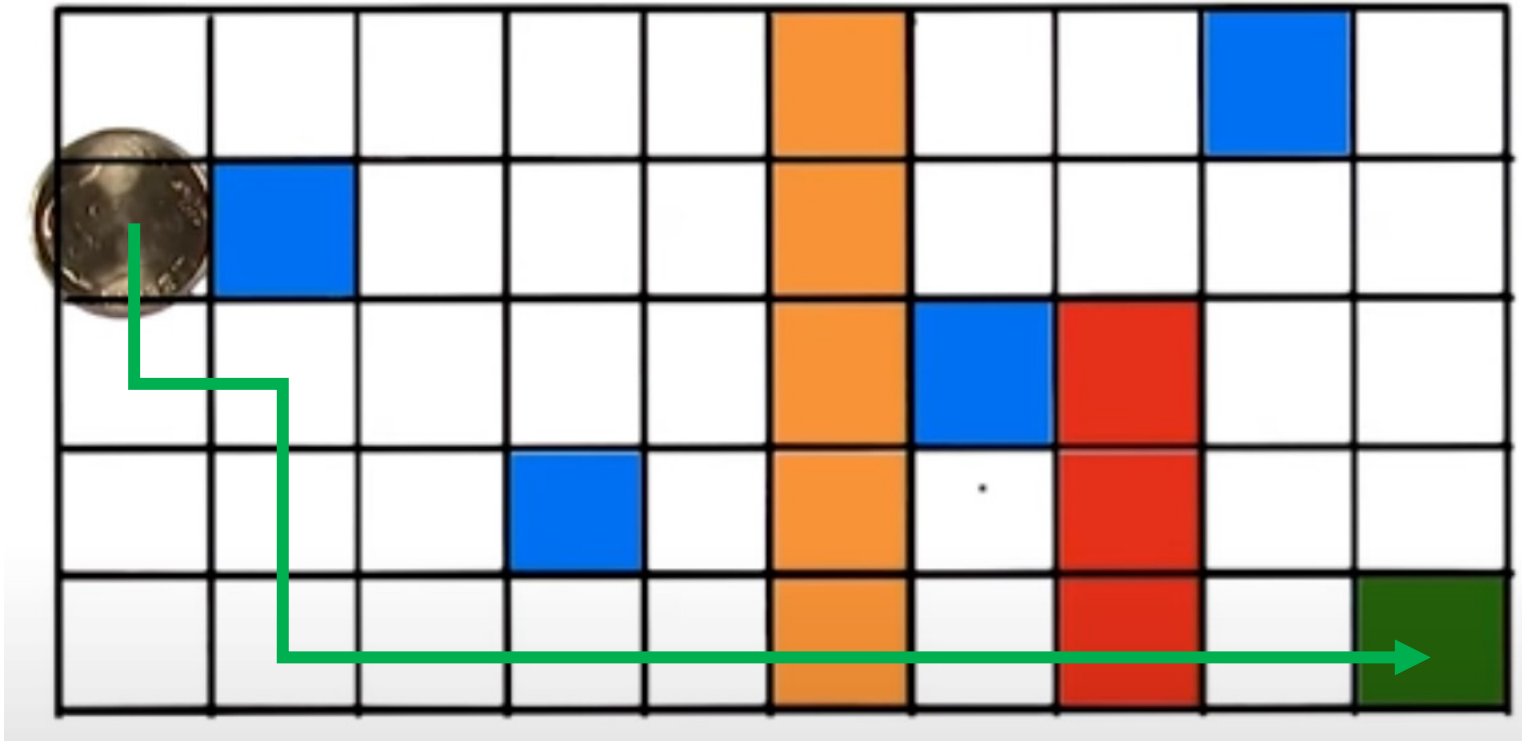
- Naïve BC treats policy learning as a supervised learning problem: “expert” provides demos first, and the learner treats the demo as an unordered bag of state-action pairs to learn $\pi(a_t|s_t)$ from.
 - Can be very useful already, but prone to distribution shift -> compounding errors
- Improvements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

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

(Non-optimal)
demonstrated data

(Non-optimal)
demonstrated actions

Reward returns: “how good was this action?” 

Injecting Reward Information into BC

Reward-weighted regression objective:

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

(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

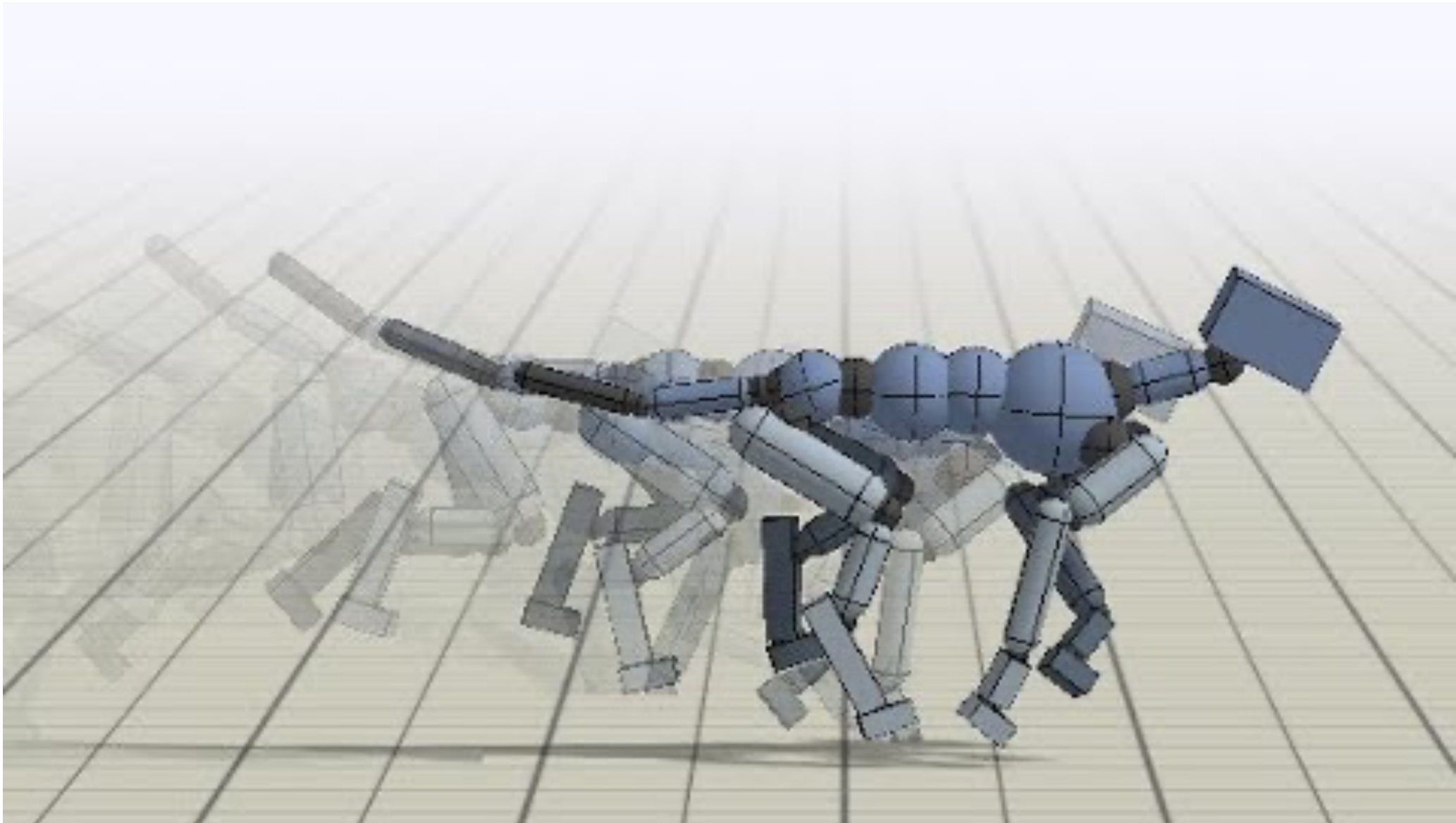
Led later to “advantage-weighted regression” [[Peng et al 2019](#)]

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.



Mulling, K.; Kober, J.; Kroemer, O.; Peters, J. (2013). Learning to Select and Generate Robot Table Tennis. International Journal on Robotics Research.

[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 ...



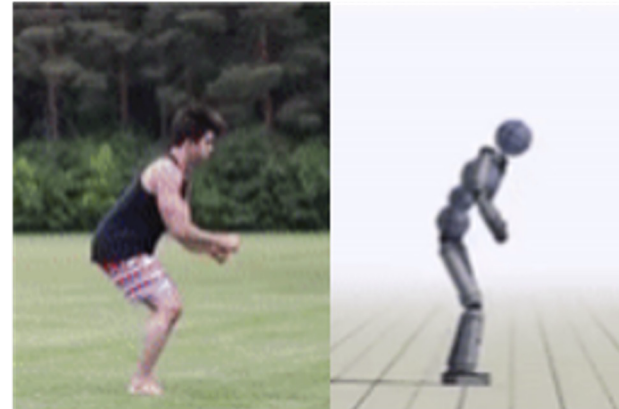
teleoperation



kinesthetic



motion capture

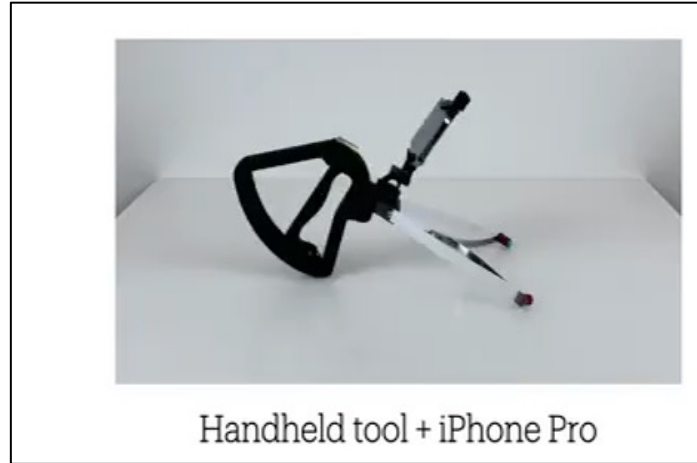


web video

Tools To Demonstrate Robot Behavior: Recent Progress



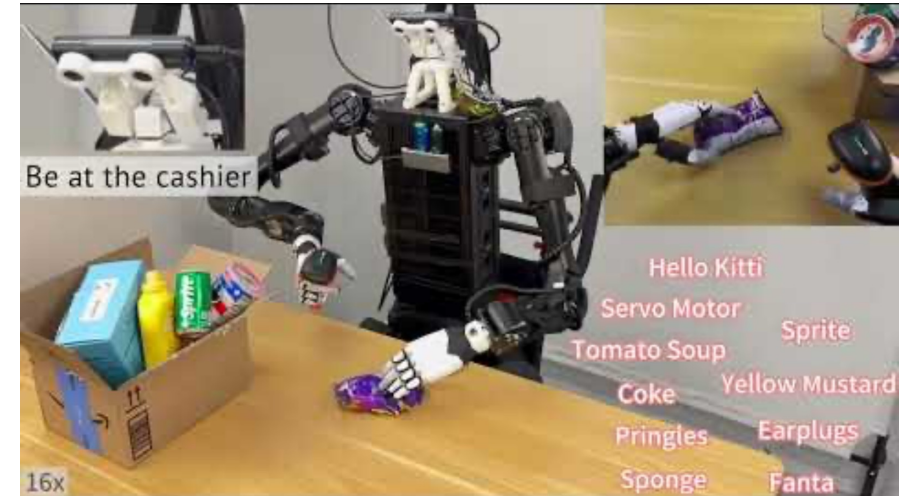
[UMI](#)



[Stick-V2 \(RUM\)](#)



[Mobile Aloha](#)



[Open-TeleVision](#)

Do we have enough robot data?

Comparing datasets

Assuming 238 words/minute, 1.33 tokens/word

OXE
4k hours

GPT-2
475k hours

π data
10k hours

Llama 3
790m hours

(π)

Some Key Limitations of Imitation For Robots

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

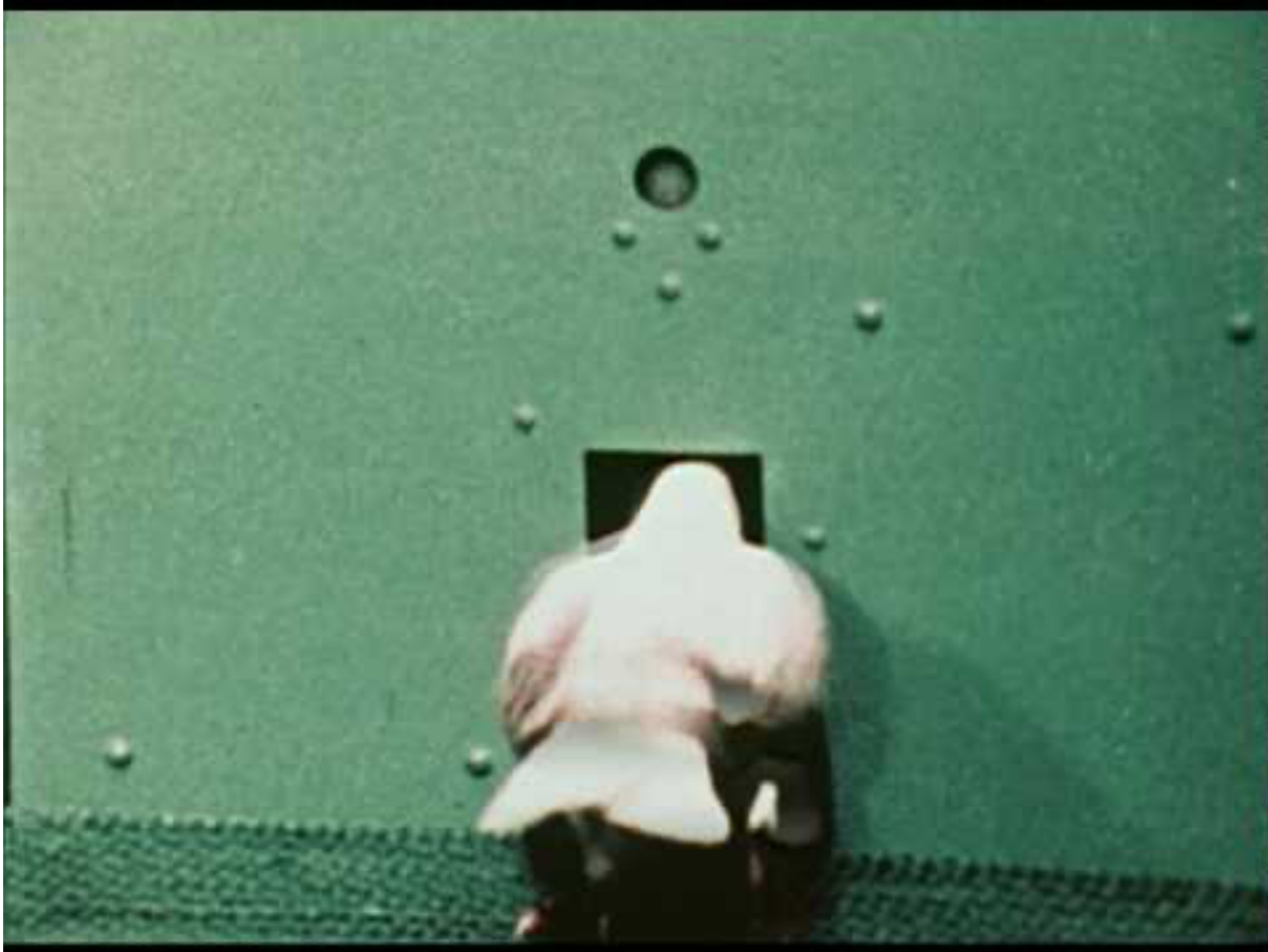
More on these in detail when we get around to the module on robot imitation later in the semester

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.

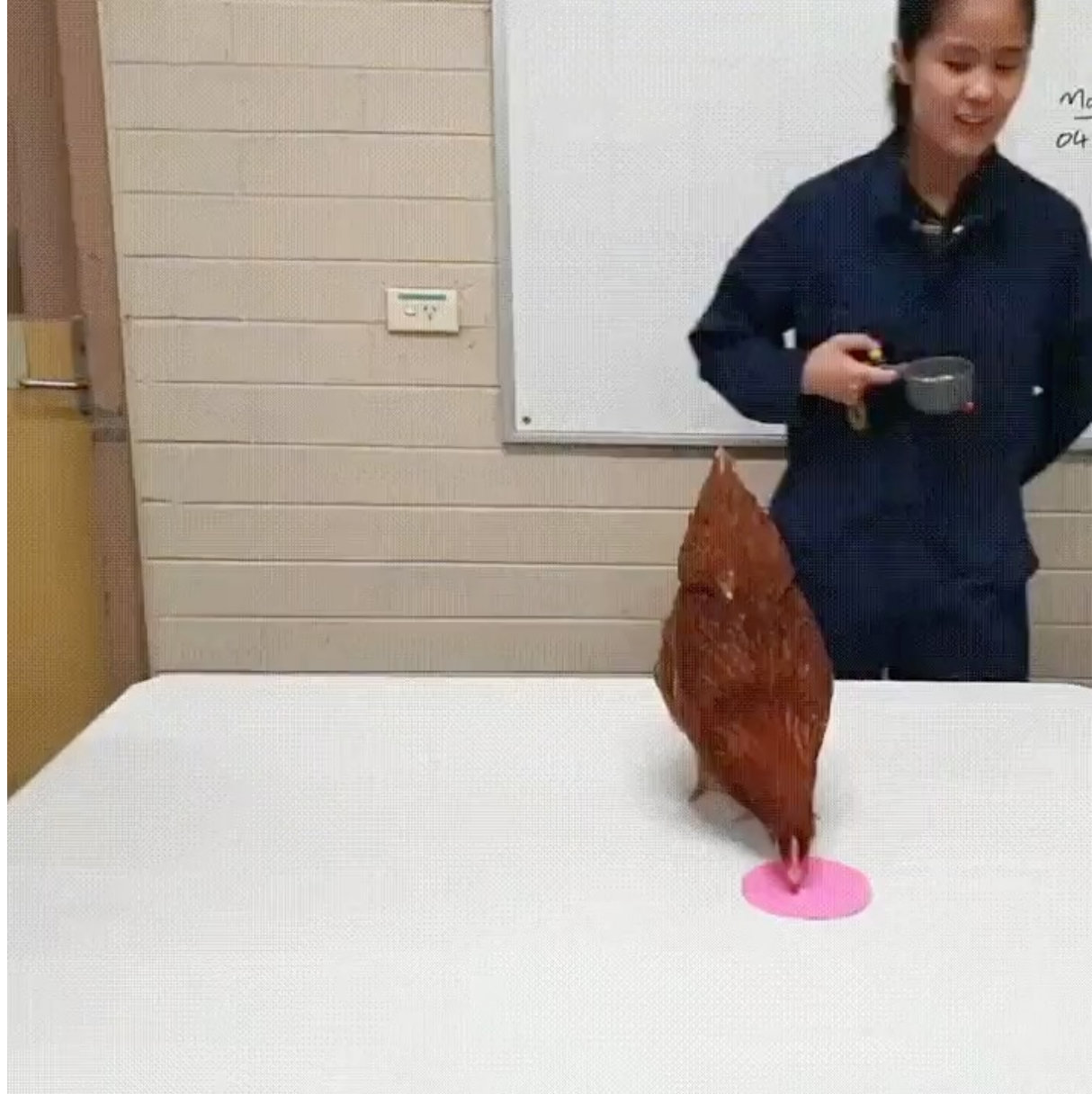
[Online] Reinforcement Learning

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



(Go to ~ 0:30)

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?

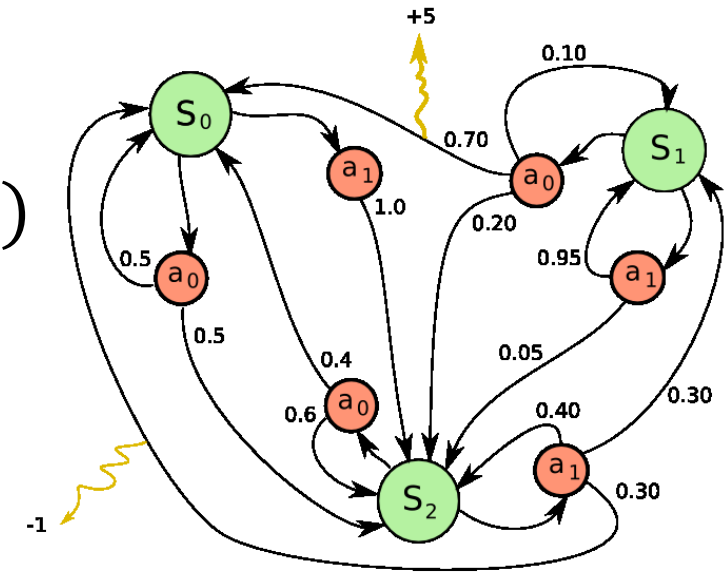
- Assume only occasional feedback, such as a tasty meal, or a car crash, or video game points.
- Assume very little is known about the “environment” in advance.
- Learn through trial and error.

Recap: MDP Formulation Notations

An MDP (S, A, P, R, γ) is defined by:

- Set of states $s \in S$
- Set of actions $a \in A$
- Transition function or “dynamics model” $P(s' | s, a)$
 - Probability $P(s' | s, a)$ 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



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. $\text{return} = \sum_t \gamma^t r_{t+1}$
 - 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” \leftrightarrow “return”
- Sometimes “return” is by default undiscounted, etc.

Online RL is hard!

Entering An Unknown Gridworld

In the shoes of an Online RL agent

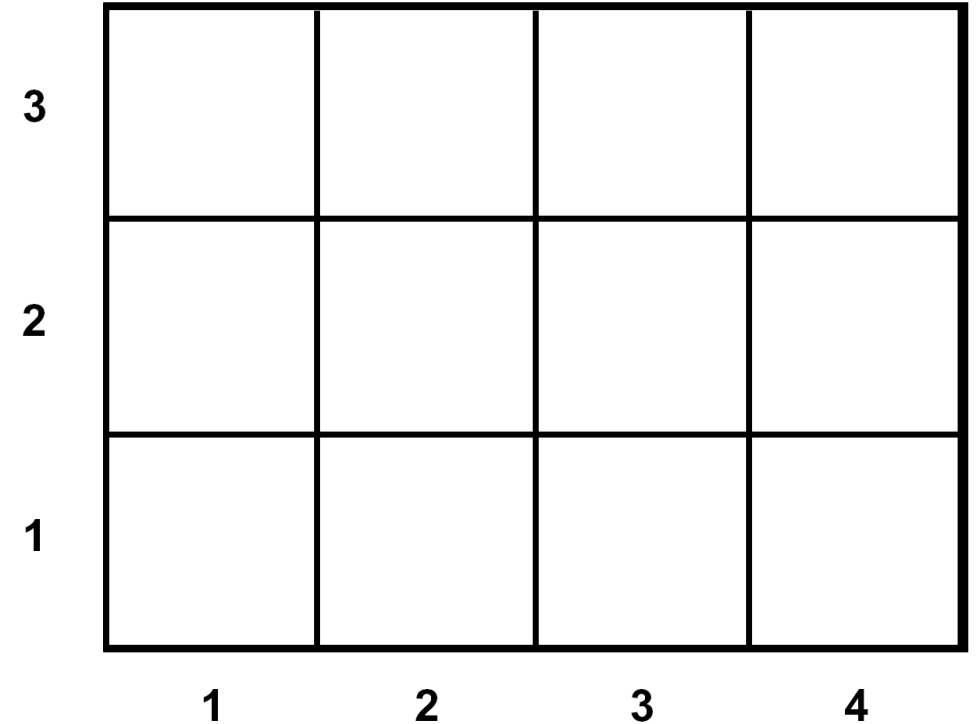


Sample RL environment: Grid World

- 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: $a = \text{"W", "E", "S", "N"}$

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

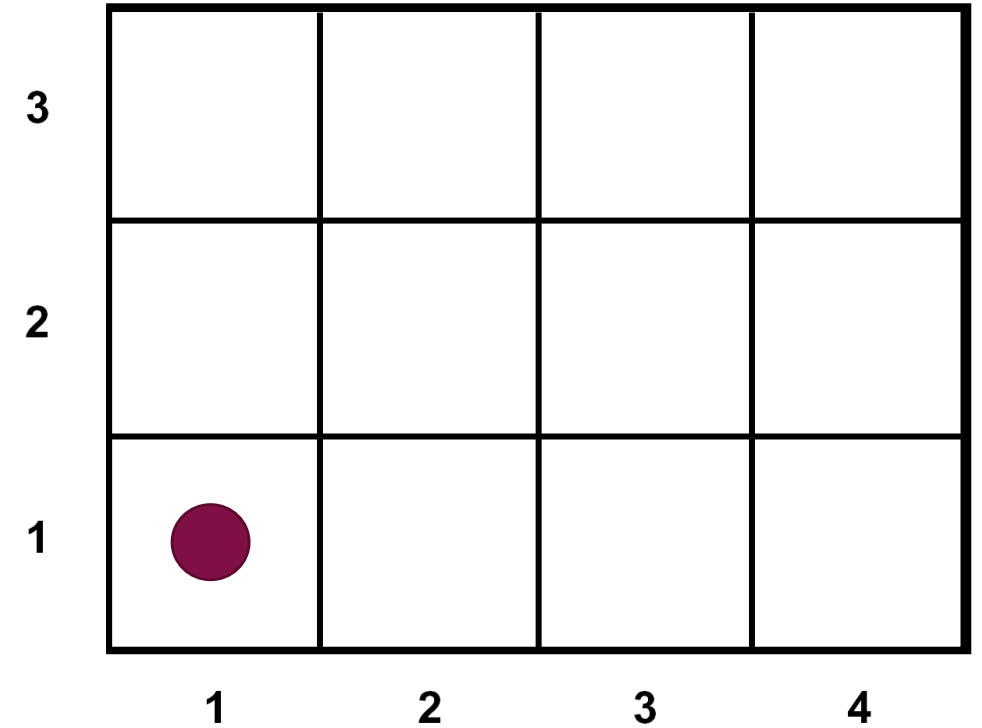
- $P(s'|s, a)$
 - Which cell would it move to, if it executes an action from a cell? (e.g. $a = \text{"N"}$ from $s = (1, 2)$)
 - The result might even be non-deterministic.
- $R(s, a, s')$
 - What is the instantaneous reward it would get if it moved from $s = (1, 2)$ to $s' = (1, 3)$ by executing action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



A random trajectory of an RL agent

Time $t=0$

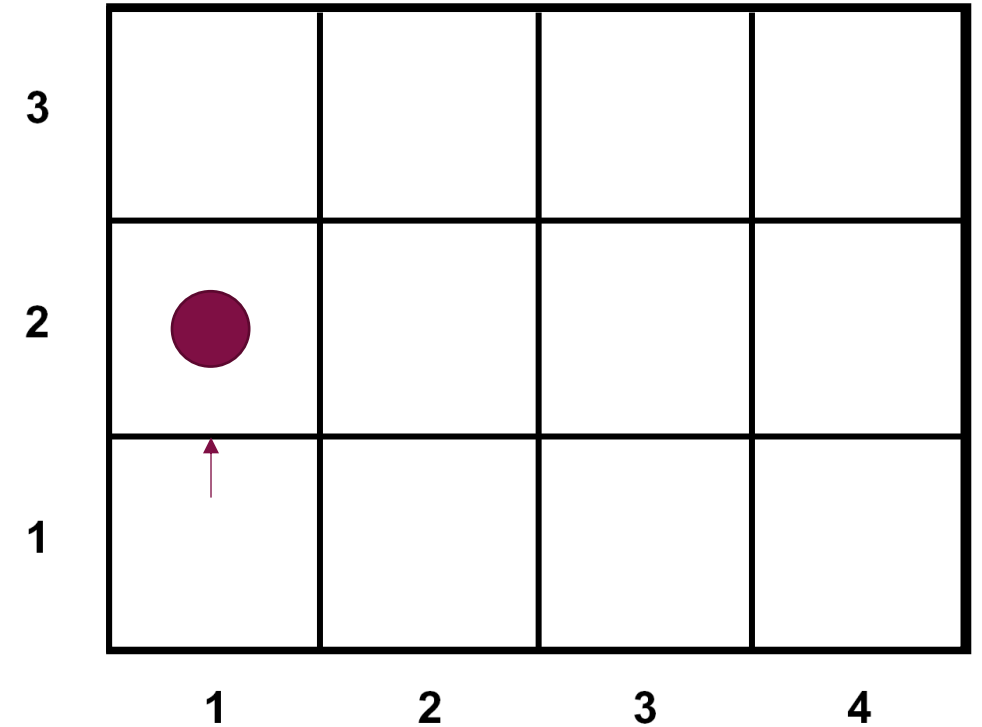
$s=(1,1)$
Action= "N"



A random trajectory of an RL agent

Time $t=0$

$s=(1,1)$
Action= "N"
 $s'=(1,2)$
Reward = -0.03

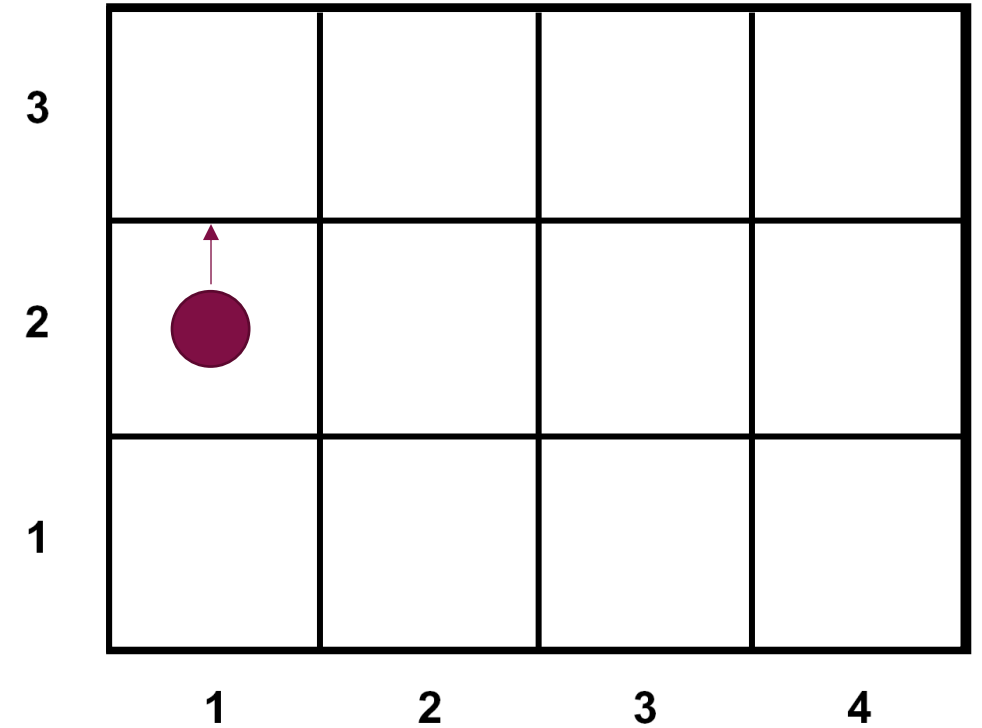


Time step $t=0$ over

A random trajectory of an RL agent

Time $t=1$

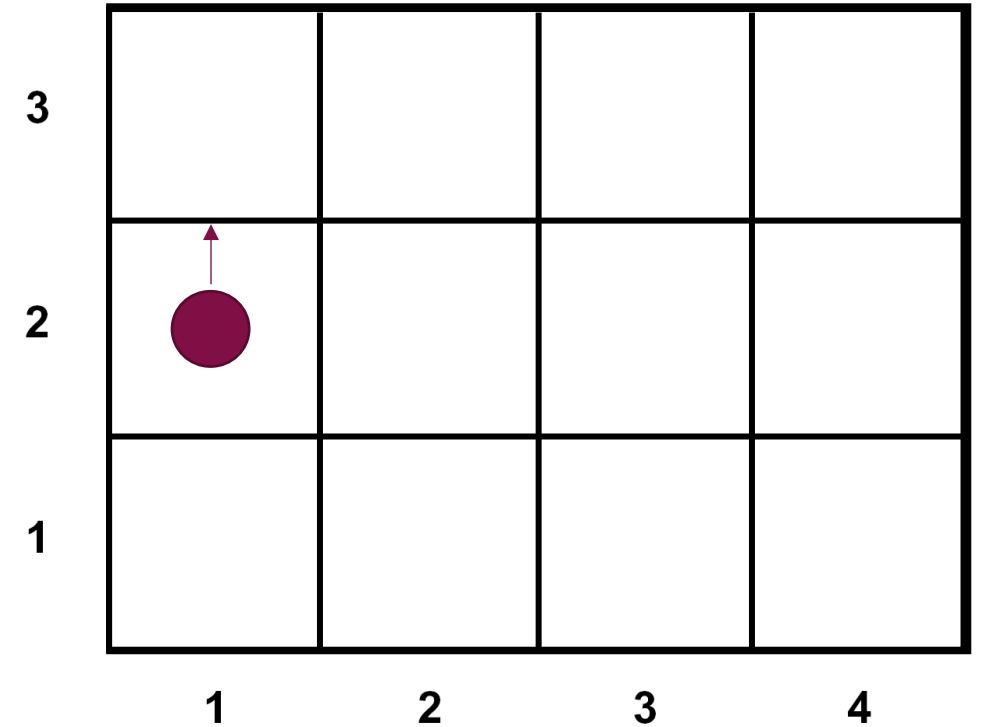
$s=(1,2)$
Action= "N"
 $s'=?$
Reward = ?



A random trajectory of an RL agent

Time $t=1$

$s=(1,2)$
Action= "N"
 $s'=(1,2)$
Reward = -0.03

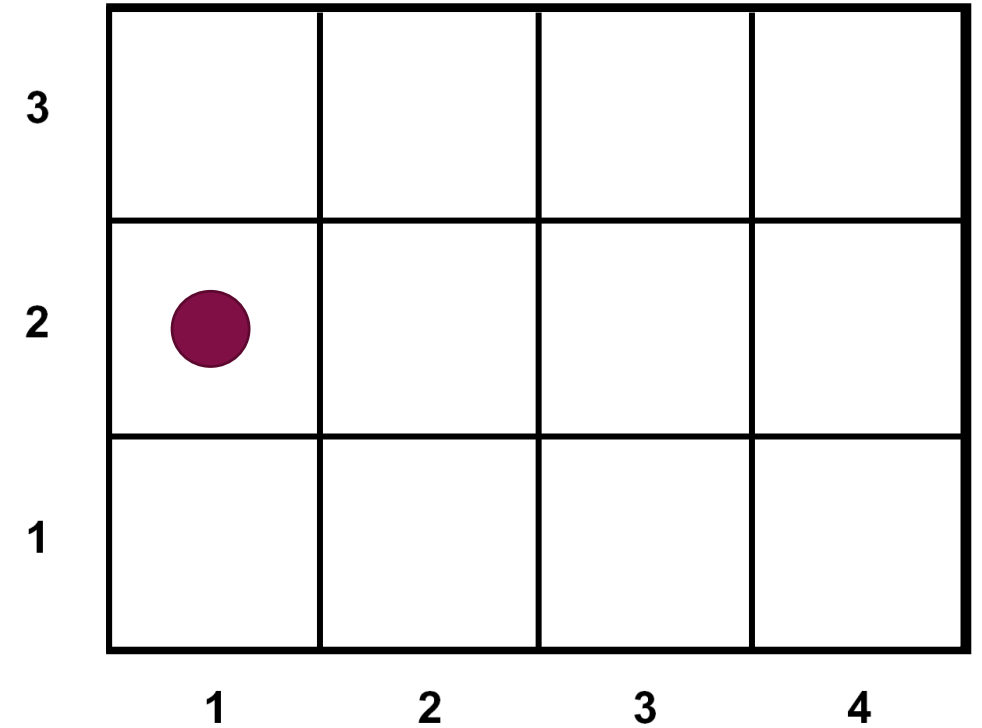


Time step $t=1$ over

A random trajectory of an RL agent

Time $t=2$

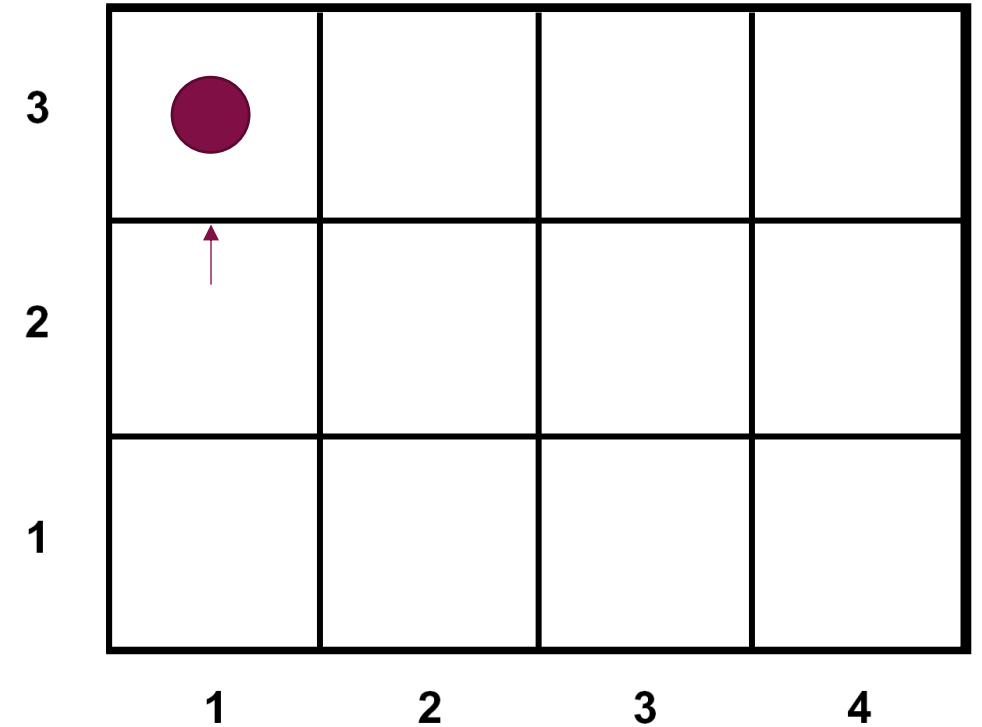
$s=(1,2)$
Action= "N"
 $s'=?$
Reward = ?



A random trajectory of an RL agent

Time t=2

$s=(1,2)$
Action= "N"
 $s'=(1,3)$
Reward = -0.03

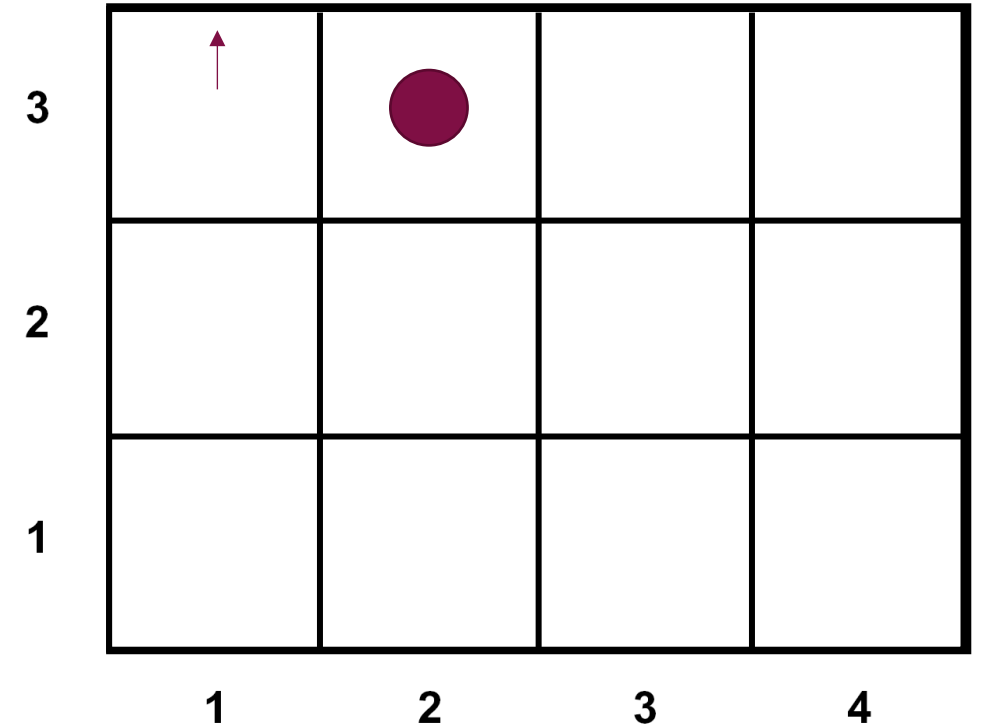


Time step t=2 over

A random trajectory of an RL agent

Time $t=3$

$s=(1,3)$
Action= "N"
 $s'=(2,3)$
Reward = -0.03

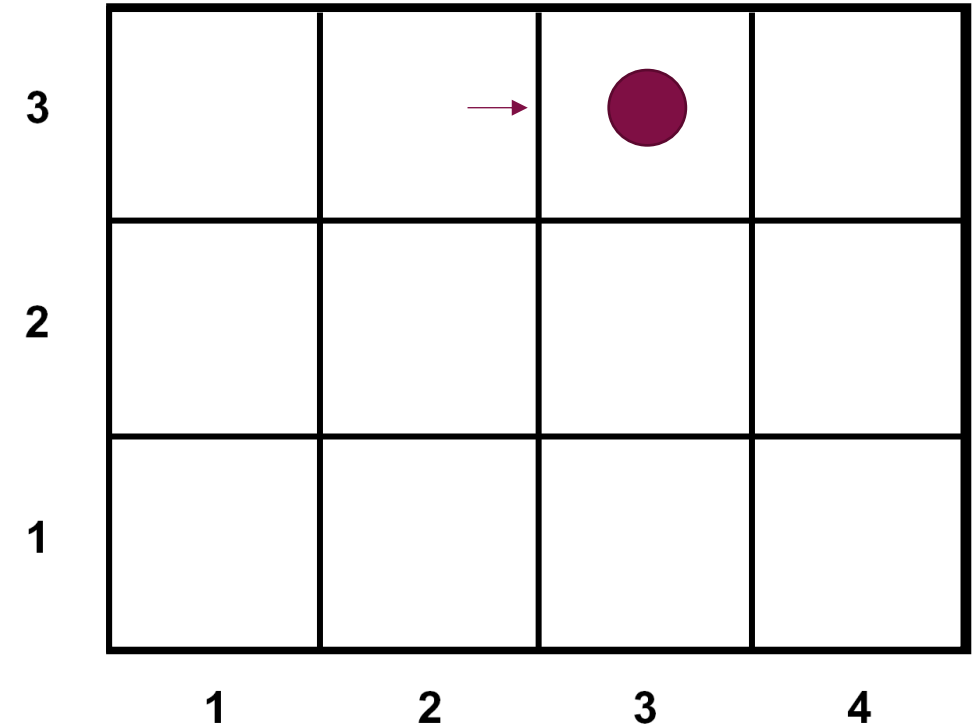


Time step $t=3$ over

A random trajectory of an RL agent

Time t=4

$s=(2,3)$
Action= "E"
 $s'=(3,3)$
Reward = -0.03

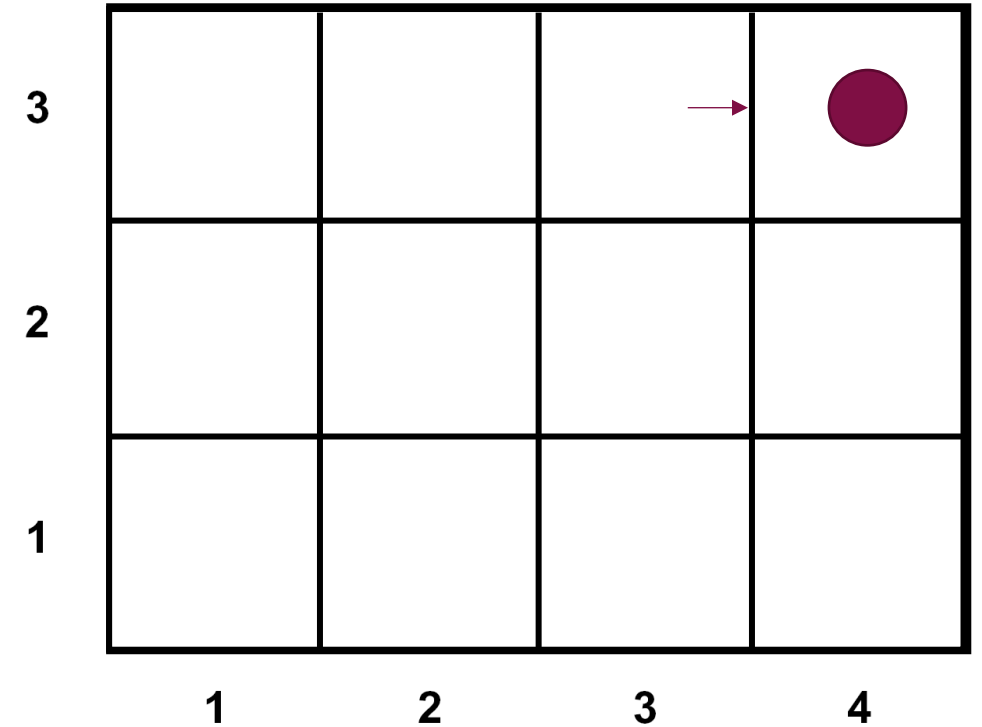


Time step t=4 over

A random trajectory of an RL agent

Time t=5

$s=(3,3)$
Action= "E"
 $s'=(4,3)$
Reward = -0.03

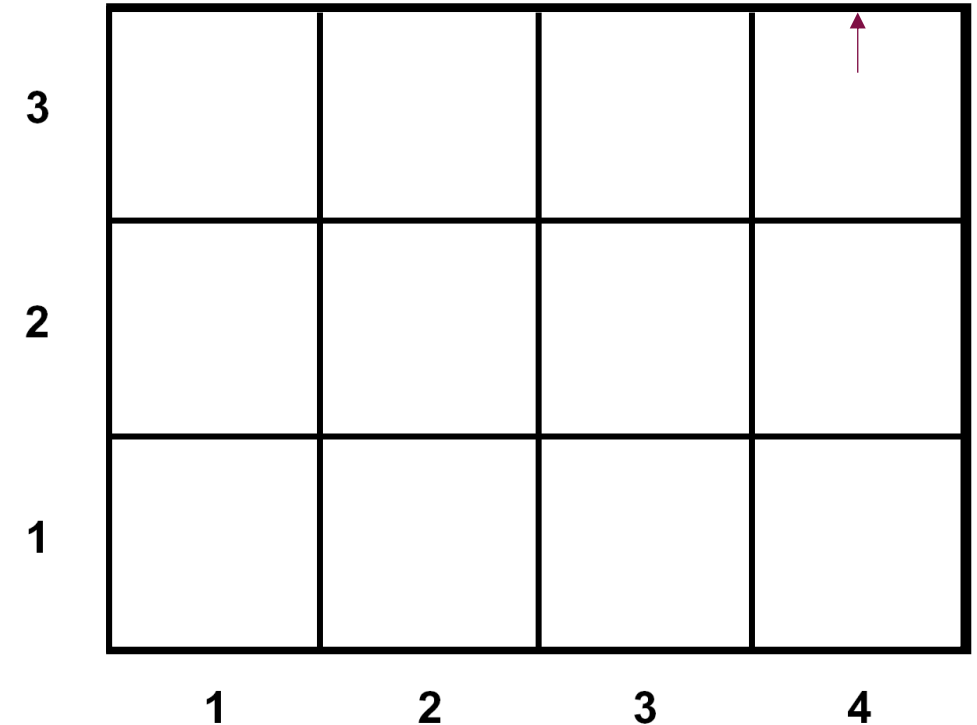


Time step t=5 over

A random trajectory of an RL agent



$s=(4,3)$
Action= "N"
 s' = special state "END"
Reward = +1



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

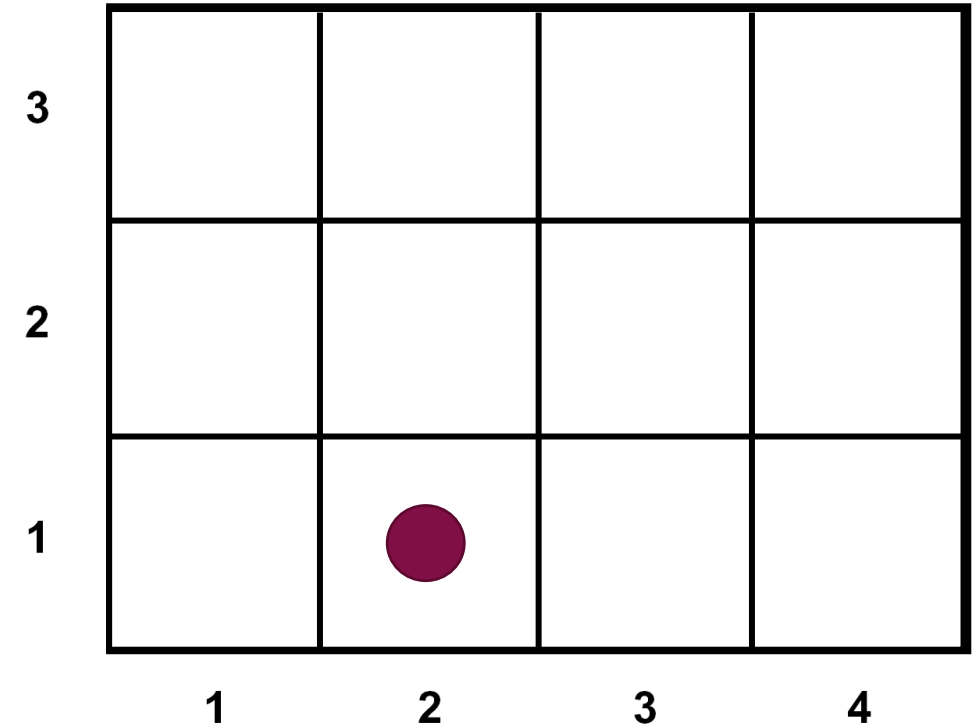
Next, the agent respawns in the environment. “Reset”

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, γ) , there may also be an “initial state probability distribution” μ 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.

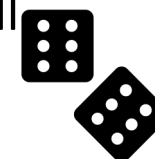
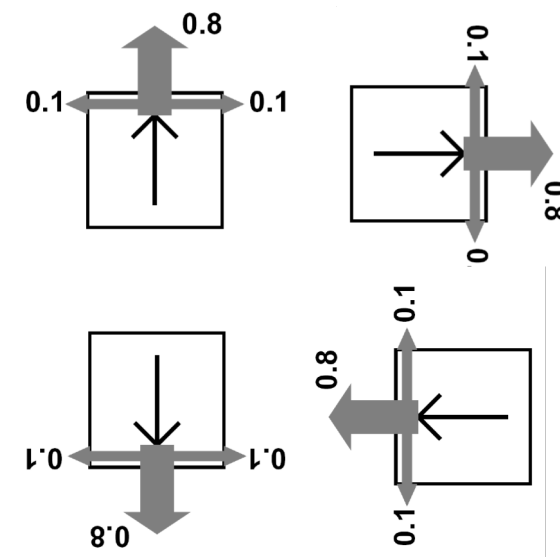
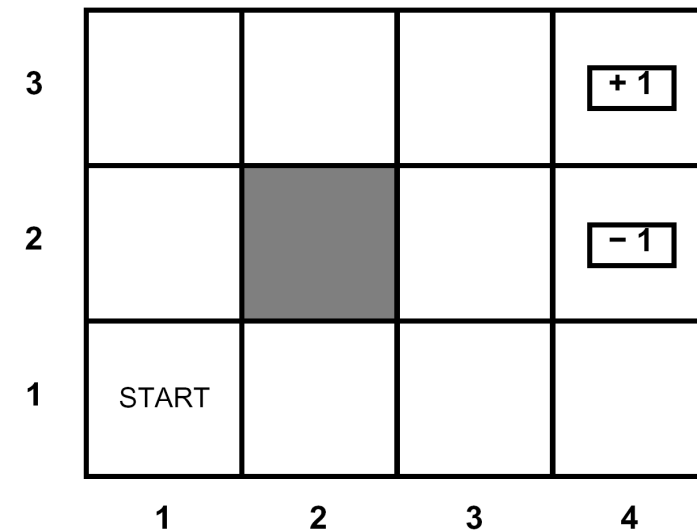
Perhaps with more experience ...?

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

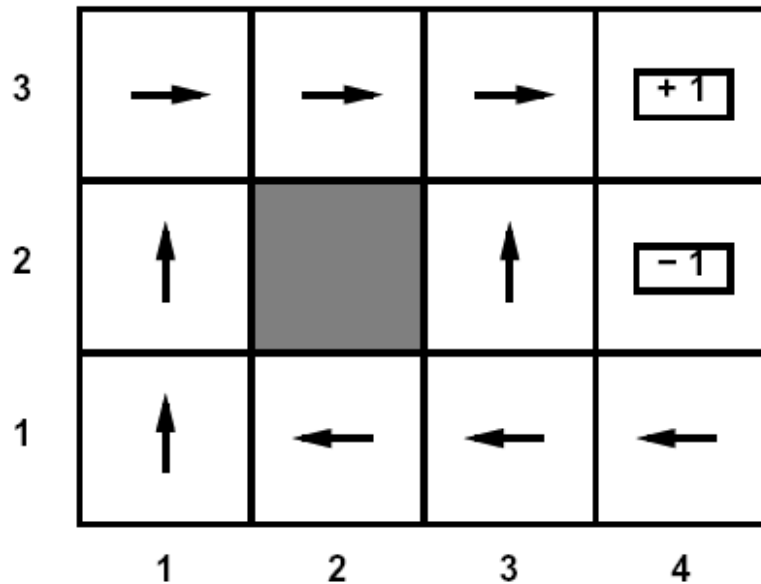


Desired Outcome of RL: Optimal Policies

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

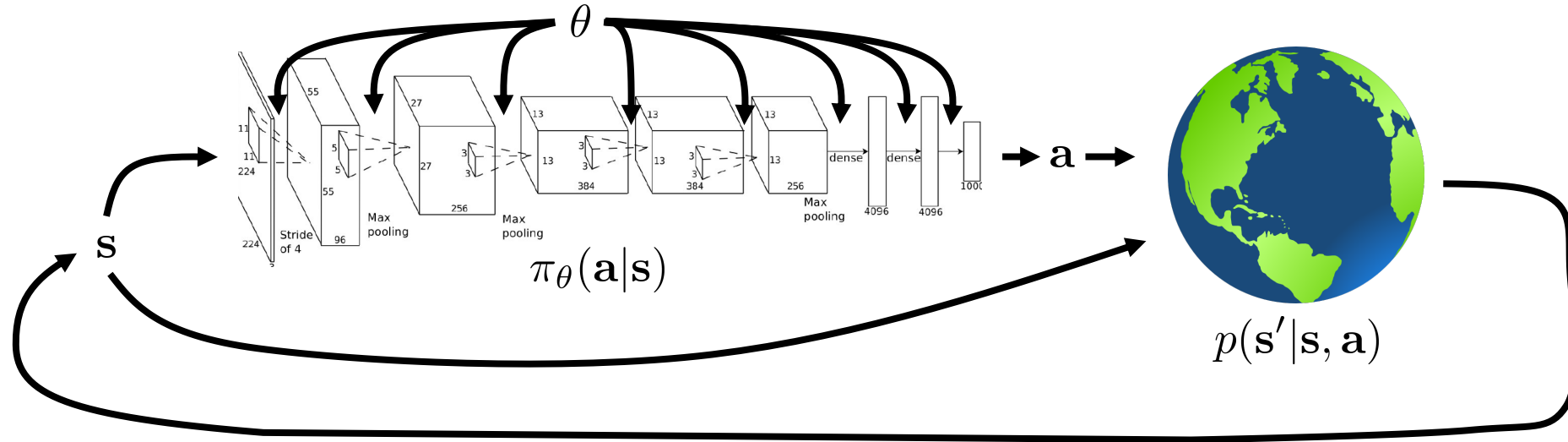
- “Optimal” \implies Following π^* maximizes expected return $\sum_t \gamma^t r_{t+1}$

Example optimal policy π^*



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

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**

π_{θ} 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:

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

(Non-optimal)
demonstrated data

(Non-optimal)
demonstrated actions

Reward returns: “how good was this action?” 

“Vanilla Policy Gradient”

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

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

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)

[Sutton, McAllester, Singh, Mansour 1999](#)



$$\mathbb{E} \left[\underbrace{r(z)}_{\text{unknown}} \right]$$

$z \sim \pi_\theta(z)$

$$\sum_t \gamma^t r_{t+1} \quad \text{unknown}$$

$$\pi_\theta(z) = \underbrace{\mu(s_0)}_{\text{unknown}} \cdot \pi_\theta(a_0 | s_0) \cdot P(s_1 | s_0, a_0) \cdot \pi_\theta(a_1 | s_1) \cdot P(s_2 | s_1, a_1) \cdot \dots \cdot \pi_\theta(a_n | s_n) \cdot P(s_{n+1} | s_n, a_n)$$

$$J(\theta) = \mathbb{E}_{z \sim \pi_\theta(z)} [r(z)] = \int \pi_\theta(z) r(z) dz$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(z) r(z) dz$$

$$= \int \pi_\theta(z) \nabla_\theta \log \pi_\theta(z) r(z) dz$$

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

$$= \mathbb{E}_{\pi_\theta(z)} [$$

